Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет информационных технологий и управления

Кафедра информационных технологий автоматизированных систем

О. В. Герман, Ю. О. Герман

Библиотека БГУИР

АДМИНИСТРИРОВАНИЕ И ПРОГРАММИРОВАНИЕ РАСПРЕДЕЛЕННЫХ ПРИЛОЖЕНИЙ

*Рекомендовано УМО по образованию
в области информатики и радиоэлектроники
в качестве учебно-методического пособия
для специальности 1-53 01 02
«Автоматизированные системы обработки информации»*

Минск БГУИР 2016

УДК 004.42(075)
ББК 32.973.26-018.2я73
Г38

Р е ц е н з е н т ы:

кафедра информатики и веб-дизайна
учреждения образования
«Белорусский государственный технологический университет»
(протокол №3 от 14.10.2015);

доцент кафедры компьютерных технологий и систем
Белорусского государственного университета,
кандидат физико-математических наук, доцент Л. А. Пилипчук

Герман, О. В.

Г38 Администрирование и программирование распределенных приложений : учеб.-метод. пособие / О. В. Герман, Ю. О. Герман. – Минск : БГУИР, 2016. – 240 с. : ил.

ISBN 978-985-543-231-0.

Содержит материалы лекций и лабораторных работ, а также краткое введение в систему программирования Java SDK. Рассмотрены вопросы создания компонентов на базе Java EE, создание web-сервисов, приложений JSF, SPRING, Hibernate, работа с сетевыми базами данных. Лабораторные работы содержат теоретическую часть, описание средств Java для решения поставленных задач и указания по выполнению работ.

УДК 004.42(075)
ББК 32.973.26-018.2я73

ISBN 978-985-543-231-0

© Герман О. В., Герман Ю. О., 2016
© УО «Белорусский государственный университет информатики и радиоэлектроники», 2016

Библиотека БГУИР

СОДЕРЖАНИЕ

1 РХИТЕКТУРА РАСПРЕДЕЛЕННЫХ СИСТЕМ 5

1.1 Распределенные системы 5
1.2 Сетевые протоколы...10
1.3 Реализация взаимодействий между сетевыми процессами 12
1.4 Установка и запуск web-сервера (сервера приложений) 16
1.5 Примеры взаимодействий по протоколу HTTP 24

2 КЛИЕНТ-СЕРВЕРНЫЕ ТЕХНОЛОГИИ 30

2.1 Простое клиент-серверное приложение на основе сокетов.......................30
2.2 Реализация соединения на основе протокола UDP 37
2.3 Высокоуровневые серверные приложения, обрабатывающие запросы к базе данных...41
2.3.1 Работа со встроенной базой данных Derby с примером на JSF 41
2.3.2 Взаимодействие Java-MySQL...63
2.3.3 Взаимодействие Java-Access 70
2.4 Работа с web-ресурсами 73
2.5 Сервлеты и jsp-страницы 84
2.6 Отправка электронной почты 97
2.7 Сериализация объектов и передача их по сети 102

3 РАСПРЕДЕЛЕННОЕ ПРОГРАММИРОВАНИЕ НА ОСНОВЕ КОМПОНЕНТОВ 115

3.1 Понятие компонента. Примеры компонентов. Создание библиотечного компонента.. 115
3.2 Реализация web-сервисов... 119
3.3 Компонентное программирование в Java EE 126
3.4 Служба JNDI 138
3.5 Современные технологии, использующие компоненты Hibernate, Spring 144
3.5.1 Технология Hibernate и персистентные классы 144
3.5.2 Технология Spring 159

4 ЛАБОРАТОРНЫЙ ПРАКТИКУМ...165

4.1 Работа с web-ресурсами 165
4.2 Работа в сети на основе сокетных соединений 170
4.3 Работа с сервлетами 176
4.4 Технология JSF.. 180
4.5 Создание web-сервисов... 188
4.6 Создание EJB-компонентов.. 196
4.7 Технология Model-View-Controller (Spring Java) 205

Библиотека БГУИР

4.8 Технология Hibernate 218

ПРИЛОЖЕНИЕ А (обязательное) Краткое введение в язык Java SDK 227

А.1 Установка Java и NetBeans 227
А.2 Объектные принципы Java 227
А.3 Обработка исключений 230
А.4 Потоки 230
А.5 Работа с файлами 234

ЛИТЕРАТУРА 239

Библиотека БГУИР

1 АРХИТЕКТУРА РАСПРЕДЕЛЕННЫХ СИСТЕМ

1.1 Распределенные системы

Под распределенными системами понимают компьютерные сети и параллельные вычислительные системы (последние в данном учебно-методическом пособии не рассматриваются). Компьютерные сети начали активно развиваться с конца 70-х − начала 80-х годов прошлого века. Значительную роль в этом сыграла глобальная сеть Интернет («выросшая» из сети `ArpaNet` (США)). В мае 1983 года Международная организация стандартов приняла документ 7498 «Базовая модель взаимосвязи открытых систем». Этот документ является стандартом, определяющим архитектуру современных сетей, услуги, которые должны предоставляться на каждом слое, и используемые протоколы передачи данных. В архитектуре различают семь уровней (слоев) [1–3]:

- физический;
- канальный;
- сетевой;
- транспортный;
- сеансовый;
- представления данных;
- программный.

На физическом уровне определяются требования к физической среде передачи данных (сетевой аппаратуре и линиям связи), электрическим сигналам и волновым сопротивлениям. В качестве линии связи может использоваться витая пара проводов, экранированный с помощью металлической оплетки электрический кабель, волоконно-оптический кабель для передачи световых сигналов, а также беспроводная линия связи на основе электромагнитных волн. Последние, собственно, и обеспечивают глобальный масштаб соединений в сети Интернет. На физическом уровне для кодирования сигналов широко используется так называемая манчестерская кодировка, в которой «1» соответствует перепад уровня сигнала, «0» – отсутствие перепада (рисунок 1).

Библиотека БГУИР

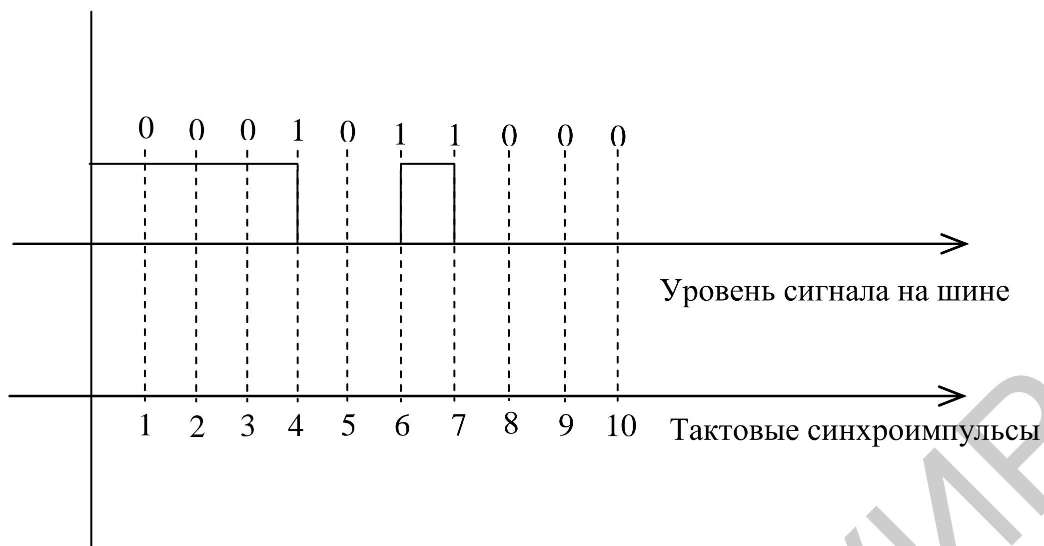


Рисунок 1 − Манчестерское кодирование

Проверка перепада уровня сигнала «привязана» к тактовым синхронизирующим импульсам. Манчестерское кодирование является более надежным по сравнению с кодированием, в котором «1» соответствует высокий уровень, а «0» – низкий (или наоборот).

На канальном уровне реализуется связь между двумя и более ЭВМ, подключенными к общему каналу (линии связи), а также определяются требования по организации соединения между приемопередающими узлами. Имеет место проблема синхронизации доступа к общему каналу при наличии многих абонентов. Используют различные стратегии доступа к каналу.

1 Методы, связанные с передачей маркера. Маркер представляет собой информационное сообщение, адресованное конкретному абоненту. При получении маркера компьютер-абонент становится «хозяином» линии связи, т. е. может осуществлять передачу информации любому другому абоненту. Завершив передачу, данный компьютер передает маркер другому компьютеру и т. д.

2 Метод прослушивания несущей частоты. Данный метод используется наиболее часто и основан на том, что каждый абонент прослушивает линию связи. Если передачи данных нет (слышна только несущая частота), то абонент начинает передачу информации. Однако может возникнуть ситуация, когда несколько узлов начнут передачу одновременно. В результате произойдет наложение одних данных на другие. ЭВМ-приемник не подтвердит в этом случае прием данных, и по истечении контрольного времени ЭВМ-передатчик установит наличие ошибки. ЭВМ-передатчик попытается повторно начать передачу, но через случайное время. В силу этого конфликтующие ЭВМ-передатчики повторно начнут передачу в разное время, что и разрешит проблему.

Библиотека БГУИР

Различают несколько типов сообщений на канальном уровне: сообщение-маркер, сообщение-данные, сообщение-прерывание.

Сообщение-данные, как правило, имеет следующий формат:

`<Преамбула><Начало кадра><Адрес получателя>`

`<Адрес отправителя><Длина сообщения>`

`<Контрольный код><Данные><Конец кадра>`

`Преамбула` указывает, что далее следуют сообщения. Начало каждого очередного сообщения устанавливает поле `<начало кадра>`.

Поля `<адрес получателя>` и `<адрес отправителя>` служат для идентификации ЭВМ-приемника и ЭВМ-передатчика. Поле `<длина сообщения>` задает число байтов передаваемых данных. Поле `<Контрольный код>` содержит контрольные проверочные разряды для обнаружения и исправления ошибок данных. В качестве контрольных кодов используют, например, циклические коды и др. Поле `<конец кадра>` устанавливает конец передаваемого кадра данных. Большие сообщения могут разбиваться на кадры, передаваемые по отдельности (факт деления на кадры фиксируется в специальных служебных флажках).

Обратимся к сетевому уровню. Сеть представляет собой множество каналов данных, соединяющих хосты, к которым подключаются локальные ЭВМ (рисунок 2).

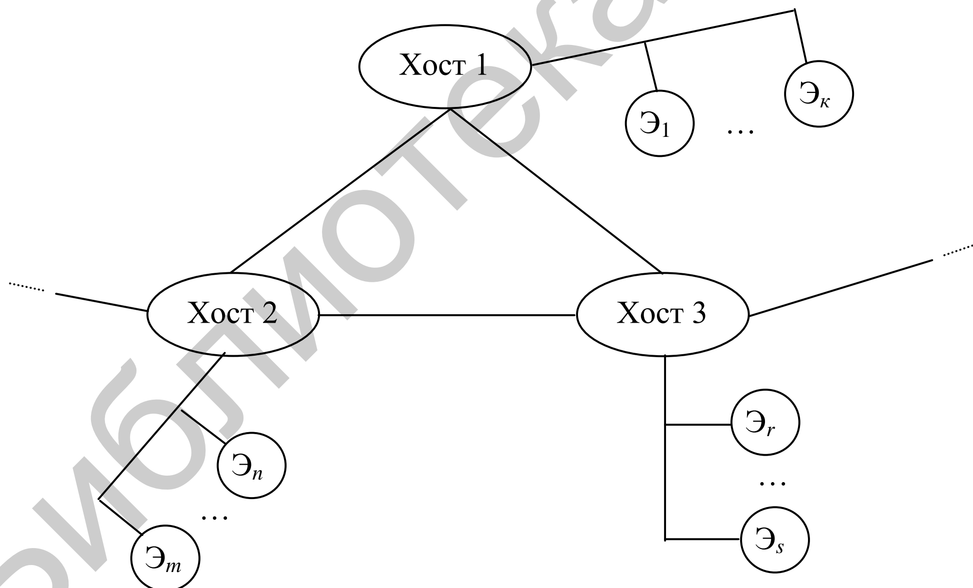


Рисунок 2 − Сеть ЭВМ

Хост играет роль машины-сервера, обслуживающей подключенные к ней локальные ЭВМ (Э$_i$) и связывающейся по сети с другими хостами. На сетевом уровне основной является задача управления трафиком сообщений и отслеживание их доставки. Различают два принципиально различных варианта управления трафиком: управление на уровне каналов (сеть с коммутацией каналов) и управление на уровне пакетов (сеть с коммутацией пакетов). При коммутации каналов между передающей и принимающей ЭВМ

Библиотека БГУИР

устанавливается фиксированный маршрут в сети. Это означает, что другие ЭВМ не могут задействовать ни один из каналов передачи данных, входящих в этот путь. При коммутации пакетов путь, по которому доставляется каждый пакет, определяется динамически с учетом загрузки каналов, поэтому разные пакеты могут доставляться по разным маршрутам.

Перейдем теперь к транспортному уровню. Этот уровень нужен, если имеются разнородные связанные через шлюзы сети передачи данных. Этот уровень обеспечивает, следовательно, согласование представления информации, принятого в той или иной сети, адресацию в рамках каждой сети и межсетевую адресацию.

Программный, представительный и сеансовый уровни относятся к локальной ЭВМ, ее операционной системе и программным приложениям, из которых выполняется запрос на установление сетевого соединения.

Задача сеансового уровня – организовать прием и передачу данных средствами операционной системы. Как правило, обмен выполняется с помощью фонового процесса, периодически просматривающего содержимое специальных ячеек памяти – портов, куда помещаются данные, предназначенные для отсылки абоненту или принимаются данные от абонента. На сеансовом уровне реализуется взаимодействие «клиент – сервер». Это взаимодействие выполняется в рамках одного сеанса. Представительский уровень предоставляет услуги по конвертированию данных в требуемый формат. Например, таким форматом, используемым в Интернете, является XML (для передачи маркированного специальным образом текста) или IMAP (для передачи почтовых сообщений) и др.

Наконец, программный уровень определяет набор средств по организации обмена с удаленными источниками в том или ином языке программирования. Так, язык Java дает возможность реализовать сокетное соединение (socket – порт) на основе технологии вызова удаленных модулей RMI или CORBA, а также вызов по адресной ссылке удаленных объектов – Enterprise Java Beans. Эти средства зависят от языка программирования и сами по себе требуют отдельного рассмотрения.

По типу и размерам различают сети LAN (local area networks) и WAN (wide area networks). WAN – сеть внешняя, глобальная, LAN – сеть внутренняя, локальная; WAN предназначена для интернет-соединений и объединяет другие локальные сети ЭВМ. Эти сети используют разные протоколы обмена. Протокол обмена определяет способ представления данных, интерпретацию (например, что это гипертекст формата HTML) и алгоритмы их обработки при передаче. Не вдаваясь в физические принципы сетевого обмена, отметим, что в настоящее время наиболее распространенным вариантом передачи является пакетная передача. Каждый пакет содержит собственно данные и управляющую информацию (адрес отправителя, адрес получателя, размер пакета, контрольные разряды для восстановления после ошибок, тип используемого протокола, управляющие флажки и др.).

Библиотека БГУИР

Общая концепция распределенных систем выстроена в системе «сервер – клиент». Серверное приложение предоставляет некоторый вид сервиса клиенту. Как правило, серверное приложение выполняется на другой машине, нежели клиентское приложение. Заметим, что машины имеют сетевые имена (номера), с помощью которых они могут направлять сообщения друг другу. Сетевое имя `localhost` (номер 127.0.0.1) характеризует собственный компьютер клиента. Сетевое имя не следует отождествлять с понятием `URL` – universal resource locator. `URL` определяет адрес ресурса в глобальной сети Интернет (причем ресурсом может быть файл с картинкой, например). При этом на одном и том же компьютере могут работать одновременно клиентское и серверное приложения. Из современных языков, ориентированных на сетевые взаимодействия, без сомнения, следует выделить `Java` и языки, объединенные в рамках системы программирования `.NET` (`c++.NET`, `c#.NET`, `j++.NET` и др.). Значительное распространение получили языки `PHP`, `Python` и `Ruby`. `Java` предоставляет технологическую платформу `J2EE` (Enterprise Edition), включающую более десятка фреймворков (`framework`) для реализации сетевых приложений. Под фреймворком понимают библиотеку модулей (классов). Назовем некоторые: `RMI`, `CORBA`, `JEE Beans`, `web-services`, `JSF`, `Sockets`, `Servlets`, `JSP`, `Ajax`, `JNDI`, `Java Mail`, `Spring` и др. (особенно в связи с нуждами мобильного программирования и облачных вычислений). В основном эти технологии рассмотрены в настоящем учебно-методическом пособии. Каждая сетевая технология имеет свою специфику и позволяет создавать определенный тип приложений. Они могут изучаться и использоваться независимо друг от друга. Особую роль играют распределенные объектные (компонентные) системы (в духе общей парадигмы объектно-ориентированного программирования и серверная, и клиентская часть реализованы как классы). Имеет место классическая триада: класс(ы) серверного приложения, класс(ы) клиентского приложения, посредник между ними – сервер приложений (который «по совместительству» может играть роль `web`-сервера). В системах сокетного типа сервера приложений нет, его функции берет на себя сетевая операционная система.

Сравнивая сокетные сетевые системы и распределенные объектные системы, можно отметить, что последние функционально значительно богаче. Такие системы можно наращивать, не меняя «внутренности» уже построенных классов и интерфейсов, в том числе их можно наращивать за счет использования библиотечных компонентов, написанных на других языках. Клиент распределенного приложения в локальной сети, очевидно, должен знать, где находится серверный объект и как установить с ним соединение. Для облегчения этой задачи в `Java` разработана служба имен (`JNDI – Java Naming Directory Interface`). Служба имен обслуживается специальным сервером. Объекты серверных классов хранятся в хранилище `JNDI` и доступны клиентам по их именам. Ранее требовалось также наличие у

Библиотека БГУИР

клиента файлов-заглушек с описанием классов сервера. Теперь служба имен снимает этот вопрос. Таким образом, сетевое взаимодействие существенно упрощается. Если речь идет о web-взаимодействии, то у клиента должен быть конфигурационный файл (файлы), в котором(ых) предоставлена информация о серверном объекте. Обычно серверные приложения пишут на одном компьютере, а потом их размещают (deploy) на другом. При этом серверные приложения «упаковываются» в специальные архивные файлы (war – для Интернета, ear – для локальных сетей). Таким образом, взаимодействия в локальных сетях и web строятся несколько различными способами. Поскольку при программировании распределенных систем должны быть приняты во внимание многие технические вопросы (настройка сервера приложений, правильное написание конфигурационных файлов, получение доступа к объектам сервера и их размещение (deployment)), то это делает написание таких систем достаточно сложным. Дело усугубляется также и наличием различных технологических платформ. Мы берем за основу язык Java.

Имеется несколько сред (IDE – Integrated Development Environment) для создания Java-приложений, включая распределенные приложения. Разумеется, можно писать распределенные приложения и через блокнот (NotePad), запускать и развертывать их с помощью специальных утилит (в этом отношении непревзойденной утилитой является ANT). Однако такое программирование очень трудоемко и не годится для учебных целей. Наиболее популярны такие IDE, как NetBeans и Eclipse (например версии Juno). Здесь за основу взята платформа NetBeans, которая содержит многие необходимые библиотеки и сервер приложений GlassFish. Это не значит, что нельзя использовать другие серверы, например Apache TomCat или JBoss. Серверы приложений можно добавлять достаточно просто с помощью соответствующего мастера. Каждая новая версия NetBeans (как, впрочем, и Java) привносит некоторые отличия по сравнению с предыдущей. Базовая версия этого учебного материала NetBeans 7.1.2, Java 7.

1.2 Сетевые протоколы

Описанные выше уровни OSI поддерживаются своими протоколами [3, 4]. В глобальной сети Интернет данные передаются на основе протокола IP (Internet Protocol). Этот протокол действует в рамках различных соединенных друг с другом сетей ЭВМ (обеспечивает межсетевую доставку пакетов). IP-пакеты, называемые также IP-датаграммами, имеют следующий формат:

- версия;
- длина заголовка;

Библиотека БГУИР

- тип сервиса;
- общая длина пакета;
- идентификация;
- версия;
- флажки;
- время доставки;
- тип транспортного протокола;
- адреса отправителя и получателя;
- данные;
- контрольная сумма.

Само сообщение состоит из заголовка и собственно данных. Заголовок содержит служебную информацию. Так, поле `тип сервиса` определяет специфические условия обслуживания пакета (срочность, повышенную надежность, значимость). Поле `идентификация` указывает номер пакета в сообщении (сообщения при пакетной обработке разбиваются на пакеты, которые доставляются по сети независимо друг от друга). Поле `флажки` задает дополнительные биты настройки, например определяет, можно ли данный пакет разбивать на еще меньшие по размеру пакеты. Поле `время доставки` указывает максимально допустимое время доставки пакета адресату. Значение этого поля постоянно уменьшается в сетевой аппаратуре и когда становится нулевым или отрицательным, пакет уничтожается. Поле `тип транспортного протокола` задает версию транспортного протокола, который использует протокол `IP` при передаче пакетов данных внутри локальной сети.

Различные транспортные протоколы имеют уникальные номера, например транспортный протокол `TCP` имеет номер 6. `IP`-адрес представляет собой 4-байтовое поле и обычно представляется как четыре десятичных числа, разделенные точкой, например 196.168.0.2. Для удобства пользователя этот числовой формат часто заменяют строковым представлением адреса, например www.oracle.org.

Одним из важнейших сетевых протоколов является `TCP` (Transmission Control Protocol). Этот протокол обеспечивает двунаправленное взаимодействие клиента и сервера. Данные передаются в форме массивов байтов, так что принципиально можно передать любой тип информации (картинки, упакованные (сериализованные) объекты или исполняемые коды), если предварительно сохранить передаваемый объект в массиве байтов. Протокол `TCP` действует в связке с протоколом `IP` (на транспортном уровне используется протокол `TCP`, на сетевом – `IP`) . Наряду с `TCP` на транспортном уровне применяют также протокол `UDP`.

Протокол `TCP` обеспечивает надежное соединение, основанное на подтверждении принимающей стороной полученных отправлений.

Библиотека БГУИР

Для отправки и доставки почты используют протокол SMTP (Simple Mail Transfer Protocol). Почтовый клиент взаимодействует с почтовым сервером обычно через порт 25. Протокол SMTP позволяет подключать к письмам вложения (файлы). Между почтовым клиентом и сервером происходит обмен командами и подтверждениями. Для установления связи почтовый клиент отправляет команду MAIL. Для передачи почты в конкретный почтовый ящик используется команда RCPT. Почтовые сообщения имеют строгий формат и содержат поля Date, Subject (тема), To (адрес получателя), From (адрес отправителя) и др.

Другим почтовым протоколом является протокол POP3. Этот протокол позволяет считывать содержимое почтовых ящиков для их обработки в прикладных программах.

Для передачи содержимого сайтов используют протоколы Http и Https (последний обеспечивает шифрование передаваемых данных). Клиентские программы типа Opera, Mozila или Chrome «понимают» формат сообщений с этим типом протокола и открывают окна, в которых отображают содержимое интернет-документов. Http относится к протоколам прикладного уровня, как и SMTP. Для идентификации ресурсов Http использует адреса URL.

Имеется множество других протоколов сообразно задачам и уровням передачи данных открытой модели.

1.3 Реализация взаимодействий между сетевыми процессами

Сетевые процессы взаимодействуют друг с другом посредством приема/отправки сообщений [2, 3]. Один из этих процессов является серверным процессом. Его особенность в том, что он (или его окружение) всегда активно. Так, если рассматривать классическую клиент-серверную технологию, то серверный процесс запускается и «висит» на определенном порту, прослушивая подключения клиентов (клиентов может быть более одного). Если рассматривать технологию распределенного (объектного) программирования, то серверное приложение запускается сервером приложений (или web-сервером или приложением, одновременно выступающим как web-сервер и сервер приложений, например GlassFish Java). Здесь постоянно активен сервер приложений, настроенный на определенный порт. Сервер приложений должен знать, где искать запрошенное приложение. Такая информация помещается в конфигурационном файле. Среда разработки, как правило, содержит встроенный сервер приложений и кроме него имеется еще автономный

Библиотека БГУИР

сервер приложений. Так обстоит дело, например, в `GlassFish Java`. У этих серверов конфигурационные файлы лежат в разных папках. При отладке мы пользуемся встроенным сервером приложений.

Сервер приложений должен правильно интерпретировать передаваемые от клиента сообщения, т. е. правильно распознавать их тип. Например, сообщение может представлять собой данные клиентской `html`-формы, либо почтовое отправление, либо упакованный объект. Тип сообщения определяется протоколом и заголовочной управляющей информацией. Каждому сообщению предшествует заголовок, который и доставляет необходимую служебную информацию. Известны различные протоколы, используемые при передаче сообщений. В локальных сетях наиболее употребительны `TCP` (Transmission Control Protocol) и `UDP` (User Datagram Protocol). Эти протоколы обеспечивают всю необходимую информацию для организации сетевого обмена между ЭВМ. Для передачи электронной почты используют протоколы `SMTP`, `POP3`, `IMAP` и др. Для взаимодействия между распределенными сетевыми приложениями используют протокол `SOAP` (Simple Object Access Protocol).

На приведенном ниже скриншоте (рисунок 3) представлены параметры заголовка сообщения, выведенные из `java`-сервлета.

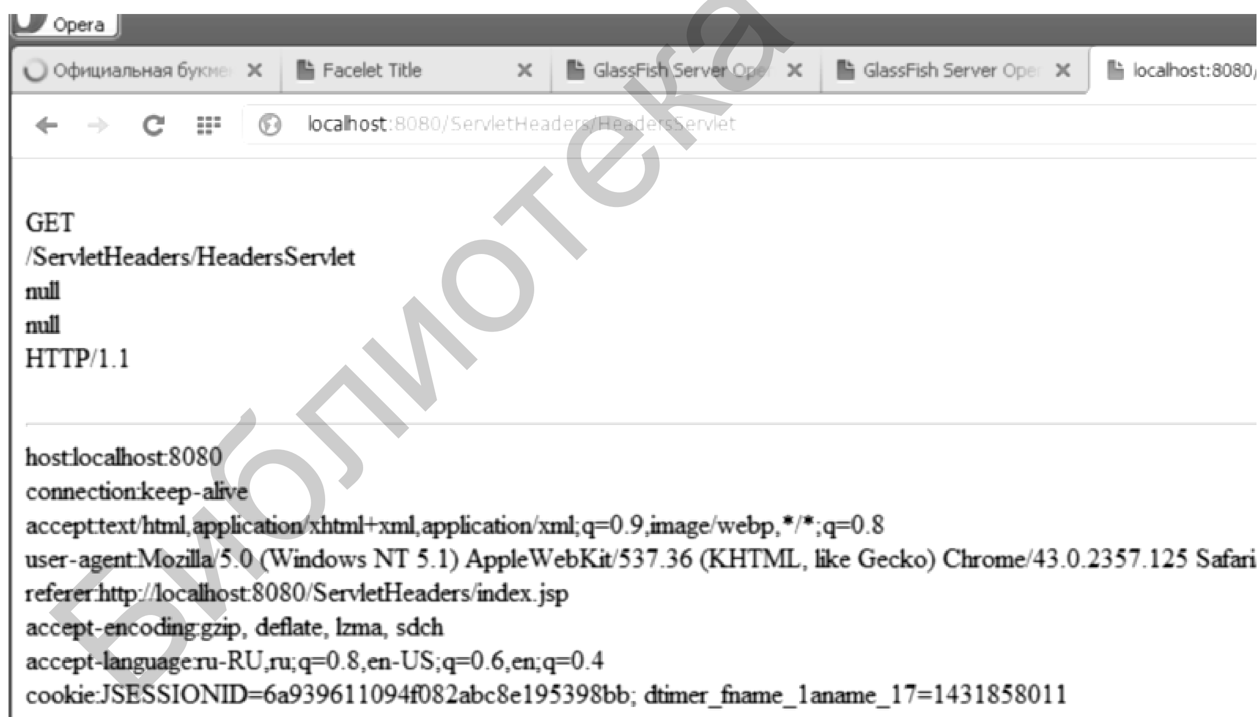


Рисунок 3 − Параметры заголовка сообщения

Здесь отмечено, что методом передачи данных является `Get` (управляющая часть и данные сообщения передаются в одном пакете). Приложение, которое обрабатывает сообщение, есть `/ServletHeaders/HeadersServlet`. Сообщение передается по протоколу

Библиотека БГУИР

http (этот протокол используется для передачи гипертекста). Компьютер, которому направлено сообщение, есть hostlocalhost:8080.

Источником сообщения является referer:http://localhost:8080. Сообщение передается в формате архива gzip. Сообщение допускает текст с тегами html, xml и может содержать картинки (image). В тексте могут быть русские и английские слова (accept-language). Для передачи можно использовать клиентские программы-браузеры, включая Chrome.

Принципиально следует различать два варианта сетевых технологий: традиционные сокетные клиент-серверные технологии и технологии на основе распределенных объектов (компонентов).

На рисунке 4 показаны две действующие стороны: сторона клиента и сторона сервера. На стороне клиента клиентское приложение представлено клиентским сайтом, который может содержать Java-апплет (программу, реализованную на Java). Клиентское приложение посредством браузера Internet Explorer (или иным) связывается с приложением Web-сервер на стороне сервера. Стандартным Web-сервером, входящим в инсталляционный пакет Windows, является IIS. Java работает с серверами GlassFish, TOMCAT, WebSphere и др. Web-сервер вызывает приложение сервера.

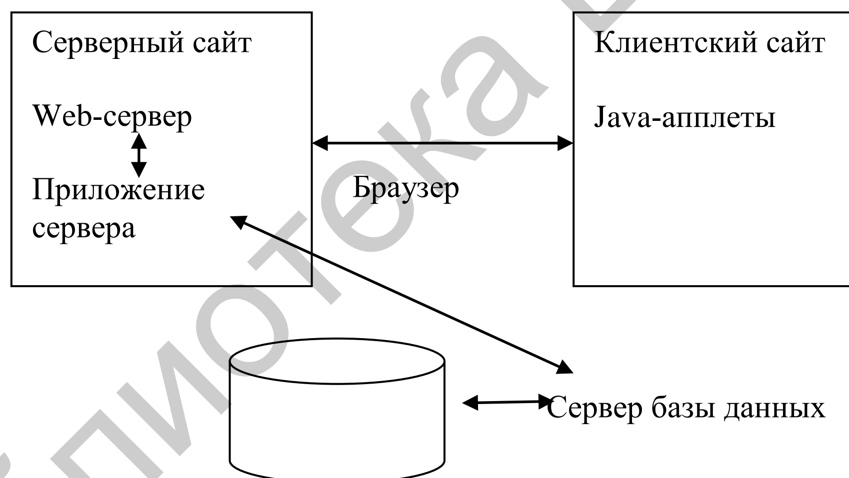


Рисунок 4 – Базовая схема клиент-серверного взаимодействия

На ранних этапах интернет-программирования приложения сервера писали на Perl, Delphi, C++. Это были самостоятельные приложения (exe-модули), которые назывались скриптами, а сама технология – CGI-программированием (Common Gate Interface). Впоследствии CGI-скрипты были вытеснены приложениями на Java (сервлеты и бобы – Enterprise Java Beans) и их аналогами на c# и других языках среды .NET. Получил распространение также язык PHP. CGI-скрипты не обеспечивают устойчивость связи. При ошибке в соединении или в скрипте возникала необходимость в повторной установке соединения. Среда выполнения приложений Java и c# (run time environment) обеспечивает сохранение соединения даже в случае ошибок в программе. Кроме того, CGI-приложение

Библиотека БГУИР

должно стартовать заново каждый раз при установлении связи между клиентом и сервером, что приводит к снижению скорости работы распределенного приложения.

Итак, web-сервер вызывает серверное приложение, которое получает от клиентской формы соответствующие данные в соответствующем формате. Таким форматом является, как правило, текст, представленный в протоколе http. Достаточно часто серверное приложение далее использует обращения к процедурам или SQL-запросам к базе данных. Таким образом, на стороне сервера участвует еще одно действующее лицо – сервер базы данных, например, MS SQL Server, MySql или Oracle. Серверное приложение, как принято говорить, реализует бизнес-логику, а клиентское служит только в качестве интерфейса для отображения результатов работы бизнес-логики. При таком разделении ролей клиента называют «**тонким**». Если на клиента возложить реализацию бизнес-задач или какой-то части этих задач, то клиент называется в этом случае «**толстым**».

Термин «бизнес-логика» означает совокупность различных функциональных процедур обработки данных на стороне сервера, связанных с задачами, возложенными на систему в целом. Бизнес-процедурой может, например, быть простая программа, рассчитывающая налог от совокупного дохода.

Фирмы Sun MicroSystems и MicroSoft создали новые технологии программирования распределенных объектов, а также web-сервисов. Распределенным объектом является экземпляр некоторого класса, помещенный, как говорят, в контейнер. В Java такие распределенные объекты называются **бобами** (beans). По существу, распределенный объект хранится как двоичный файл в системной базе данных. Этот объект представляет собой экземпляр какого-то класса и содержит набор методов для реализации бизнес-логики. Клиент выставляет запрос на создание в своем приложении удаленного объекта с тем, чтобы использовать его методы. Этот запрос передается на сторону сервера – так называемому серверу приложений (на рисунке 4 он не показан). Задача сервера приложений состоит в построении экземпляра класса, загрузки его в оперативную память и возврате клиенту адресной ссылки на созданный экземпляр. Таким образом, приложение клиента работает с удаленным объектом так, как будто он находится в адресном пространстве клиентского приложения.

Для доступа к удаленному объекту, хранящемуся в контейнере, клиентское приложение реализует интерфейс связи именно с контейнером, а не с объектом, что обеспечивает нужную защиту и ограничение прав доступа для различных категорий пользователей.

Web-сервисы – это заранее созданные байт-коды (классы), размещенные на стороне сервера. Доступ к ним реализуется через конфигурационные файлы XML, в которых описывается структура

Библиотека БГУИР

соответствующего класса (методы и переменные). Таким образом, web-сервисы – это не удаленные объекты (как бобы), а удаленные классы, связь с которыми осуществляется по протоколу XML. Пользователь может объявлять удаленные классы в своем клиентском приложении и использовать их как и другие классы.

1.4 Установка и запуск web-сервера (сервера приложений)

Существует множество web-серверов. В частности, поскольку мы имеем дело с Java, то web-серверами, поддерживающими приложения (классы) Java, являются, например, Apache TomCat, GlassFish, JBoss, Web Sphere и др.

В этом учебно-методическом пособии мы ориентируемся на среду разработки Java NetBeans (версии 7.2 и выше). Эта среда располагает встроенным сервером GlassFish [5]. При установке IDE Java NetBeans автоматически устанавливается сервер GlassFish. Чтобы проверить его наличие в системе откройте вкладку Службы (Services) и затем Серверы (Servers). Увидите следующий скриншот (рисунок 5).

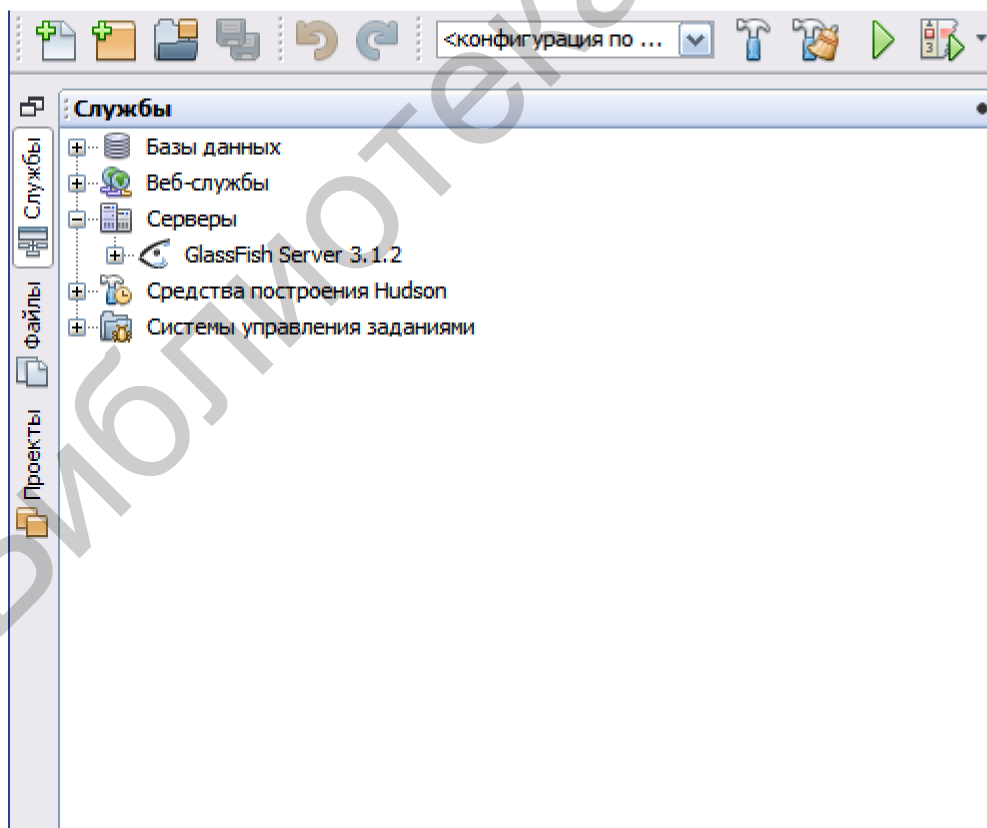


Рисунок 5 – Вкладка Службы

Библиотека БГУИР

Откройте свойства (`properties`) сервера. Вы можете добавить или удалить имеющийся сервер, используя соответствующие кнопки (рисунок 6).

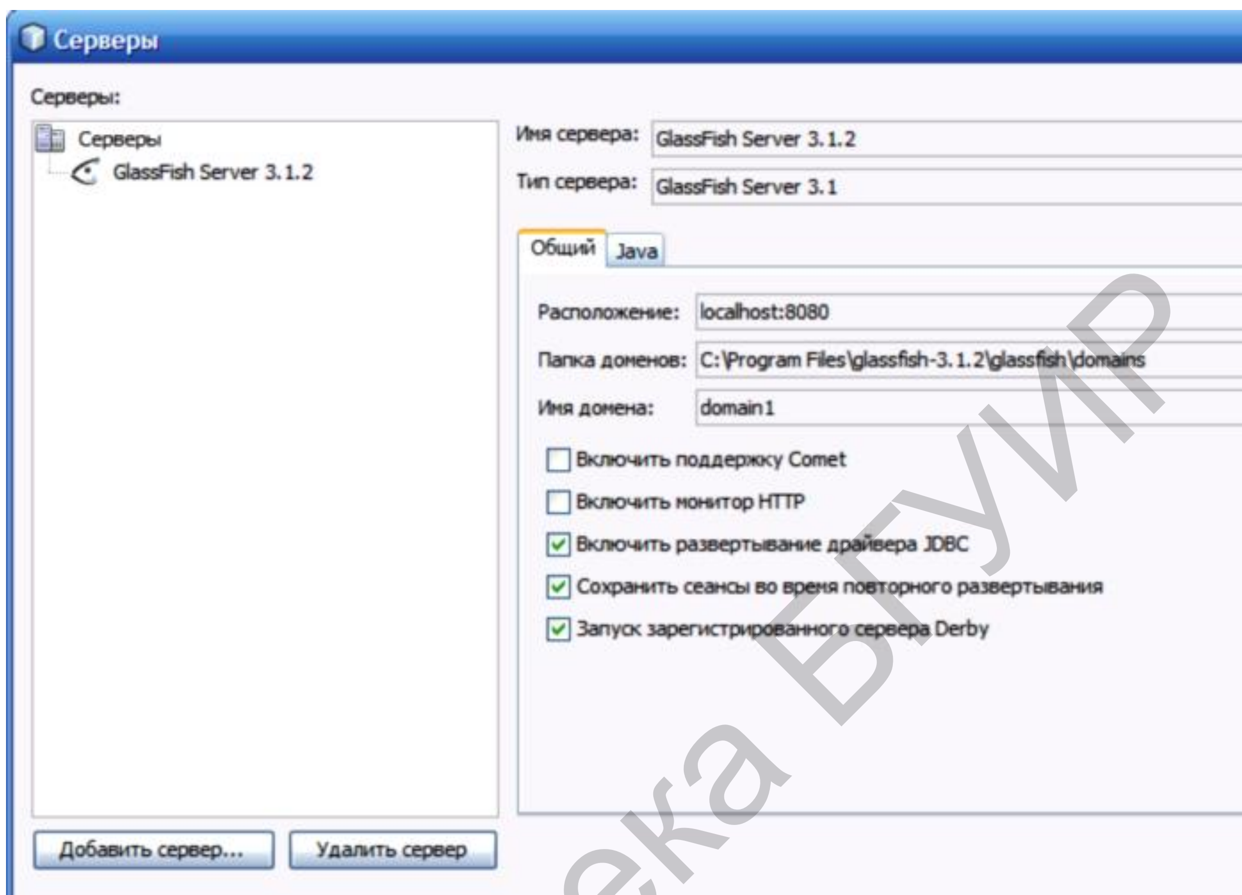


Рисунок 6 – Окно `Свойства` сервера

Обратим внимание на папку доменов: `C:\Program Files\glassfish-3.1.2\glassfish\domains`. Откроем домен `C:\Program Files\glassfish-3.1.2\glassfish\domains\domain1`

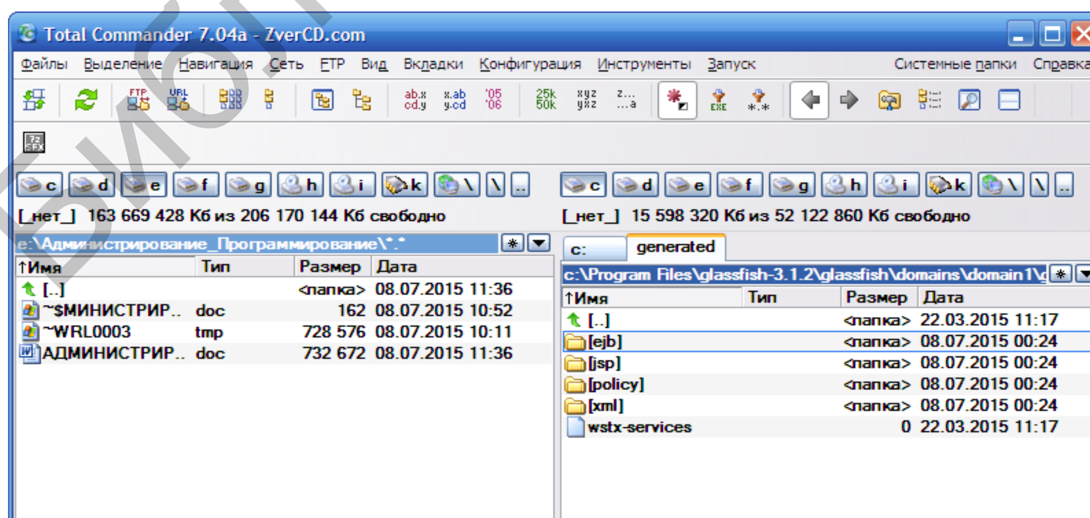


Рисунок 7 – Папка доменов сервера

Библиотека БГУИР

Мы видим папки ejb, jsp. В этих папках находятся web-приложения, такие как страницы jsp, сервлеты, компоненты ejb.

Для того чтобы сервер GlassFish запустил web-приложение, оно должно быть развернуто, т. е. расположено в том месте, где его будет искать сервер. IDE NetBeans позволяет развертывать приложения автоматически (место развертывания определяется конфигурационным доменным файлом) либо вручную, например, с помощью специальной утилиты ANT. Конфигурационный файл домена находится в папке C:\Program Files\glassfish-3.1.2\glassfish\domains\domain1\config\domain.xml. Чуть ниже мы приведем его контент в нужной нам части.

Запуск сервера GlassFish можно выполнить из среды NetBeans, а также из консольного окна. В последнем случае откройте консольное окно (Пуск, Выполнить, cmd). Наберите

```
asadmin start-domain --verbose
```

Получите следующий консольный вывод, подтверждающий запуск сервера (рисунок 8).

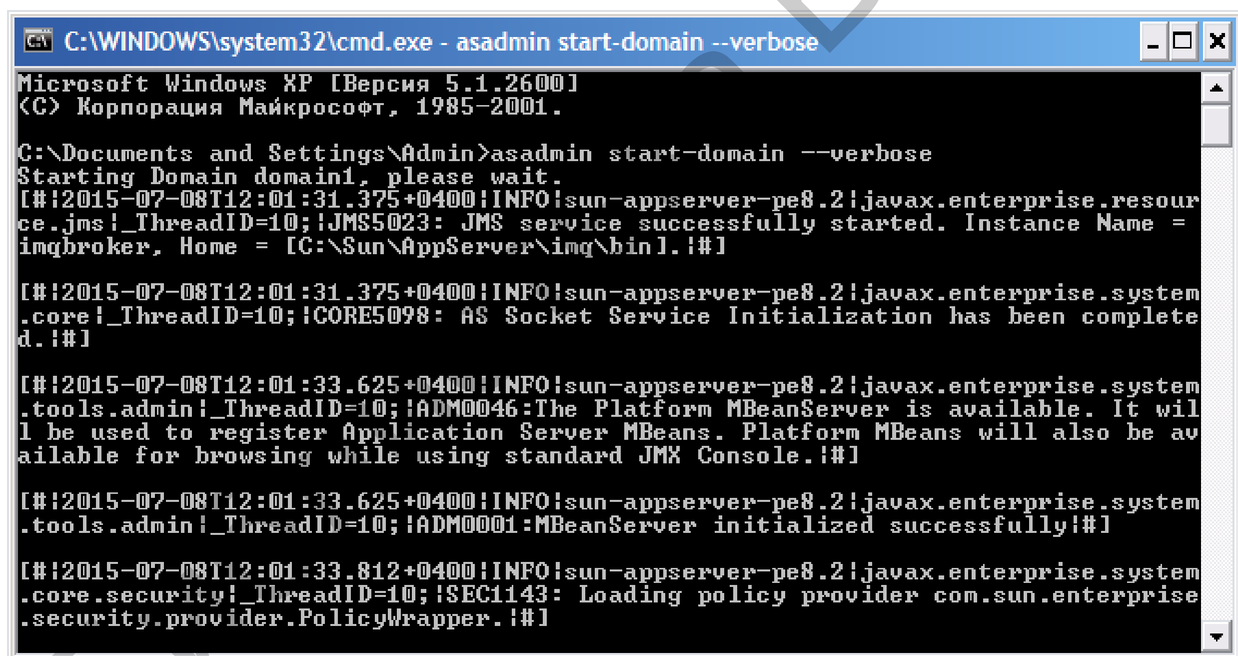


Рисунок 8 – Консольное окно сервера GlassFish

Без установленного сервера GlassFish нельзя создать распределенное web-приложение.

Создадим простое web-приложение в среде Java NetBeans 7.2 (рисунок 9). В главном меню выберем Файл – Создать проект – Java web (рисунок 10).

БГУИР

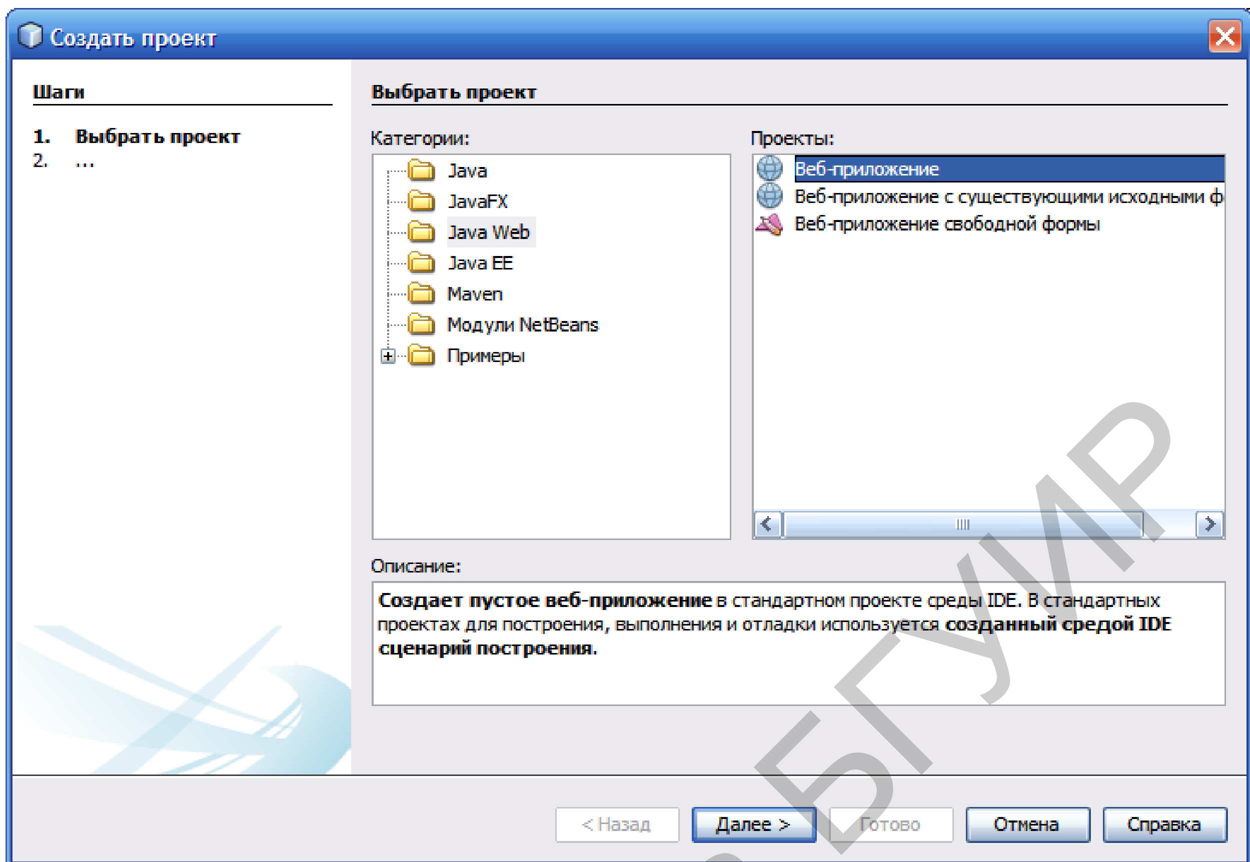


Рисунок 9 – Тип приложения – web–приложение

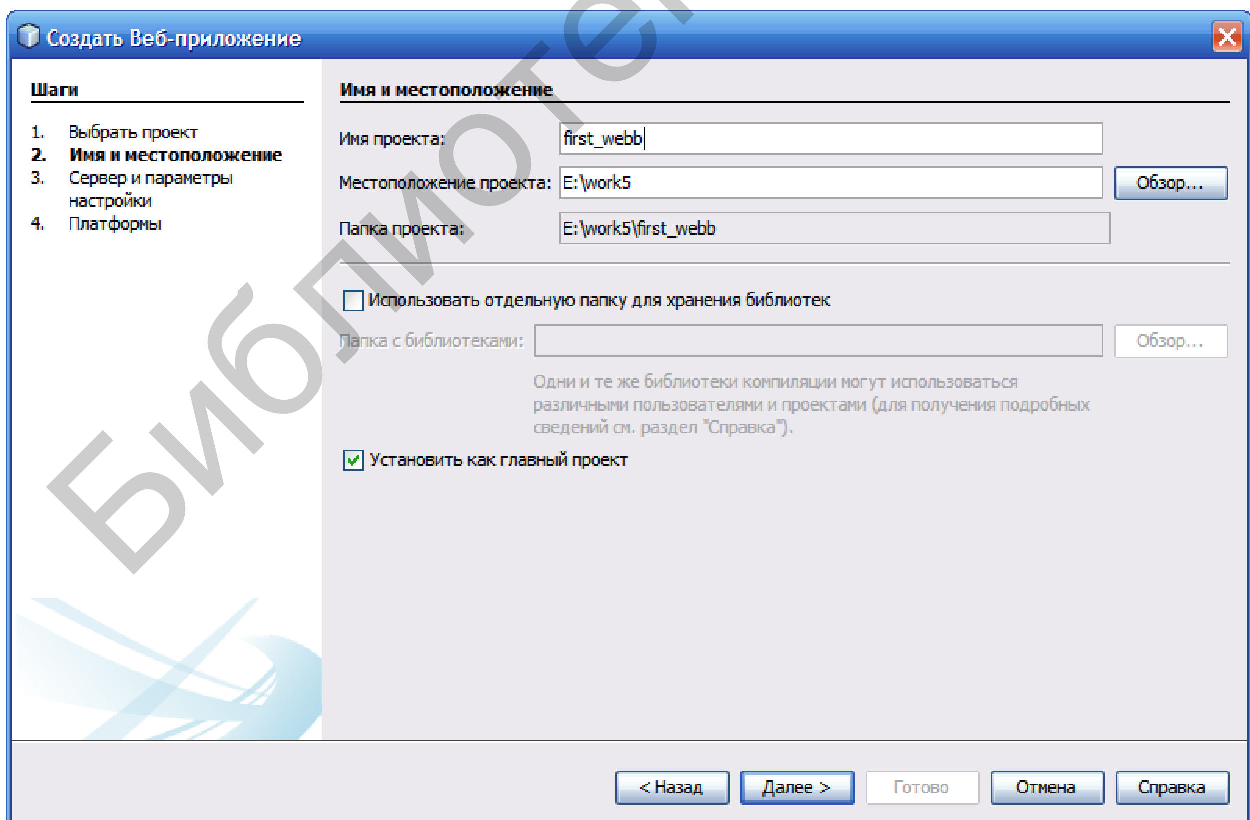


Рисунок 10 – Задание имени приложения и его дислокации

Библиотека БГУИР

Зададим имя проекта first_web. В дереве проектов это простейшее приложение будет содержать единственный файл index.jsp. Откроем его двойным щелчком и изменим содержимое, как показано ниже:

```
<%--
    Document   : index
    Created on : 08.07.2015, 22:36:56
    Author     : Admin
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta    http-equiv="Content-Type"    content="text/html; 
charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body bgcolor="#aabbff">
        <h2>Hello World!This is the first web application</h2>
    </body>
</html>
```

Теперь построим проект, щелкнув правой кнопкой мыши на его имени и выбрав пункт контекстного меню Очистить и построить. В результате этих действий будет построен архивный файл first_web.war (рисунок 11). Именно этот war-файл и запускается сервером GlassFish.

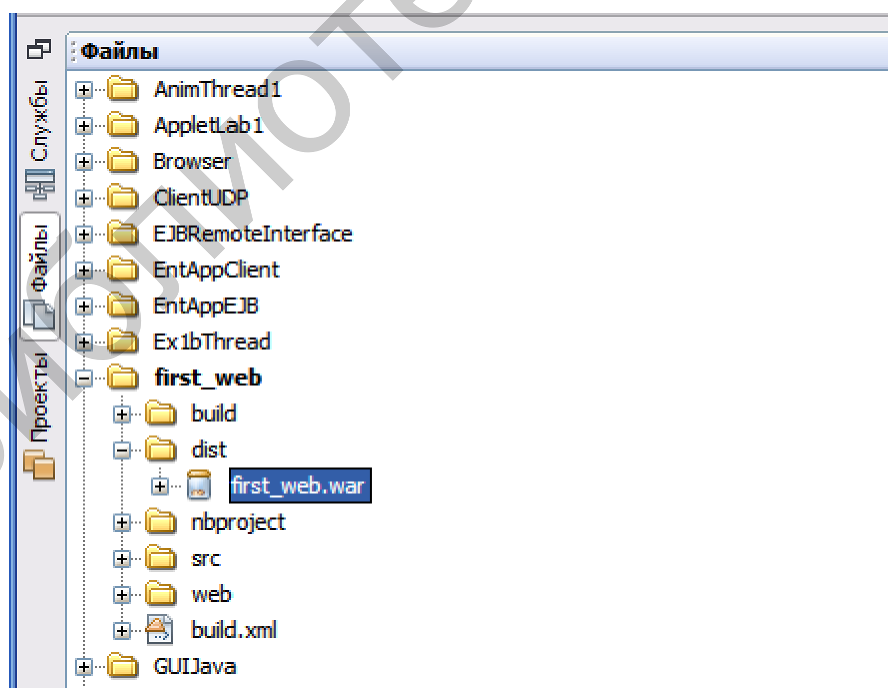


Рисунок 11 – Дислокация архивного файла war

Библиотека БГУИР

Посмотрим, как вручную запустить этот файл. Запустим `GlassFish`, затем откроем окно `Chrome`. Введем `url: localhost:4848`. Откроется окно сервера (рисунок 12).

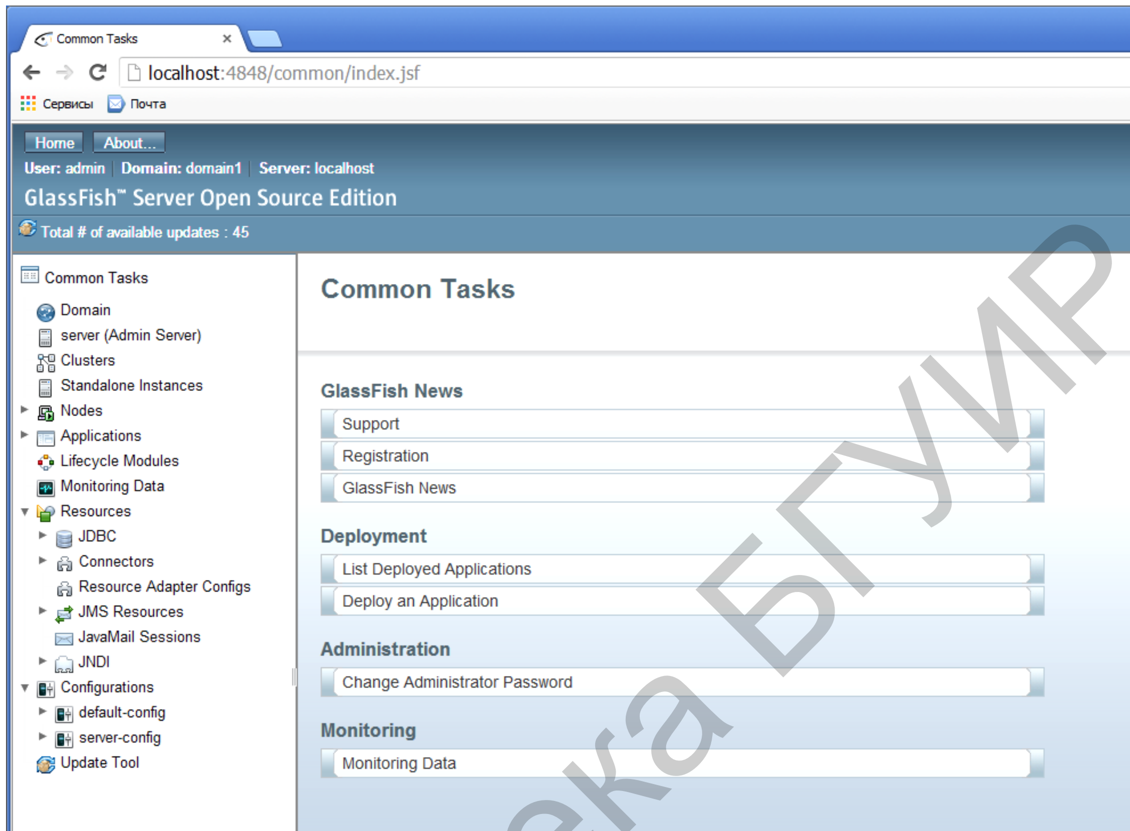


Рисунок 12 – Окно сервера `GlassFish`

Выберем пункт `Deploy an Application`. После этого нажмем кнопку `Выберите файл`. Получим следующий экран (рисунок 13).

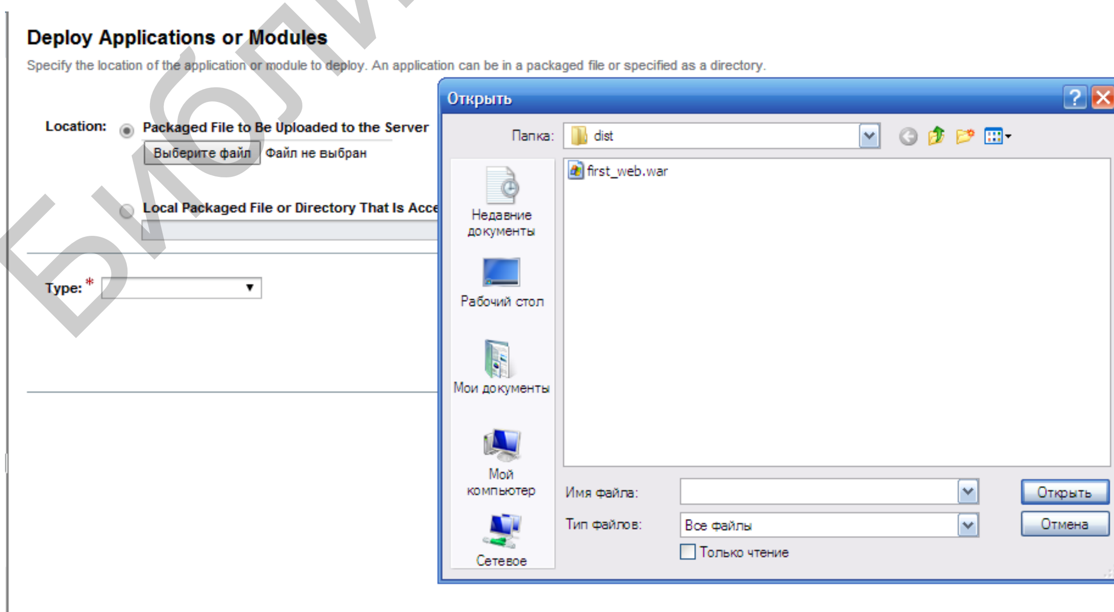


Рисунок 13 – Выбор архивного файла

Библиотека БГУИР

Находим созданный файл `first_web.war` и нажимаем кнопку `Открыть`. В результате этих манипуляций приложение будет развернуто. Другими словами, оно будет помещено в папку, в которой его будет искать сервер для запуска.

Чтобы выполнить развернутое приложение, зайдем в меню `Applications` (рисунок 14) и запустим наше приложение (выделим в левом окне наше приложение `first_web`, а в правом – нажмем `launch`).

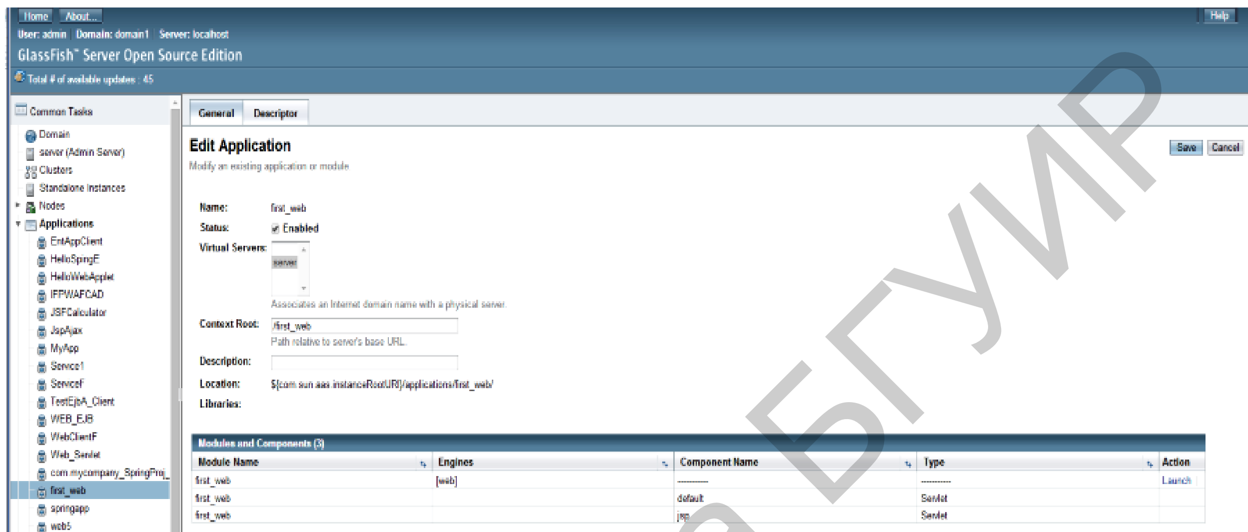


Рисунок 14 – Выбор развернутого приложения для запуска

Результат будет таким, как показано на рисунке 15.

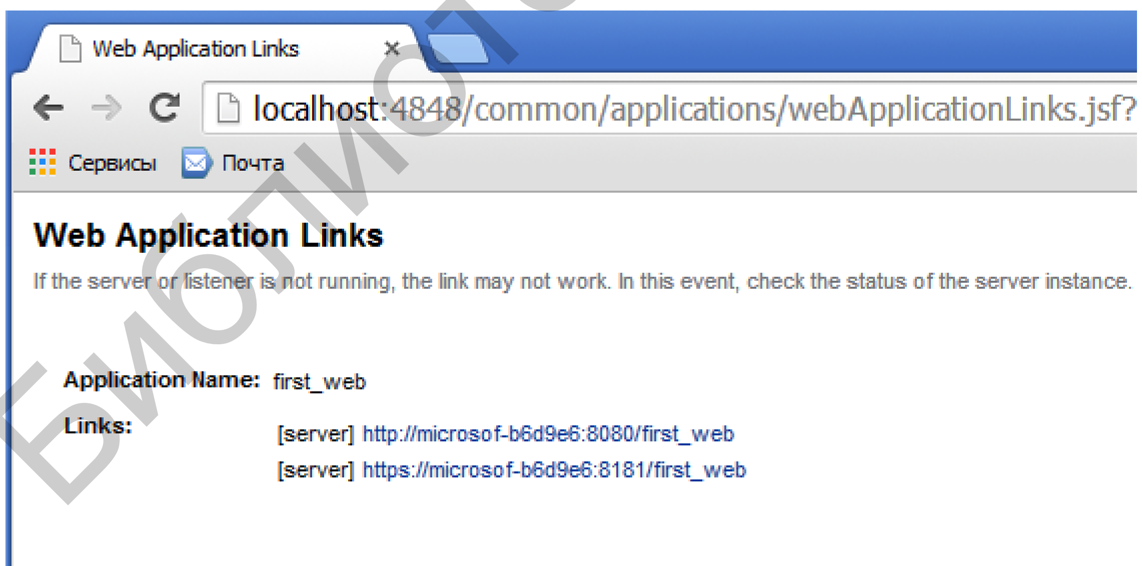


Рисунок 15 – Переход по ссылке к приложению

Щелкнем левой кнопкой мыши на первой из ссылок. Откроется окно нашего документа (рисунок 16).

Библиотека БГУИР

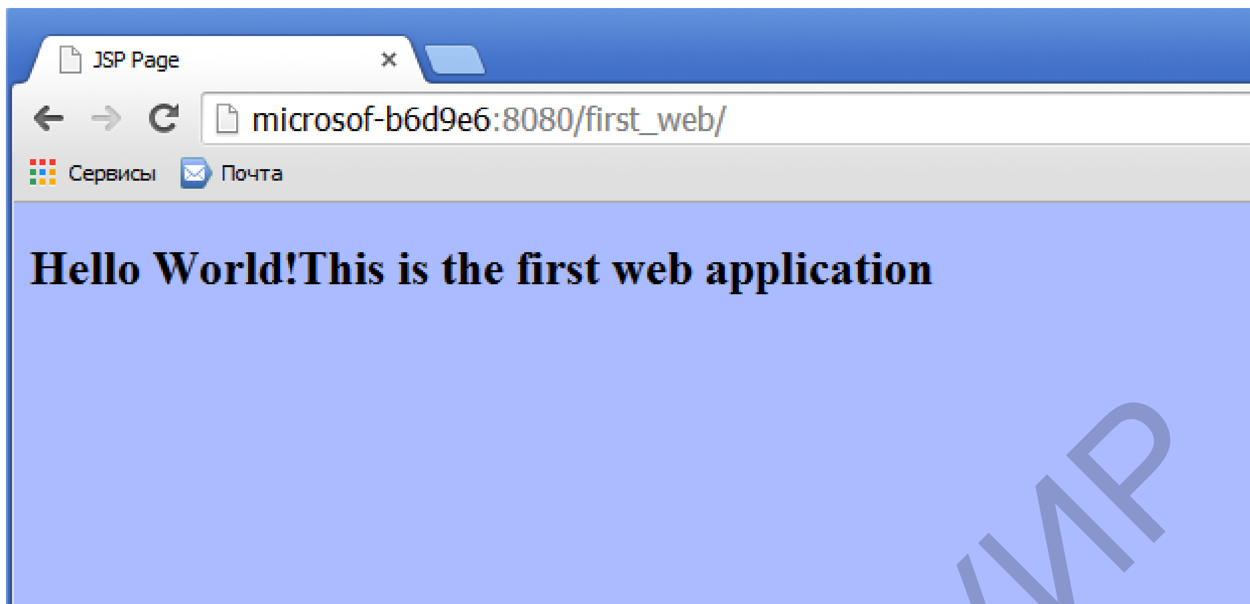


Рисунок 16 – Окно работающего приложения

Этого же результата можно достичь, если открыть URL http://microsof-b6d9e6:8080/first_web, где напрямую указывается сетевое имя компьютера.

Остается еще раз рассмотреть доменный конфигурационный файл:

```
C:\Program Files\glassfish-
.1.2\glassfish\domains\domain1\config\domain.xml.
```

Покажем фрагмент этого файла, относящийся к нашему приложению:

```
  <application context-root="/first_web"
location="file:/E:/work5/first_web/build/web/" name="first_web"
directory-
       deployed="true" object-type="user">
      <property name="appLocation"
value="file:/E:/work5/first_web/build/web/" />
      <property name="defaultAppName" value="web" />
   <module name="first_web">
   <engine sniffer="security" />
      <engine sniffer="web" />
   </module>
   </application>
```

Здесь прямо указано место развертывания:

```
location="file:/E:/work5/first_web/build/web/".
```

Если изменить данное место на другое, то из среды IDE GlassFish запустить web-приложение не удастся.

Библиотека БГУИР

1.5 Примеры взаимодействий по протоколу HTTP

Проще всего продемонстрировать взаимодействие между сайтом (клиентской частью распределенного приложения) и сервлетом (серверной частью распределенного приложения), представлющим класс Java. Запустим Java NetBeans. Выберем пункт Файл – Создать проект – web-приложение (как мы делали ранее). Назовем проект ServletHeaders. Минимальное web-приложение будет состоять из одного файла index.jsp. Файл index.jsp – это серверный файл типа java server page. Он представляет собой документ html со встроенными командами языка Java. Полагая, что читателю известны основные команды языка разметок html (см., например, [6–8]), изменим содержимое этого файла таким образом:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
 <meta       http-equiv="Content-Type"       content="text/html;
charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Study net messaging
<a href="<%=request.getContextPath()%>/HeadersServlet">Call
the servlet</a></h1>
    </body>
</html>
```

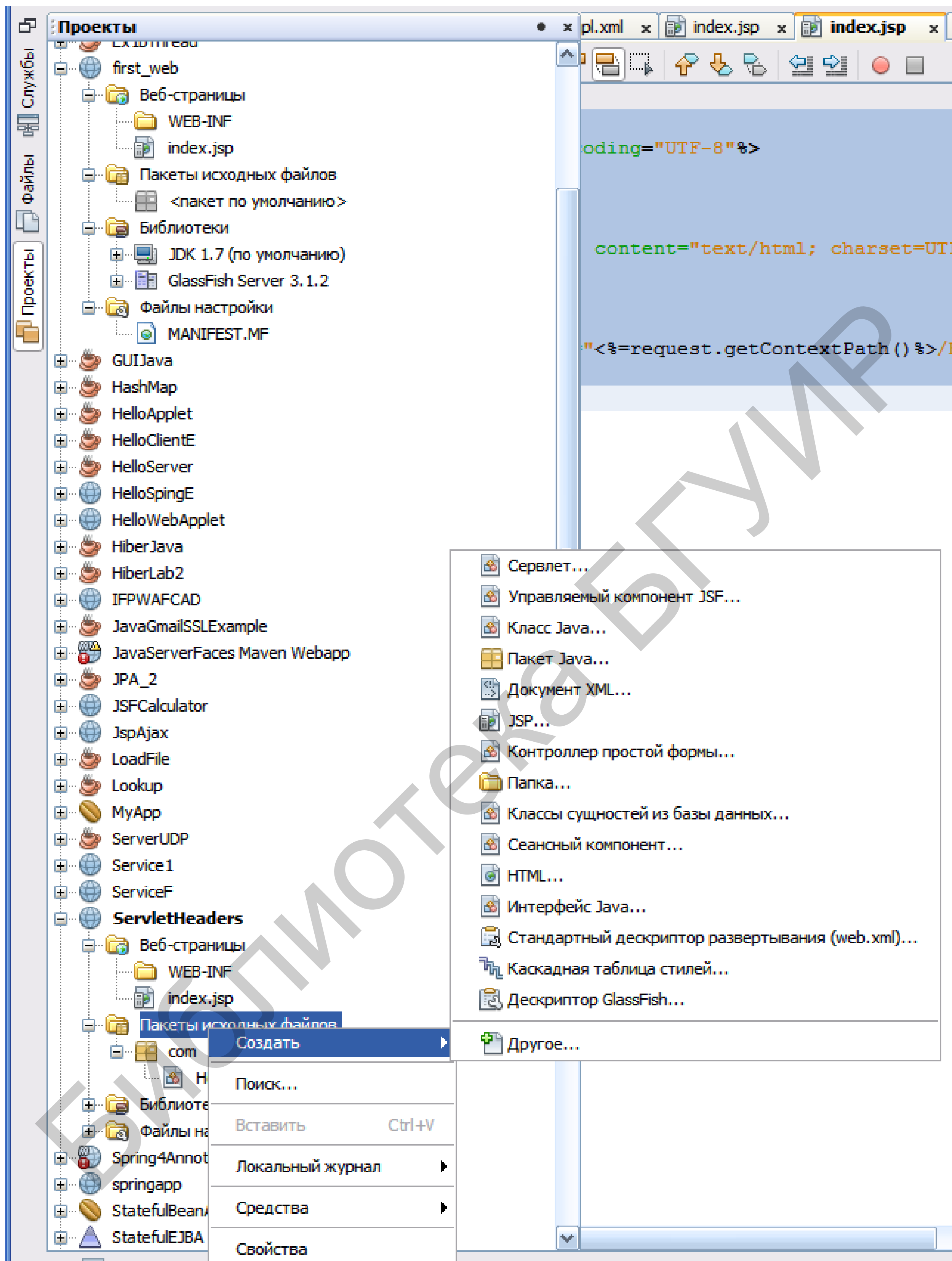
Здесь вставлена одна-единственная команда языка Java:

```
%=request.getContextPath()%
```

Вставка обрамляется слева и справа знаком процента. Объект request является стандартным объектом класса HttpServletRequest и позволяет получать данные, передаваемые в сообщении от клиента. В данном примере получаем путь к корневому каталогу, где расположен файл index.jsp в проекте ServletHeaders.

Сервлет представляет собой программу на языке Java [6, 8–10]. Программы на Java называют классами (они имеют расширение class). Нам нужно создать сервлет. Для этого вызываем контекстное меню щелчком правой кнопки мыши на папке «Пакеты исходных файлов» и выбираем пункт Сервлет (рисунок 17).

Библиотека БГУИР



Библиотека БГУИР

Рисунок 17 – Создание класса сервлета

Задаем имя сервлета `HeadersServlet`. Текст сервлета такой:

```
package com;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.*;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


@WebServlet(name    =    "HeadersServlet",    urlPatterns    =
{"/HeadersServlet"})
public class HeadersServlet extends HttpServlet {

    protected  void  processRequest(HttpServletRequest  request,
HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter ot = response.getWriter();
        try {

        String s="<html><body bgcolor=#aabbee><br>";
               s=s+"<h2> Hello from servlet</h2>";
               ot.println(s);
               ot.println("</body></html>");
            }
        finally {
            ot.close();
        }
    }

     @Override
    protected     void     doGet(HttpServletRequest     request,
HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    Protected void doPost(HttpServletRequest request,
                          HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    public String getServletInfo() {
        return "Short description";
    }// </editor-fold>
}
```


Библиотека БГУИР

Остается сделать последний шаг – внести дополнение в конфигурационный файл. Откроем вкладку `Файлы` основного окна `IDE NetBeans` (рисунок 18).

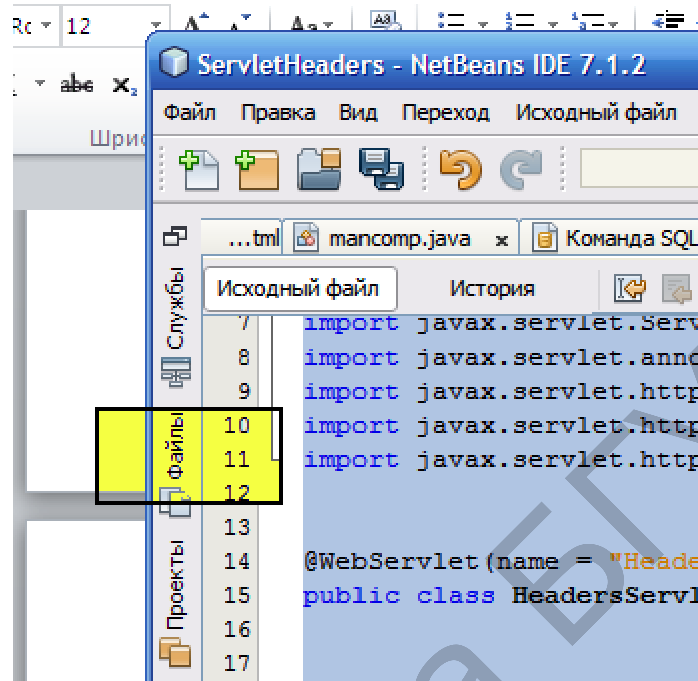


Рисунок 18 – Выбор закладки `Файлы`

Откроем конфигурационный файл `glassfish-web.xml` (рисунок 19).

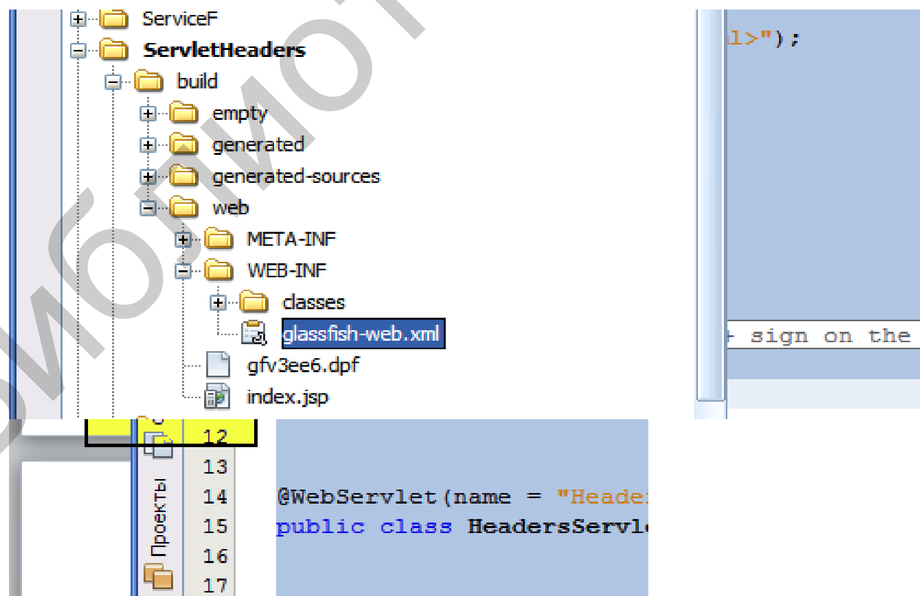


Рисунок 19 – Выбор конфигурационного файла

Для открытия используем контекстное меню с помощью правой кнопки мыши, выбирая пункт `Правка`.

Библиотека БГУИР

Наш файл первоначально имеет такой вид:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE    glassfish-web-app    PUBLIC    "-//GlassFish.org//DTD
GlassFish    Application    Server    3.1    Servlet    3.0//EN"
"http://glassfish.org/dtds/glassfish-web-app_3_0-1.dtd">
<glassfish-web-app error-url="">
  <class-loader delegate="true"/>
  <jsp-config>
    <property name="keepgenerated" value="true">
      <description>Keep a copy of the generated servlet class'
java code.</description>
    </property>
  </jsp-config>
</glassfish-web-app>
```

В него нужно добавить сведения о сервлете. В результате запишем вручную новое содержимое конфигурационного файла:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE    glassfish-web-app    PUBLIC    "-//GlassFish.org//DTD
GlassFish    Application    Server    3.1    Servlet    3.0//EN"
"http://glassfish.org/dtds/glassfish-web-app_3_0-1.dtd">
<glassfish-web-app error-url="">
  <class-loader delegate="true"/>
  <servlet>
<description>call servlet</description>
<display-name>call servlet</display-name>
<servlet-name>HeadersServlet</servlet-name>             <servlet-
class>com.HeadersServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>HeadersServlet</servlet-name>
<url-pattern>/HeadersServlet</url-pattern>
</servlet-mapping>
  <jsp-config>
    <property name="keepgenerated" value="true">
      <description>Keep a copy of the generated servlet class'
java code.</description>
    </property>
  </jsp-config>
</glassfish-web-app>
```

Теперь можно посмотреть все в действии. Щелкаем правой кнопкой мыши на имени проекта и в контекстном меню выбираем пункт `Очистить и построить`. После этого щелкаем правой кнопкой мыши на имени `index.jsp`

Библиотека БГУИР

и выбираем пункт `Выполнить файл`. Открывается следующее окно (рисунок 20).

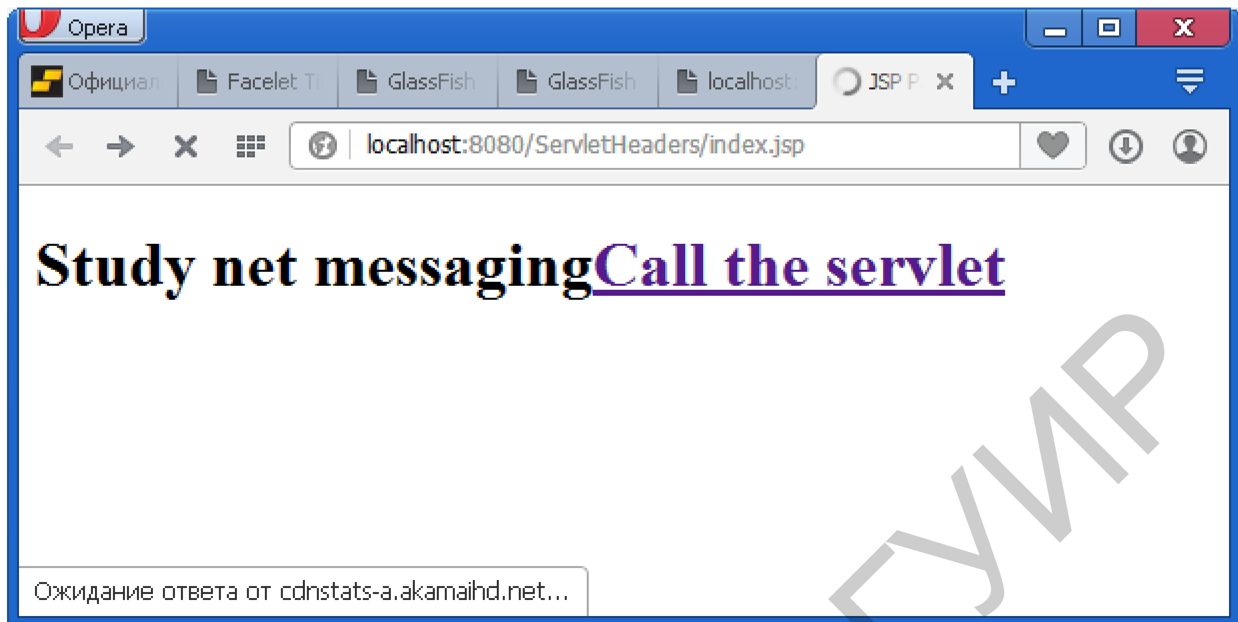


Рисунок 20 – Окно документа `jsp`

Переходим по гиперссылке, вызывая сервлет (рисунок 21).

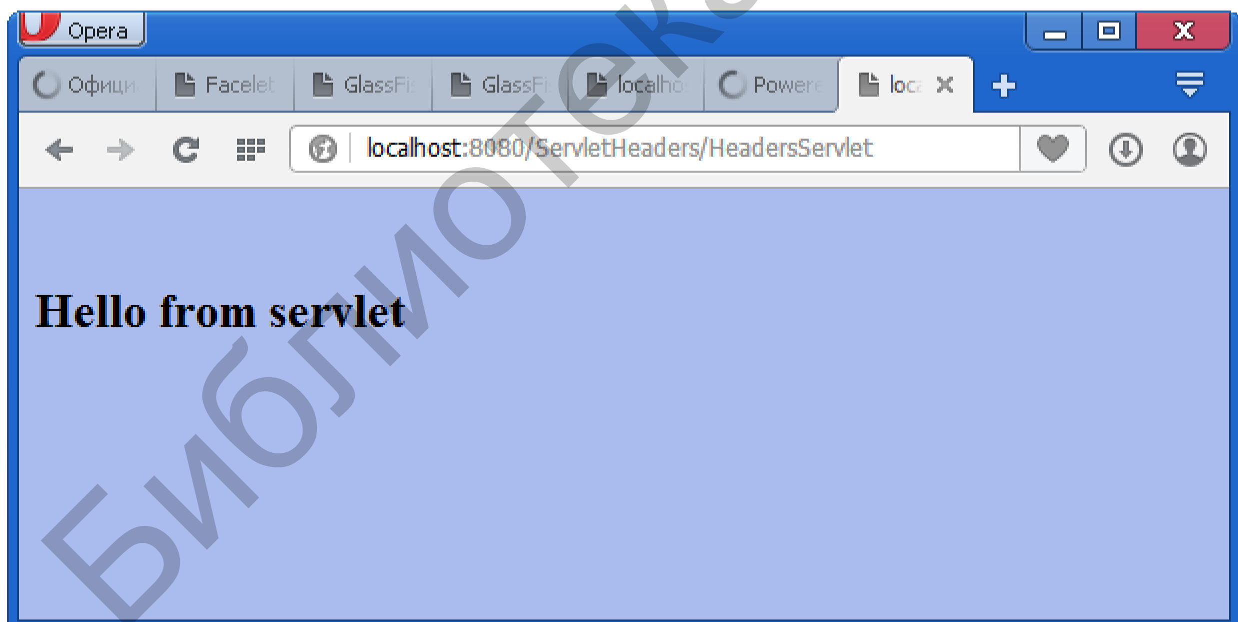


Рисунок 21 – Окно работающего сервлета

В заключение этого раздела заметим, что в распределенном приложении, создаваемом в среде `NetBeans`, нужно корректно прописывать конфигурационный файл.

Библиотека БГУИР

2 КЛИЕНТ-СЕРВЕРНЫЕ ТЕХНОЛОГИИ

2.1 Простое клиент-серверное приложение на основе сокетов

Простое клиент-серверное приложение на основе сокетов состоит из серверного приложения и одного или нескольких клиентских приложений [6, 9, 11]. Клиентское приложение связывается с серверным приложением через сокет (порт). Сокеты имеют номера от 1 до 2^{17}. Серверное приложение всегда активно. Оно прослушивает порт и при наличии запроса от клиента обрабатывает его. Передача данных может идти в обоих направлениях – как от клиента к серверу, так и от сервера к клиенту. Для передачи сообщений, как правило, используют протокол TCP. Этот протокол использует синхронное взаимодействие с подтверждением приема сообщений. Создадим в NetBeans Java клиент-серверное приложение. Ниже приведен текст программы сервера.

```
package javaserversimple;
import java.io.*;
import java.net.*;

public class JavaServerSimple {
    public static int PORT=8013;
    public static void main(String[] args) {

        ServerSocket s=null;
        try
        {
          s= new ServerSocket(PORT);
        }
        catch (Exception e1)
        {
    System.out.println("Endedwith error:"+e1.getMessage());
           System.exit(-1);
        }

          System.out.println("Server started");
          try
          {
            Socket sock =s.accept();
            BufferedReader   in=   new   BufferedReader   (new
InputStreamReader(sock.getInputStream()));
PrintWriter  out  =  new  PrintWriter(new  BufferedWriter(new
OutputStreamWriter(sock.getOutputStream())),true);
            while(true)
            {
              String str=in.readLine();
```


Библиотека БГУИР

```
              if(str.equals("END"))
              break;
              System.out.println("The server answer is:"+str);
              out.println(str);
            }
          }
          catch(Exception e2)
          {
          }

         try
          {
           s.close();
          }

         catch(Exception e3)
         {
System.out.println("Cannot normally close:"+e3.getMessage());
         }
          }

          }
```

Для взаимодействия используется порт PORT=8013. На его основе создаем серверный сокет, который используется далее для получения рабочего сокета и чтения (передачи) данных от клиента (клиенту). Серверный сокет s создаем таким образом:

```
ServerSocket s=null;
        try
        {
          s= new ServerSocket(PORT);
        }
        catch (Exception e1)
        {
    System.out.println("Endedwith error:"+e1.getMessage());
           System.exit(-1);
        }
```

Команда

```
Socket sock =s.accept();
```

является ключевой: именно она обеспечивает ожидание связи с клиентом. Пока клиент не пришлет запрос, дальнейшее выполнение серверной части приостанавливается. Данные считываются в буферизованном потоковом объекте. Буферизованный поточный объект объявляется таким образом:

```
BufferedReader in= new BufferedReader (new
InputStreamReader(sock.getInputStream()));
```


Библиотека БГУИР

```
PrintWriter out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(sock.getOutputStream())),true);
```

Основной цикл серверного приложения такой:

```
while(true)
        {
          String str=in.readLine();
          if(str.equals("END"))
              break;
          System.out.println("The server answer is:"+str);
          out.println(str);
        }
```

Строка от клиента читается в команде `String str=in.readLine()`. Она также выводится на консоль. Если приходит строка «END», то связь разрывается.

Обратимся теперь к клиентской программе. Вот ее текст (создаем как обычное приложение `Java Application`):

```
package javaclientsimple;
import java.net.*;
import java.io.*;

public class JavaClientSimple {

    public static void main(String[] args) {
           Socket sock=null;
        try
        {
        InetAddress addr = InetAddress.getByName(null);
        System.out.println("address="+addr);
        sock= new Socket(addr,8013);
        BufferedReader in = new BufferedReader( new
InputStreamReader(sock.getInputStream()));
        PrintWriter out =
new PrintWriter(
new  BufferedWriter(new
OutputStreamWriter(sock.getOutputStream())),true);
        for(int i=0;i<10;i++)
        {
            out.println("howdy? -"+i );
            String str=in.readLine();
            System.out.println(str);

        }
          System.out.println("The END");
          throw new Exception();

        }
        catch(Exception e)
```


Библиотека БГУИР

```
        {
            try
            {sock.close();}
            catch(Exception e4)
            { }}}}
```

Также создаем сокет с указанием порта:

```
sock= new Socket(addr,8013);
```

Создаем объектные потоковые переменные для сокетного ввода-вывода:

```
BufferedReader in = new BufferedReader( new
InputStreamReader(sock.getInputStream()));
PrintWriter   out   =   new   PrintWriter(new   BufferedWriter(new
OutputStreamWriter(sock.getOutputStream())),true);
```

Окно с выводом информации от сервера представлено на следующем скриншоте (номера выводимых сообщений пронумерованы от 1 до 9) (рисунок 22).

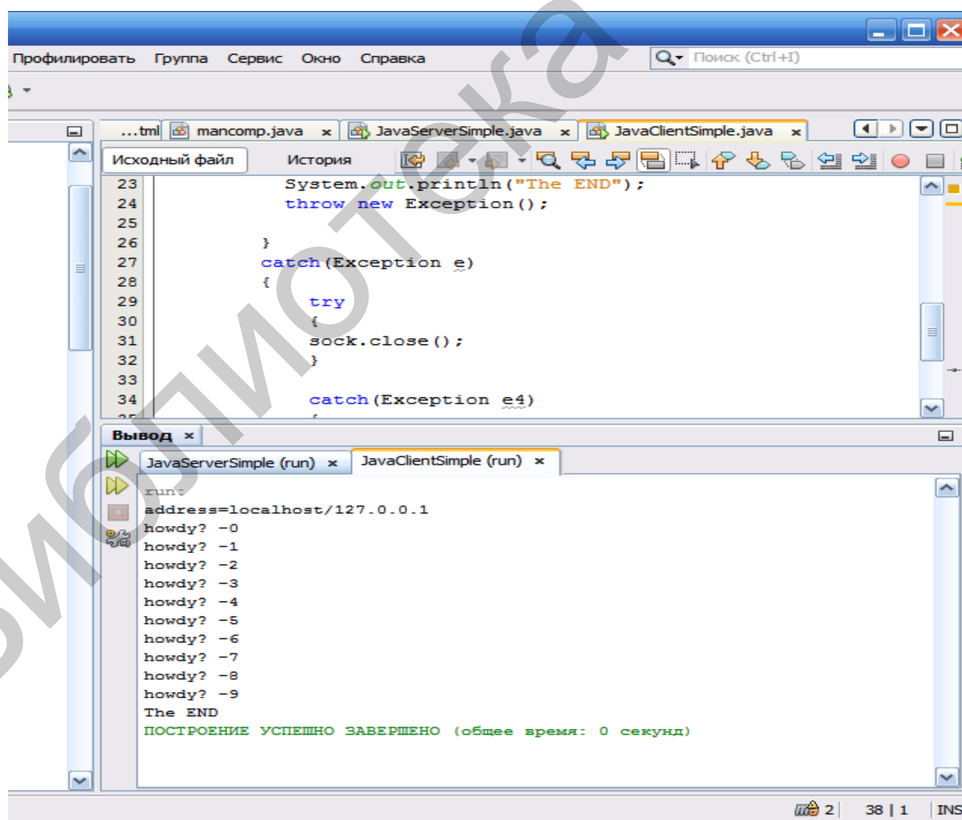


Рисунок 22 – Отображение данных, полученных от серверного приложения

Библиотека БГУИР

Несколько изменим приложение. Пусть клиентская часть обращается к серверному приложению, чтобы получить курс валют (передается строка `dollar` или `euro`; сервер возвращает курс клиенту).

Серверная часть теперь будет выглядеть таким образом:

```
package javaserversimple;
import java.io.*;
import java.net.*;

public class JavaServerSimple {
    public static int PORT=8013;

    public static void main(String[] args) {

        ServerSocket s=null;
        try
        {
          s= new ServerSocket(PORT);
        }
        catch (Exception e1)
        {
 System.out.println("Ended with error:"+e1.getMessage());
           System.exit(-1);
        }
          System.out.println("Server started");
          try
          {
            Socket sock =s.accept();
            BufferedReader   in=   new   BufferedReader   (new
InputStreamReader(sock.getInputStream()));
            PrintWriter    out     =     new     PrintWriter(new
BufferedWriter(new
OutputStreamWriter(sock.getOutputStream())),true);
            while(true)
            {
              String str=in.readLine();
              if(str.equals("END"))
                  break;
              if(str.trim().toLowerCase().equals("dollar"))
              {
                  out.println("server: --  15000 Blr");

              }
              else
               if(str.trim().toLowerCase().equals("euro"))
              {

                  out.println("server: ---  17000 Blr");
              }
              else
```


Библиотека БГУИР

```
out.println("server: not found correct solution to request!");
            }
          }
        catch(Exception e2)
          {

          }

         try
         {
           s.close();

          }

         catch(Exception e3)
         {
          System.out.println("Cannot normally
close:"+e3.getMessage());
         }
          }

          }
```

Приведенный код серверного приложения по сути не изменился. Нетрудно понять его «бизнес-логику». Обратимся к клиентской части. Она имеет следующий вид

```
package javaclientsimple;
import java.net.*;
import java.io.*;
import java.util.Scanner;

public class JavaClientSimple {

    public static void main(String[] args) {
          Socket sock=null;
        try
        {
        InetAddress addr = InetAddress.getByName(null);
        System.out.println("address="+addr);
        sock= new Socket(addr,8013);
        BufferedReader in = new BufferedReader( new
InputStreamReader(sock.getInputStream()));
        PrintWriter out = new PrintWriter(new
BufferedWriter(new
OutputStreamWriter(sock.getOutputStream())),true);

        Scanner scanner = new Scanner(System.in);

                String ename;
    System.out.println("Enter currency: dollar  or  euro ");
```


Библиотека БГУИР

```
ename= scanner.nextLine();
//This is needed to pick up the new line

        out.println(ename);
        String answer=in.readLine();
        System.out.println(answer);
        out.println("END");
        System.out.println("The END");
        throw new Exception();

    }

    catch(Exception e)
    {
        try

        {
        sock.close();
        }
        catch(Exception e4)
        {

        }
        }
    }
}
```

В клиентской части для ввода наименования валюты используем объект `Scanner`. Чтение строки с клавиатуры реализуется этим объектом в строке

```
ename= scanner.nextLine();
```

Дальнейшее взаимодействие клиента с сервером реализуется таким образом:

```
out.println(ename);
String answer=in.readLine();
```

Сначала название валюты отсылается на сервер, а затем считывается ответ.

Следующий скриншот (рисунок 23) показывает обмен информацией между клиентом и сервером.

Библиотека БГУИР

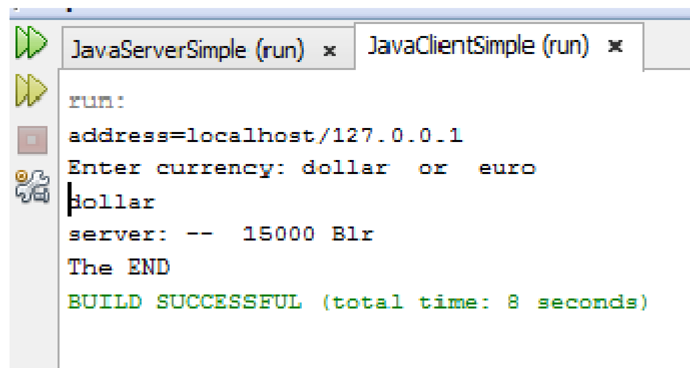


Рисунок 23 – Обмен информацией между клиентом и сервером

2.2 Реализация соединения на основе протокола UDP

В отличие от протокола TCP протокол UDP является более скоростным, поскольку не требует подтверждения приема посылок (сообщений) от ЭВМ-получателя. Таким образом, более высокое быстродействие достигается за счет уменьшения надежности связи. Рассмотрим реализацию клиент-серверного приложения на основе UDP, аналогичного тому, которое приведено в предыдущем подразделе. Начнем со стороны сервера:

```
package serverudp;

import java.io.*;
import java.net.*;

public class ServerUDP {

      public static int PORT=8014;
      private static  DatagramSocket sock=null;
      public static void main(String[] args) {
      byte[] buffer= new byte[1024];

     try
          {
              sock= new DatagramSocket(PORT);
              sock.getLocalPort();
          }

          catch(Exception ex)
          {
              System.out.println("unable to set port");
              System.exit(-1);
          }

            while(true)
            {
                 try
```

Библиотека БГУИР

```
        {
     DatagramPacket pc= new DatagramPacket(buffer,1024);
     System.out.println("Server waits for a new message");
            sock.receive(pc);
            String gotstr= new String(pc.getData());
            String answer="";
     System.out.println("Server got message->"+gotstr);

             if(gotstr.trim().indexOf("end")>=0)
             {
                 sock.close();
                 System.exit(0);
             }

            if(gotstr.trim().equals("dollar"))
            {
                answer="15000";
            }
            else

               if(gotstr.trim().equals("euro"))

            {
                answer="17500";
            }
 System.out.println("server prepared the answer->"+answer);
                 byte[] answerbf=answer.getBytes();
                 pc.setData(answerbf) ;
                 sock.send(pc);
      System.out.println("the answer is sent to client");

           }
catch(Exception e2)
         {
System.out.println("The error on server ->"+e2.getMessage());
         }
           }

         }
     }
```

Взаимодействие сервера и клиента (и наоборот) выполняется через объект

`DatagramSocket sock.`

Этот объект имеет метод `send` для отправки сообщения клиенту и метод `receive` для чтения сообщения от клиента. Аргументом обоих методов является датаграмма (пакет данных) `DatagramPacket pc`. Данные в объект датаграмма заносятся с помощью метода `setData`. В качестве аргумента

Библиотека БГУИР

методу передается массив байтов. Заметим, что получить массив байтов из текстовой строки можно с помощью команды наподобие следующей

```
byte[] answerbf=answer.getBytes();
```

Здесь формируется массив байтов `answerbf` из строки `answer` с помощью метода `getBytes()`. Данные, переданные в самой датаграмме, можно получить на основе команды, показанной в качестве примера ниже:

```
String gotstr= new String(pc.getData());
```

Обмен данными между процессором и клиентом выполняется в цикле, пока не придет строка, содержащая слово «`end`»:

```
if(gotstr.trim().indexOf("end")>=0)
{
    sock.close();
    System.exit(0);
}
```

С учетом сказанного логика работы серверного приложения очевидна.

Программ `UDP`-клиента имеет следующий вид:

```
package clientudp;
import java.net.*;
import java.io.*;

public class ClientUDP
{
    public static int PORT=8014;
    public static int bufsize=1024;
    public static void main(String[] args) {

        InetAddress adr=null;
        try
        {
        adr=InetAddress.getByName(null);

        DatagramSocket sc1=new DatagramSocket();
        String message="euro";
        byte [] sendbf = message.getBytes() ;
        DatagramPacket p1=
        new DatagramPacket(sendbf,sendbf.length,adr,PORT);
        System.out.println("Prepare to send -> "+message);
        sc1.send(p1);
        System.out.println("Send to Server:"+message);

         p1.setData( new byte[1024] ) ;
         sc1.setSoTimeout(3000) ;
         sc1.receive(p1);
```


Библиотека БГУИР

```
            String ans_currency=new String(p1.getData());
            int k=ans_currency.indexOf('\0');
            if (k>=0)
            {
                ans_currency=ans_currency.substring(0,k);

            }

           System.out.println(ans_currency) ;
           String message2="end";
           byte[] answerbf=message2.getBytes();
                     p1.setData(answerbf) ;
                     sc1.send(p1);

       }
                 catch(Exception e2)
           {
       System.out.println("Client has error:"+e2.getMessage());
               System.exit(-1);
           }}
   }
```

Клиент также использует объект `DatagramSocket sc1` для отсылки или приема датаграмм (`DatagramPacket`). Его логика зеркальна по отношению к логике серверного приложения: то, что клиент отсылает (`send`) на сервер, последний должен принять (`receive`), и наоборот. В приведенном приложении введено время тайм-аута для ожидания ответа от сервера:

```
sc1.setSoTimeout(3000) ;
```

Ожидание не превосходит 3000 мс = 3 с. Если время тайм-аута истекло без ответа, то программа завершается.

При обмене окно сервера содержит следующие выходные данные (рисунок 24).

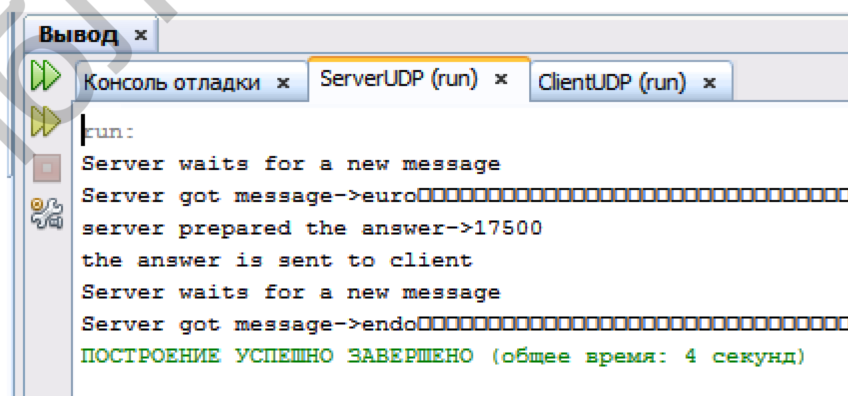


Рисунок 24 − Отображение данных в окне сервера

Выходное окно клиента имеет вид, показанный на рисунке 25.

Библиотека БГУИР

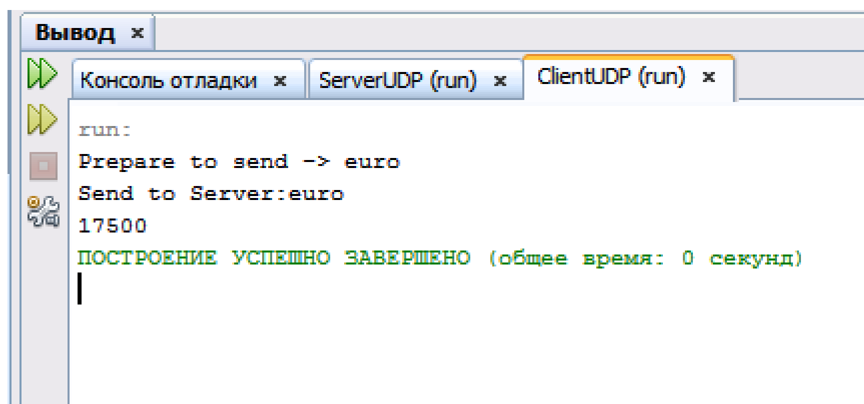


Рисунок 25 – Отображение данных в окне клиента

2.3 Высокоуровневые серверные приложения, обрабатывающие запросы к базе данных

2.3.1 Работа со встроенной базой данных Derby с примером на JSF

Работа с базой данных (БД) в среде NetBeans реализуется с помощью встроенной оснастки (мастера). Выберем закладку Services (в скриншоте ниже – самая левая панель). Откроем щелчком на плюсике вершину Databases. Выберем Java DB в качестве СУБД (встроена в NetBeans – рисунок 26). Затем запустим сервер баз данных и создадим пустую базу (через контекстное меню на имени сервера Java DB – рисунок 27).

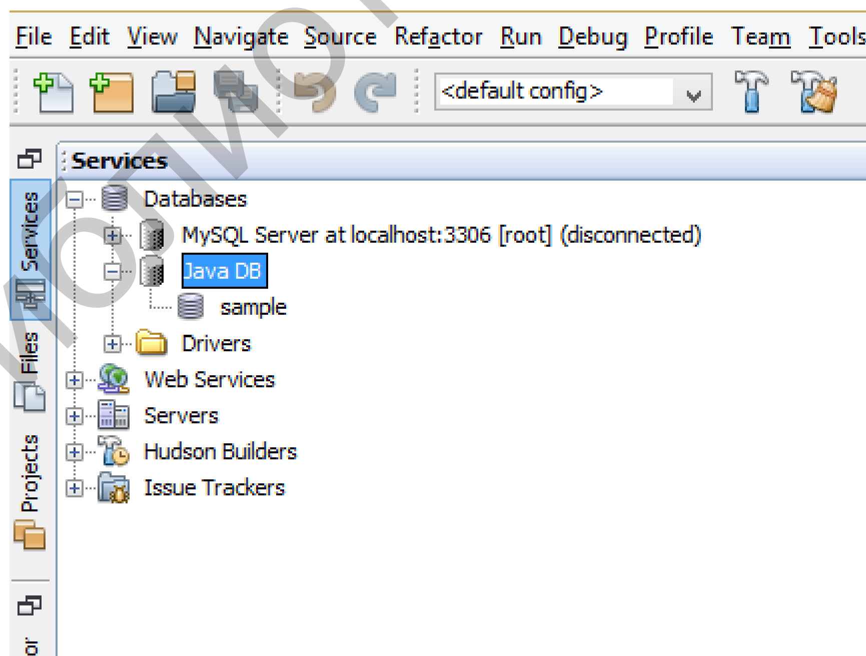


Рисунок 26 – Выбор СУБД

Библиотека БГУИР

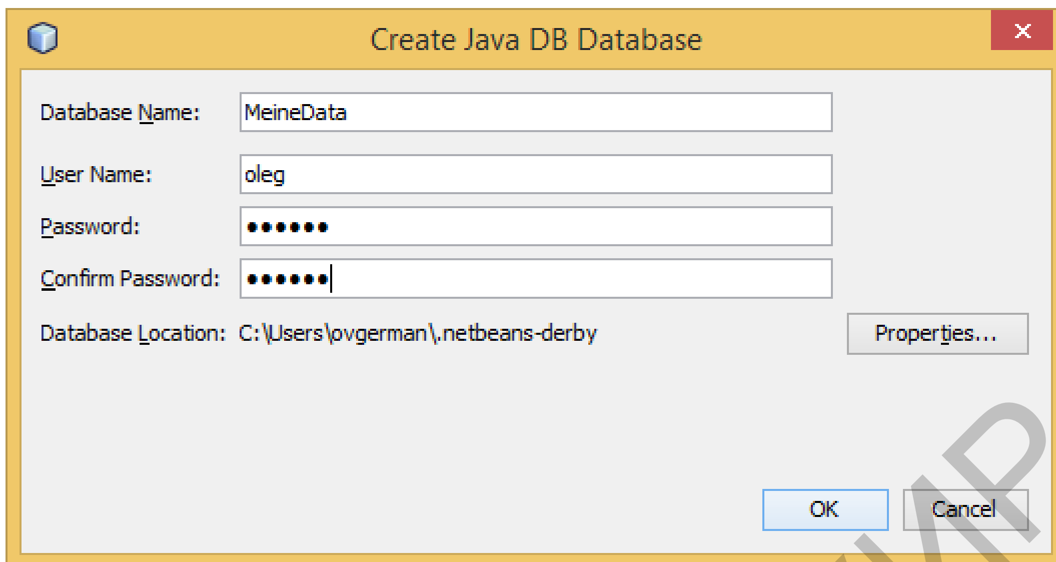


Рисунок 27 – Создание пустой базы данных

Наша база называется `MeineData`. Мы задали для нее имя пользователя и пароль. Теперь надо подключиться к этой базе (из контекстного меню активированного правой кнопкой мыши на имени базы выбрать пункт `Connect`). Наличие подключения определяется по появлению строки, выделенной полосой. Щелкнем мышью на этой полосе и выберем опцию `execute command` (рисунок 28). Теперь мы можем вводить `SQL`-команды.

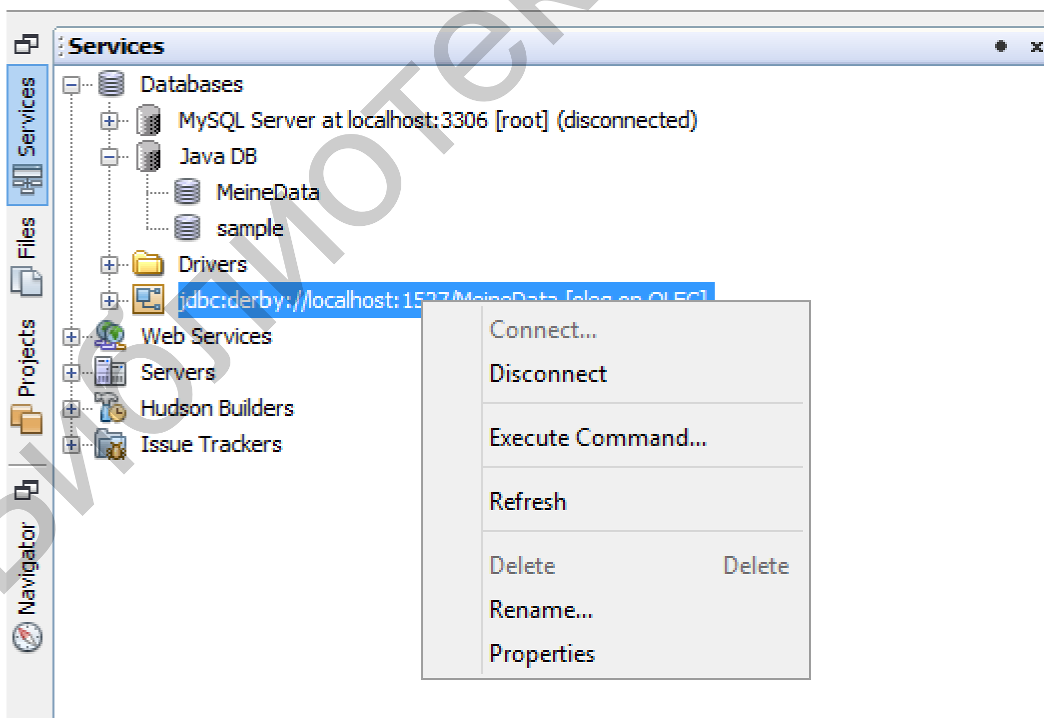
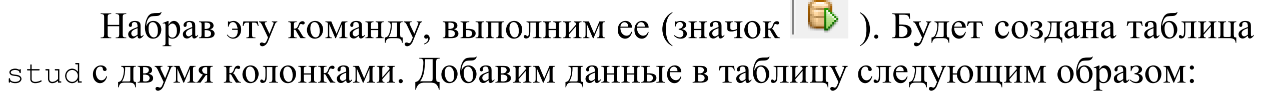


Рисунок 28 – Подключение к редактору `SQL`-команд

Библиотека БГУИР

```
create table stud( fio varchar(25) PRIMARY KEY NOT NULL,
age int);
```

Набрав эту команду, выполним ее (значок ). Будет создана таблица `stud` с двумя колонками. Добавим данные в таблицу следующим образом:

```
insert into stud values('petrov',17);
insert into stud values('sidorov',18);
insert into stud values('abasov',19);
```

Имея эту маленькую базу, подсоединимся к ней и произведем выборку данных. Запомните, нужно обязательно *подключить* к проекту библиотечный архивный `jar`-файл с драйвером базы данных `ClientDerby`. Это делаем так: щелкаем правой кнопкой мыши на вершине `Libraries` в дереве проекта и выбираем подключить `jar folder`. Заходим в дистрибутив `java`. В данном случае это производится по адресу: `c:\Program Files (x86)\Java\jdk1.7.0_45\db\lib\`. В этой библиотеке находится целый ряд модулей и среди них `derbyClient` (см. скриншот на рисунке 29). Выбираем его. Нажимаем кнопку `Open`.

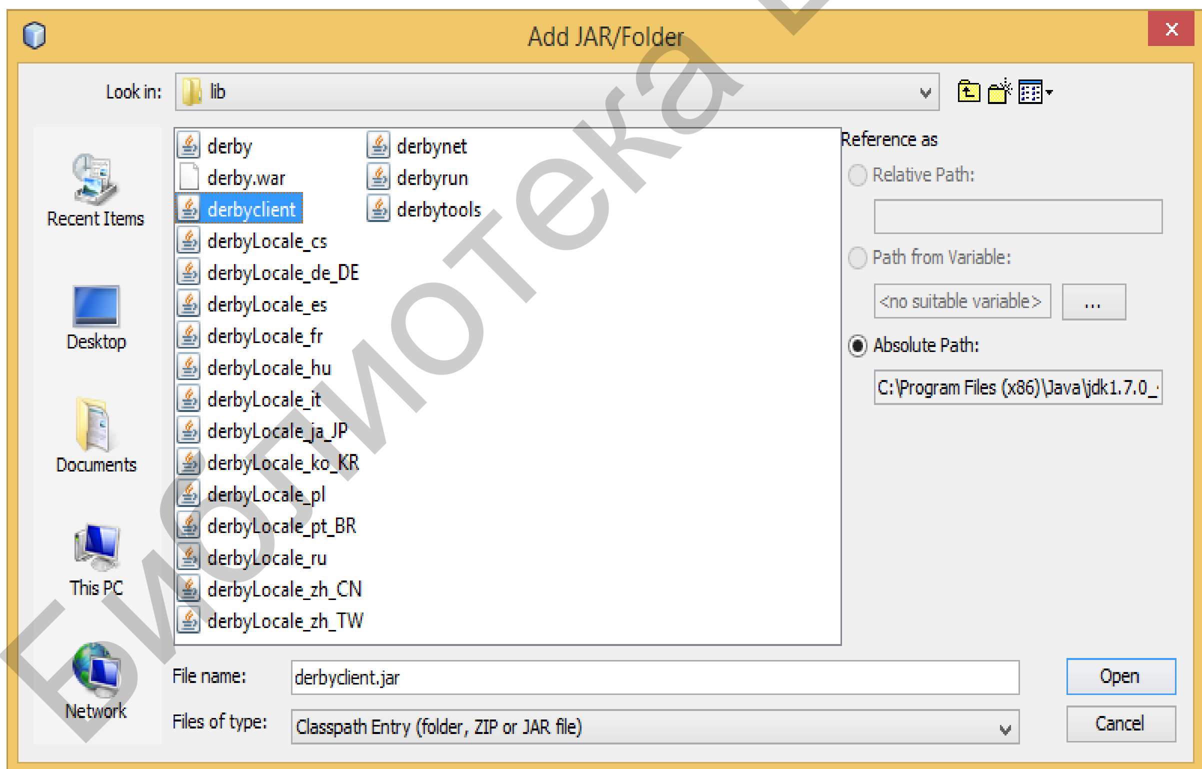


Рисунок 29 – Подключение драйвера `derbyclient.jar`

Теперь приступаем к написанию кода:

```
package javadatabase;
```

Библиотека БГУИР

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSetMetaData;

public class JavaDataBase {

// connection string with data base

    private         static         String         dbURL        =
"jdbc:derby://localhost:1527/MeineData;user=oleg;password=germ
an";
    private static String tableName = "stud";
    // jdbc Connection
    private static Connection conn = null;
    private static Statement stmt = null;

    public static void main(String[] args)
    {
        createConnection();
        insertStud("Julia", 28);
        selectStud();
        shutdown();
    }

    private static void createConnection()
    {   try
        {

//Driver class
Class.forName("org.apache.derby.jdbc.ClientDriver").
newInstance();
            //Get a connection
            conn = DriverManager.getConnection(dbURL);
        }
        catch (Exception except)
        {
            except.printStackTrace();
        }}

    private static void insertStud(String name, int ag)
    {  try
        {
            stmt = conn.createStatement();
     stmt.execute("insert into " + tableName + " values ("+
                   "'" + name + "'," + ag +")");
            stmt.close();
        }

        catch (SQLException sqlExcept)
```

```
        {
            sqlExcept.printStackTrace();
        }
    }

    private static void selectStud()
    {
        try
        {
            stmt = conn.createStatement();
            ResultSet  results  =  stmt.executeQuery("select  *
from " + tableName);
            ResultSetMetaData rsmd = results.getMetaData();
            int numberCols = rsmd.getColumnCount();
            for (int i=1; i<=numberCols; i++)
            {
                //print Column Names

System.out.print(rsmd.getColumnLabel(i)+"\t\t");
            }

            System.out.println("\n----------------------------
---------------------");

            while(results.next())
            {

                String Name = results.getString(1);
                int age = results.getInt(2);
        System.out.println("\t\t" + Name + "\t\t" + age);
            }
            results.close();
            stmt.close();
        }
        catch (SQLException sqlExcept)
        {
            sqlExcept.printStackTrace();
        }
    }

    private static void shutdown()
    {
        try
        {
            if (stmt != null)
            {
                stmt.close();
            }
            if (conn != null)
            {
                DriverManager.getConnection(dbURL             +
";shutdown=true");
```

```
                    conn.close();
                }
            }
            catch (SQLException sqlExcept)
            {

            }
        }
    }
```

Результат работы этого приложения показан на рисунке 30.

```
Output
JavaDataBase (run) ×   Java DB Database Process ×
run:
FIO                 AGE
------------------------------------------------
                    petrov          17
                    sidorov         18
                    abasov          19
                    Julia           28
BUILD SUCCESSFUL (total time: 3 seconds)
```

Рисунок 30 – Вывод информации по SQL-запросу

Работа с базой данных реализуется через соединение (Connection conn). При открытии соединения ему нужно передать адрес (URL) базы данных:

```
conn = DriverManager.getConnection(dbURL);
```

В нашем примере адрес устанавливается в команде

```
Private static String dbURL =
"jdbc:derby://localhost:1527/MeineData;user=oleg;password=germ
an";
```

Строка адреса, вообще говоря, для разных СУБД различная. В адресе содержится сетевое имя ЭВМ (localhost), номер порта (1527), имя базы (MeineData), имя пользователя (oleg) и пароль (german). Для работы с базой нужно подключить класс драйвера:

```
Class.forName("org.apache.derby.jdbc.ClientDriver").
newInstance();
```

Затем следует выполнить запрос к базе данных. Запросы бывают разных типов: на выборку, изменение, удаление и вставку записей. Запрос на выборку представлен ниже:

Библиотека БГУИР

```
stmt = conn.createStatement();
ResultSet   results   =   stmt.executeQuery("select   *   from   "   +
tableName);
```

В результате выполнения этого запроса формируется курсор (набор записей). По курсору можно выполнять перемещение и получать значения конкретных записей. В нашем примере это делается следующим образом (метод `next`):

```
while(results.next())
  {
   String Name = results.getString(1);
   int age = results.getInt(2);
   System.out.println("\t\t" + Name + "\t\t" + age);
  }
```

Таблица состоит из двух колонок: `fio`, `age`. Первая имеет тип строки, вторая – целого числа. Соответственно выборка значений из этих колонок реализуется методом `getString` или `getInt`. В скобках записывается номер столбца.

При выборке данных формируется результирующий набор записей, предназначенный только для просмотра вперед (`FORWARD_ONLY`). Изменим несколько текст этой программы, чтобы показать, как перемещаться по набору записей.

```
package javadatabase;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSetMetaData;

public class JavaDataBase {

    private         static         String         dbURL         =
"jdbc:derby://localhost:1527/MeineBase;user=oleg;password=german
";
    private static String tableName = "stud";
    // jdbc Connection
    private static Connection conn = null;
    private static Statement stmt = null;
    private static ResultSet rs=null;
    public static void main(String[] args)
    {
        try
        {
       createConnection();
```

```
            conn = DriverManager.getConnection(dbURL);
            stmt=
conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet
.CONCUR_UPDATABLE);

       String  sq_str="SELECT * FROM stud";
//Строка запроса на выборку

                rs = stmt.executeQuery(sq_str);
                rs.absolute(2);
          int rowN = rs.getRow(); // rowNum should be 2
          System.out.println("Текущая строка есть " + rowN);
                rs.relative(-1);
                String name1 = rs.getString("fio");
                    int gr1 = rs.getInt("age");
                    System.out.println(name1 + "      " +gr1);
       conn.close();
     }

     catch(Exception ex)
     {
          System.out.println("Navigation error :"+ex);
     }

        shutdown();
    }

    private static void createConnection()
    {
        try
        {
Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance(
);
            //Get a connection
            conn = DriverManager.getConnection(dbURL);
        }
        catch (Exception except)
        {
            except.printStackTrace();
        }
    }

    private static void insertStud(String name, int ag)
    {
        try
        {
            stmt = conn.createStatement();
     stmt.execute("insert into " + tableName + " values ("+
                    "'" + name + "'," + ag +")");
            stmt.close();
        }
```


Библиотека БГУИР

```
        catch (SQLException sqlExcept)
        {
            sqlExcept.printStackTrace();
        }
    }

   private static void shutdown()
    {
        try
        {
            if (stmt != null)
            {
                stmt.close();
            }
            if (conn != null)
            {
      DriverManager.getConnection(dbURL+ ";shutdown=true");
                conn.close();
            }
        }
        catch (SQLException sqlExcept)
        {
        }
    }
}
```

Обратим внимание на то, как мы создаем команду, обеспечивающую просмотр результирующего набора записей в любом направлении:

```
       stmt=
conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet
.CONCUR_UPDATABLE);
```

Указывается, что результирующий набор можно просматривать вперед/назад (`ResultSet.TYPE_SCROLL_INSENSITIVE`), а также, что его можно изменять (`ResultSet.CONCUR_UPDATABLE`).

Приведены команды, показывающие, как перемещаться по набору:

```
rs.absolute(2);    //переход по абсолютному адресу на пятую запись;
rs.relative(-1); // возврат от текущей записи на одну запись вверх.
```

Можно проверить, не вышли ли мы за последнюю запись:

```
if (!rs.isAfterLast()){…}
```

Движение по записям набора выполняется с помощью методов `next()` (переход к следующей записи), `previous()` (переход к предыдущей записи),

Библиотека БГУИР

`first()` (переход к первой записи), `last()` (переход к последней записи набора).

Рассмотрим другие стандартные действия с БД. Изменим приведенную выше программу таким образом, чтобы она вставляла новую запись в БД. Новую запись сформируем так, как показано ниже:

```
String sq_str ="Insert into stud
               values('"+sname+"',"+sg+","+sage+")";
```

Вставка текстового поля требует, чтобы оно было взято в кавычки (для этой цели мы используем одиночные кавычки в приведенной выше строке. Вставка числовых полей (и полей других типов) каких-либо дополнительных условий не требует. Строку для вставки можно было бы сформировать и таким образом:

```
String sq_str ="Insert into stud values('"+sname+"',"+
Integer.parseInt(sage)+")";
```

Здесь метод `parseInt` класса `Integer` преобразует строку `sage` в целое число.

Программа для вставки имеет следующий вид:

```
package javadatabase;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSetMetaData;

public class JavaDataBase {

    private          static          String          dbURL          =
"jdbc:derby://localhost:1527/MeineBase;user=oleg;password=german
";
    private static String tableName = "stud";
    // jdbc Connection
    private static Connection conn = null;
    private static Statement stmt = null;
    private static ResultSet rs=null;
    public static void main(String[] args)
    {
        try

        {
       createConnection();
```


Библиотека БГУИР

```
              conn = DriverManager.getConnection(dbURL);
         stmt=
conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet
.CONCUR_UPDATABLE);

                String sname;
                String sage;
        byte [] barray=new byte[20];
        System.out.println("Input Name");
        System.in.read(barray);
        sname=new String(barray);
        sname=sname.trim();
                   System.out.println("Input age");
                   System.in.read(barray);
                   sage=new String(barray);
                   sage=sage.trim();
      int ag=   Integer.parseInt(sage.replaceAll("[\\D]", ""));

      System.out.println("New Record is: "+sname+"   "+ag);
              insertStud(sname,ag);
              selectStud();
         conn.close();
       }
       catch(Exception ex)
       {
            System.out.println("Navigation error :"+ex);
       }

          shutdown();
      }

      private static void createConnection()
      {
          try
          {

Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance(
);
              //Get a connection
              conn = DriverManager.getConnection(dbURL);
          }
          catch (Exception except)
          {
              except.printStackTrace();
          }
      }

      private static void insertStud(String name, int ag)
      {
          try
          {
              stmt = conn.createStatement();
```


Библиотека БГУИР

```
     stmt.execute("insert into " + tableName + " values ("+
                  "'" + name + "'," + ag +")");
            stmt.close();
        }
        catch (SQLException sqlExcept)
        {
            sqlExcept.printStackTrace();
        }
    }

       private static void selectStud()
    {
        try
        {
            stmt = conn.createStatement();
            ResultSet results = stmt.executeQuery("select * from
" + tableName);
            ResultSetMetaData rsmd = results.getMetaData();
            int numberCols = rsmd.getColumnCount();
            for (int i=1; i<=numberCols; i++)
            {
                //print Column Names
                System.out.print(rsmd.getColumnLabel(i)+"\t\t");
            }

            System.out.println("\n---------------------------------
-------------------");

            while(results.next())
            {
                String Name = results.getString(1);
                int age = results.getInt(2);
        System.out.println("\t\t" + Name + "\t\t" + age);
            }
            results.close();
            stmt.close();
        }
        catch (SQLException sqlExcept)
        {
            sqlExcept.printStackTrace();
        }
    }

    private static void shutdown()
    {
        try
        {
            if (stmt != null)
            {
                stmt.close();
            }
            if (conn != null)
```


Библиотека БГУИР

```
            {
                DriverManager.getConnection(dbURL                +
";shutdown=true");
                conn.close();
            }
        }
        catch (SQLException sqlExcept)
        {
        }
    }
}
```

Метод вставки записи в таблицу имеет такой вид:

```
stmt.execute("insert into " + tableName + " values ("+
                    "'" + name + "'," + ag +")");
```

Здесь `stmt` – объект класса `Statement`. Метод `execute` позволяет вставлять и удалять записи. Для изменения записей или компонентов БД следует использовать метод `executeUpdate`.

Пример вывода данной программы помещен ниже:

```
run:
Input Name
german
Input age
55
New Record is: german   55
FIO       AGE
-------------------------------------------------
          petrov        16
          ivanov        17
          abasov        18
          german        55
ПОСТРОЕНИЕ УСПЕШНО ЗАВЕРШЕНО (общее время: 8 секунд)
```

Естественно, возникает желание реализовать приложение с базой данных в среде `web`, сделав его доступным для пользователей Интернета. Здесь мы остановимся на возможностях технологии `JSF` (`Java Server Faces`). Начнем с простого приложения `JSF` в `NetBeans`. Выполним следующие шаги.

1. Создаем обычное `web`-приложение – `webjsf` (рисунки 31, 32).
2. Выбираем тип проекта `web-приложение`.

Библиотека БГУИР

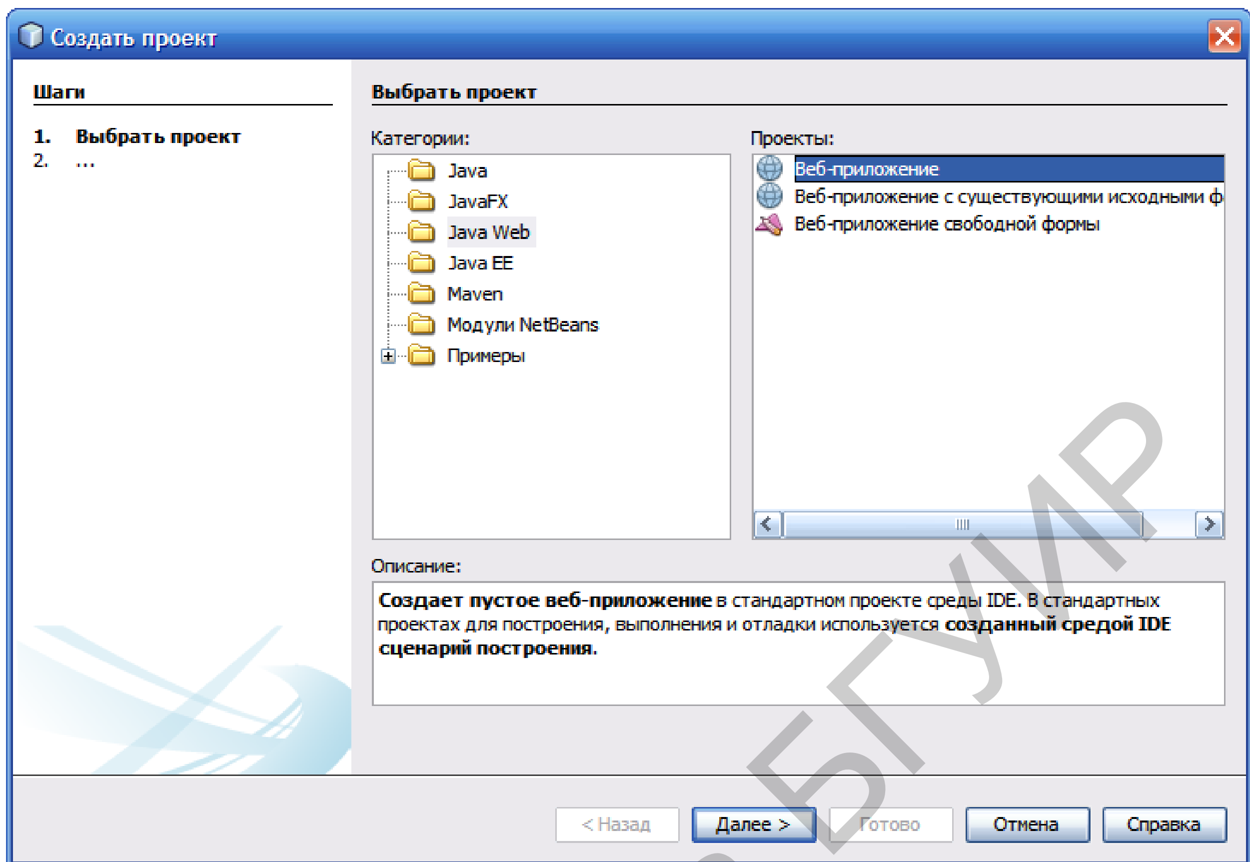


Рисунок 31 – Тип проекта – `Java Web`

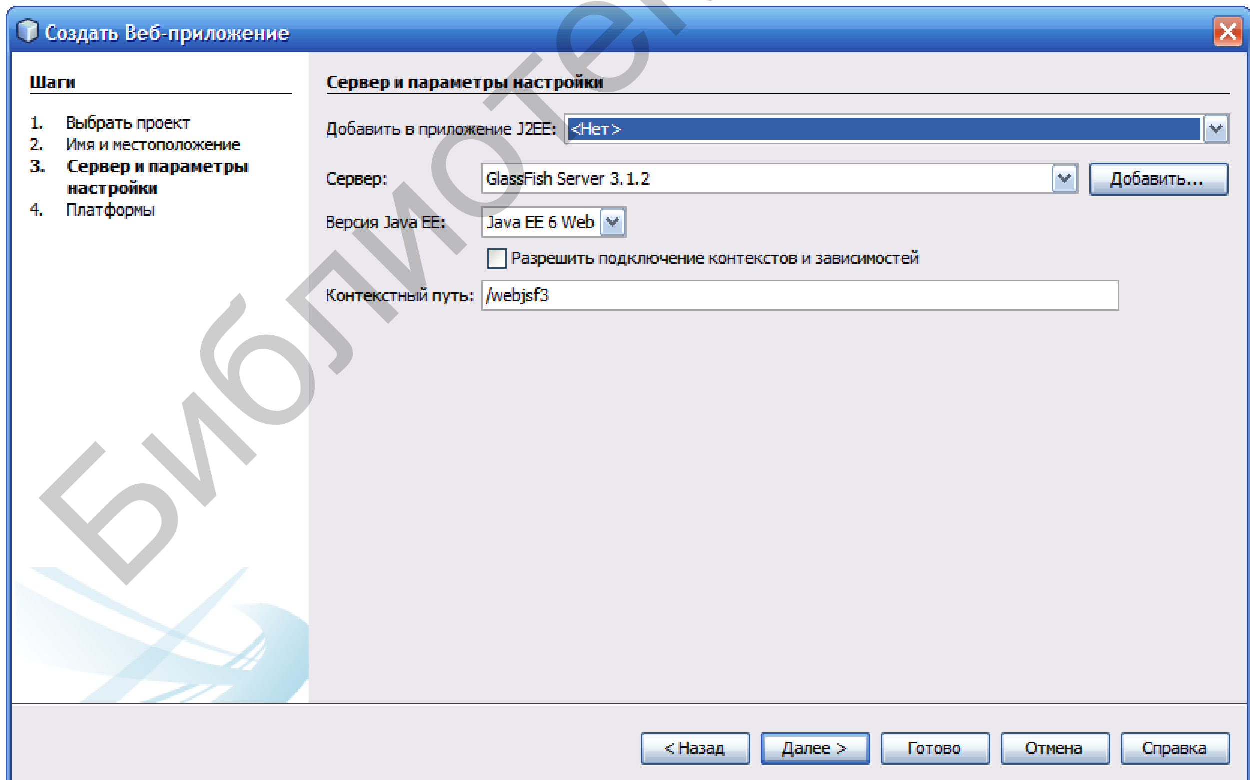


Рисунок 32 – Задаем сервер `GlassFish Server` 3.1.2

Библиотека БГУИР

3. При создании выбираем платформу JSF (рисунок 33).

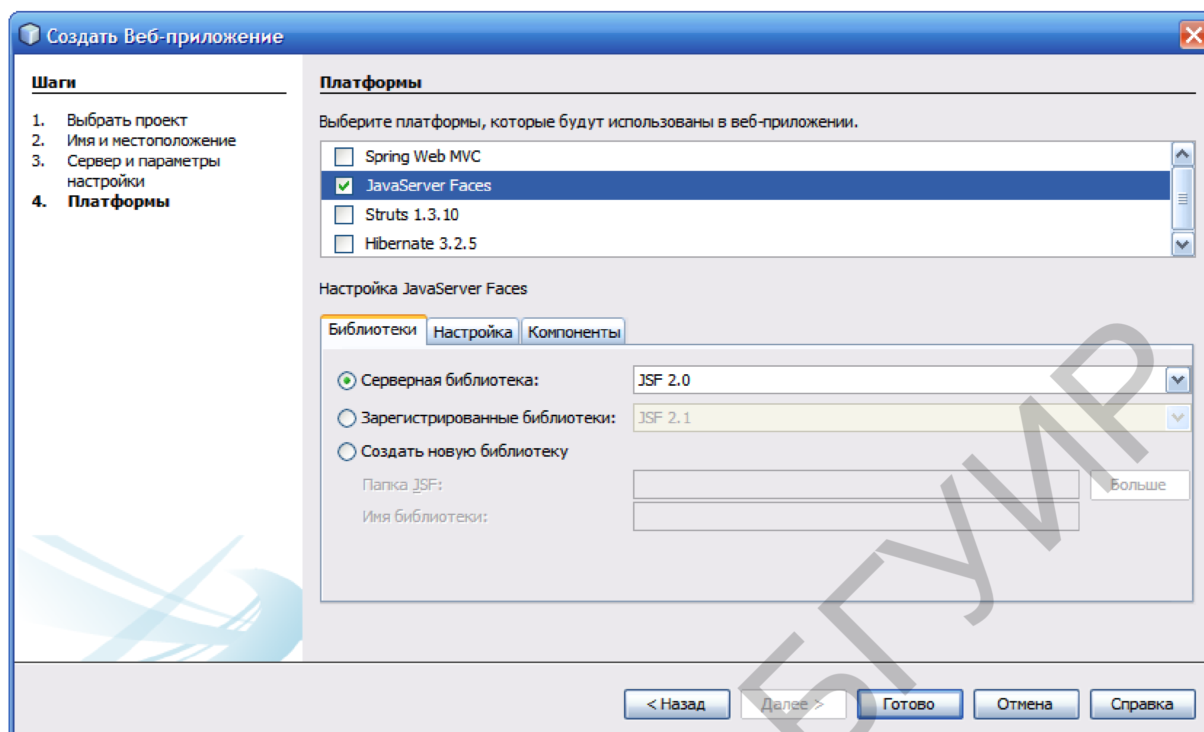


Рисунок 33 – Выбираем платформу Java Server Faces

4. Добавляем в web-приложение managed bean (управляемый компонент).

С этой целью щелкаем правой кнопкой мыши на узле Пакеты исходных файлов дерева проекта и выбираем пункты Создать и Управляемый компонент JSF (рисунок 34).

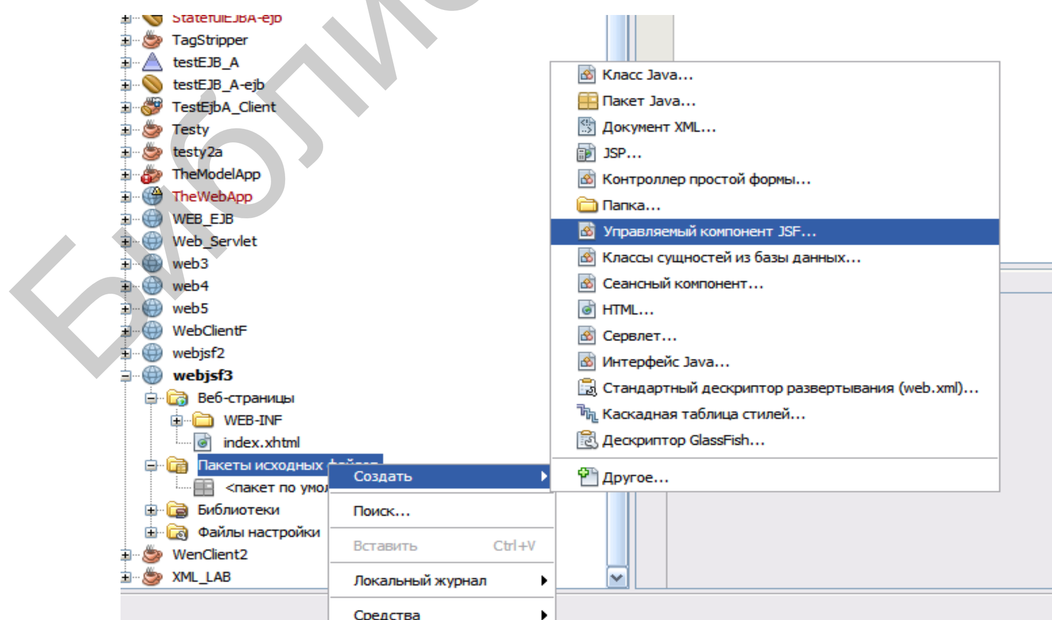


Рисунок 34 – Выбираем управляемый компонент JSF

Библиотека БГУИР

В англоязычной версии это реализуется через `JSF Managed Bean`. Задаем парметры управляемого компонента (рисунок 35).

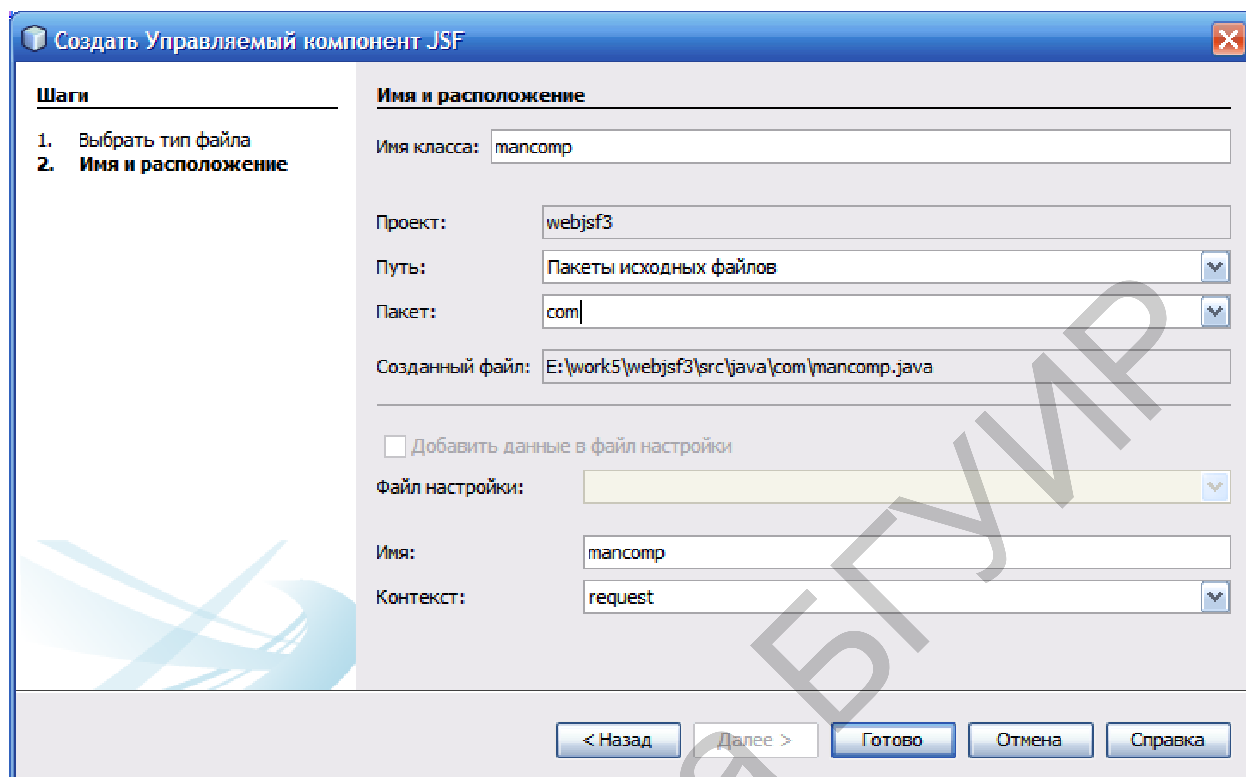


Рисунок 35 – Задаем параметры управляемого компонента `JSF`

Изменяем его текст следующим образом:

```
package com;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
@ManagedBean
@RequestScoped
public class mancomp {
    /**
     * Creates a new instance of mancomp
     */
    public mancomp() {
    };
        public String getAnswer(String x) {
        return "Hello from JSF "+x;
    }}
```

В файле `index.xhtml` пишем обращение:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
```

Библиотека БГУИР

```
        xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        <h2> Current Lesson on JFACE</h2>
        <br>
               #{mancomp.getAnswer("Oleg German")}
        </br>
<h:form>
    <h:inputText id="userNumber" size="40" maxlength="40"
value="#{mancomp.getAnswer('Oleg 2')}" />

</h:form>
    </h:body>
</html>
```

Здесь мы видим обращение к управляемому бину:

```
#{mancomp.getAnswer('Oleg 2')
```

Запускаем этот индексный файл на выполнение, получаем пример скриншота (рисунок 36).

Рисунок 36 – Окно работающего приложения JSF

Итак, последовательно разберем этот пример. JSF-приложение содержит главную точку входа – индексный файл index.xhtml. Этот файл находится на сервере (web-сервером является GlassFish). Можно

Библиотека БГУИР

непосредственно обратиться к методу `getAnswer` объекта `managed bean` (управляемого компонента `JSF`) через инструкцию

```
"#{mancomp.getAnswer('Oleg 2')}"
```

Обращение происходит к методу `getAnswer` с передачей параметров. Очевидным образом получаем возможность обратиться к базе данных за нужной информацией. Чтобы выполнить задуманное, рассмотрим для начала следующее приложение. Индексный файл имеет следующий вид:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">

    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
     <h2>JSF 2.0 + Ajax Hello World Example</h2>

     <h:form>
        <h:inputText id="name"
              value="#{mancomp.name}"></h:inputText>
        <h:commandButton value="Get Student Info">
           <f:ajax execute="name" render="output" />
        </h:commandButton>

        <h2><h:outputText id="output" value="#{mancomp.sayAge}"
/></h2>
     </h:form>

    </h:body>
</html>
```

В документе определено входное текстовое поле `h:inputText id="name" value="#{mancomp.name}"`, ввод значения в это поле автоматически устанавливает значение одноименного текстового поля в управляемом компоненте (см. ниже). Кроме того, в документе указывается выходное текстовое поле `h:outputText id="output"`, в которое подставляется значение, возвращаемое в результате вызова метода `mancomp.sayAge`. Строка

```
<f:ajax execute="name" render="output" />
```

указывает, что при нажатии кнопки вызывается управляемый компонент, ему передается в качестве параметра значение входного поля `inputText`

Библиотека БГУИР

id="name", а результат помещается в выходное текстовое поле outputText id="output". Здесь реализована технология Ajax. Суть этой технологии состоит в том, что она обеспечивает вызов серверного приложения и передачу ответа на клиентский сайт без необходимости перерисовки (повторного отображения) клиентского сайта. Ответ от сервера поступает на определенный управляющий элемент клиентского сайта, который его и отображает. В нашем примере ответ от серверного управляемого JSF- компонента (называемого managed bean) направляется в текстовое поле.

Управляемый компонент JSF реализован следующим образом:

```
import javax.faces.bean.RequestScoped;
@ManagedBean
@RequestScoped
public class mancomp {
    private static final long serialVersionUID = 1L;

     private String name;

     public String getName() {
        return name;
     }
     public void setName(String name)

     {
        this.name = name;
     }
     public String getSayAge(){
        //check if null?
        if("".equals(name) || name ==null)
       {
          return "";
        }
        else
        {
          return "Ajax message : You are " + name;
        }
     }
}
```

Обратим внимание на реализацию свойства name через set/get и аналогичную реализацию метода SayAge. Результат работы приложения иллюстрирует скриншот, представленный на рисунке 37.

Библиотека БГУИР

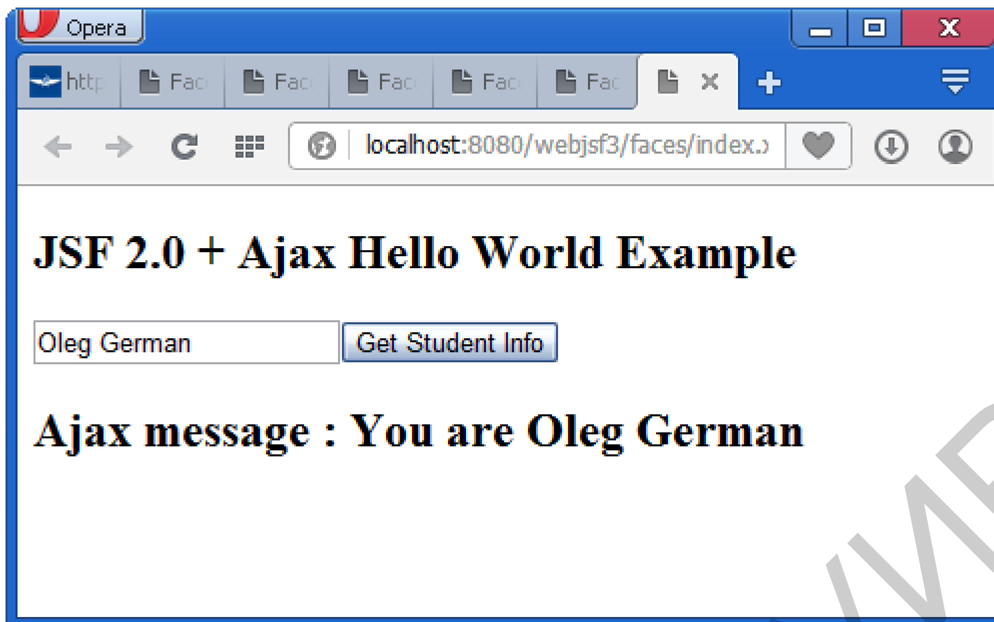


Рисунок 37 – Технология AJAX в приложении JSF

Значение (Oleg German) вводится в текстовом поле, нажимаем кнопку Get Student Info, получаем ответ в последней строке скриншота. Теперь можно подключить работу с базой данных. Изменяем текст управляемого компонента таким образом:

```
package com;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSetMetaData;

@ManagedBean
@RequestScoped
public class mancomp {
    private static final long serialVersionUID = 1L;
 private static String dbURL =
"jdbc:derby://localhost:1527/MeineBase;user=oleg;password=german
";
        private static String tableName = "STUD";

        private static Connection conn = null;
        private static Statement stmt = null;

     private String name;
     public String getName() {
        return name;
```


Библиотека БГУИР

```
      }
      public void setName(String name) {
         this.name = name;
      }
      public String getSayAge(){
         //check if null?

             int age=-1;
         if("".equals(name) || name ==null){
           return "";
         }
              else{
                try
                {
                createConnection();

                }
                catch(Exception e)
                {

                }

                try
                {
                age=selectStud(name);
                }
                catch(Exception e2)
                {

                }
                try
                {
                shutdown();
                }
                catch(Exception ee3)
                {

                }
return "Ajax message : You are " + name+"  Your age is "+age;
         }
      }

            private static void createConnection()
    {
        try
        {

Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance(
);

            conn = DriverManager.getConnection(dbURL);
```


Библиотека БГУИР

```
        }
        catch (Exception except)
        {
            except.printStackTrace();
        }
    }

    private static int selectStud(String znam)
    {
        int age=-1;
        try
        {
            stmt = conn.createStatement();
            ResultSet results = stmt.executeQuery("select * from
" + tableName+ " where FIO='"+znam+"'");

            while(results.next())
            {
             age = results.getInt(2);
                break;
         }
            results.close();
            stmt.close();
        }
        catch (SQLException sqlExcept)
        {
            sqlExcept.printStackTrace();
        }
         return age;
    }

    private static void shutdown()
    {
        try
        {
            if (stmt != null)
            {
                stmt.close();
            }
            if (conn != null)
            {
                DriverManager.getConnection(dbURL +
";shutdown=true");
                conn.close();
            }
        }
        catch (SQLException sqlExcept)
        {
        }
    }
}
```


Библиотека БГУИР

Работу с базой данных мы объяснили ранее. Теперь пользователю программы нужно задать имя студента в текстовом окне и нажать кнопку `Get Student Info`. Результирующий экран показан на рисунке 38.

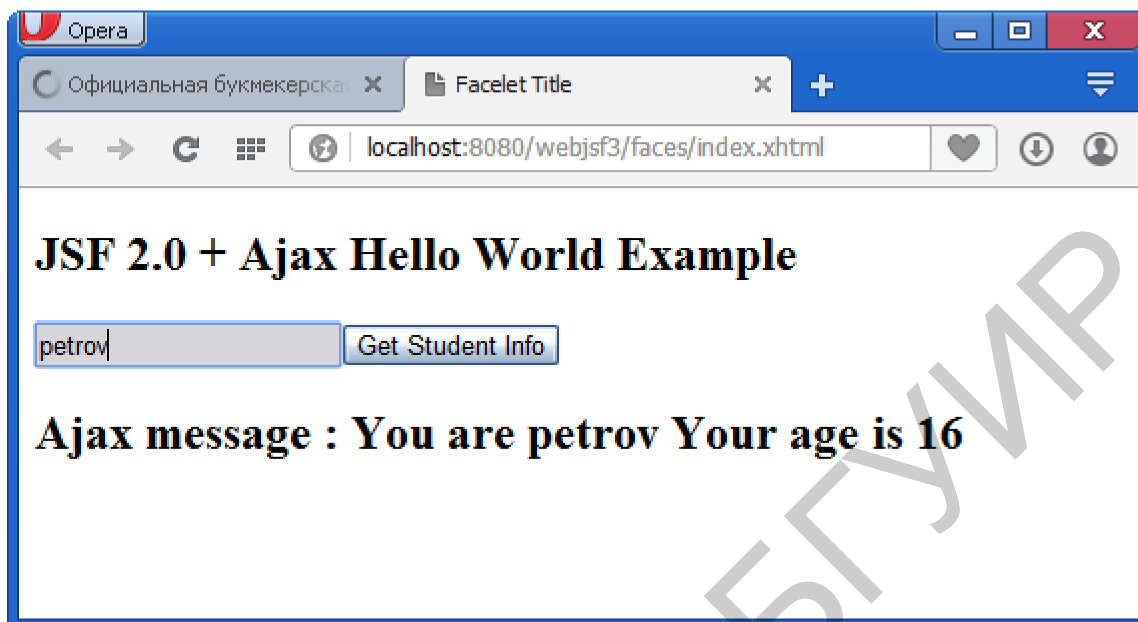


Рисунок 38 – Работа с базой данных в приложении `JSF`

Итак, в данном пункте было объяснено, как создать базу в среде `NetBeans`, заполнить ее записями и отобразить содержимое по `SQL`-запросу. Показано, как создать `web`-серверное приложение, осуществляющее поиск в таблице по данным, вводимым `web`-клиентом. При этом нами использована технология `JSF` для создания серверных страниц `Java`. Еще раз напомним, что перед работой с базой данных нужно всегда запускать сервер базы данных (в данном случае сервер `Derby`).

2.3.2 Взаимодействие Java-MySql

Более «продвинутые» возможности доставляет СУБД `MySql`. Во-первых, она бесплатная, во-вторых, она позволяет использовать хранимые процедуры и функции пользователя, в-третьих, она позволяет размещать и хранить записи средних по размеру организаций. Для работы с `MySql` нужно скачать и установить драйвер:

```
mysql-connector-java-gpl-5.1.34.msi
```

Следует обратить внимание, что при работе с `MySql` нужно подключать `jar`-файл `JDBC MySQL` к проекту (рисунок 39).

Библиотека БГУИР

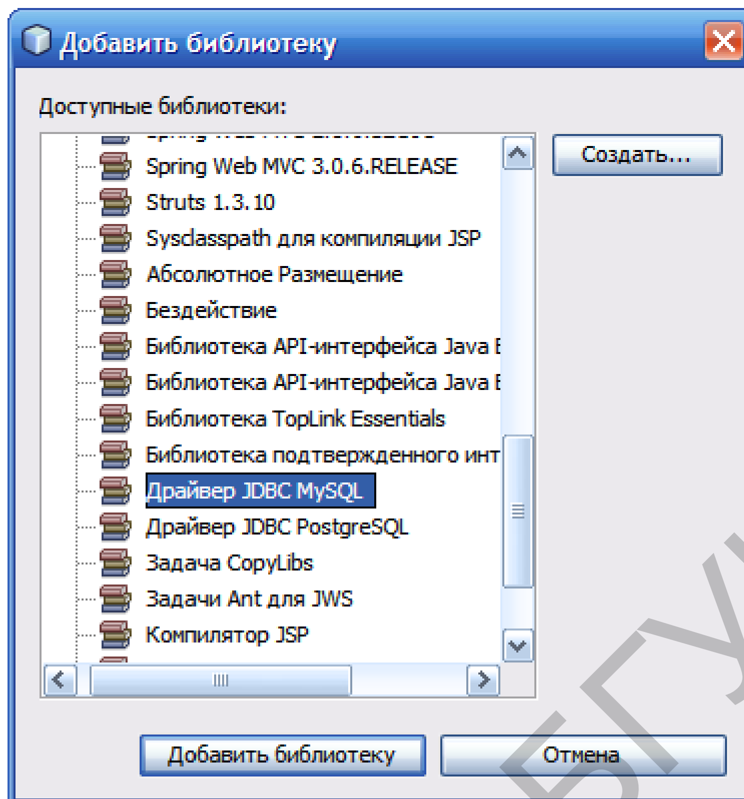


Рисунок 39 – Подключение драйвера MySQL

Для создания базы используем закладку Службы. Создание аналогично действиям, выполненным для Derby. Необходимо запустить сервер MySQL через контекстное меню (рисунок 40), «залогиниться», затем активировать контекстное меню на строке jdbc:mysql://localhost:3306... .

Библиотека БГУИР

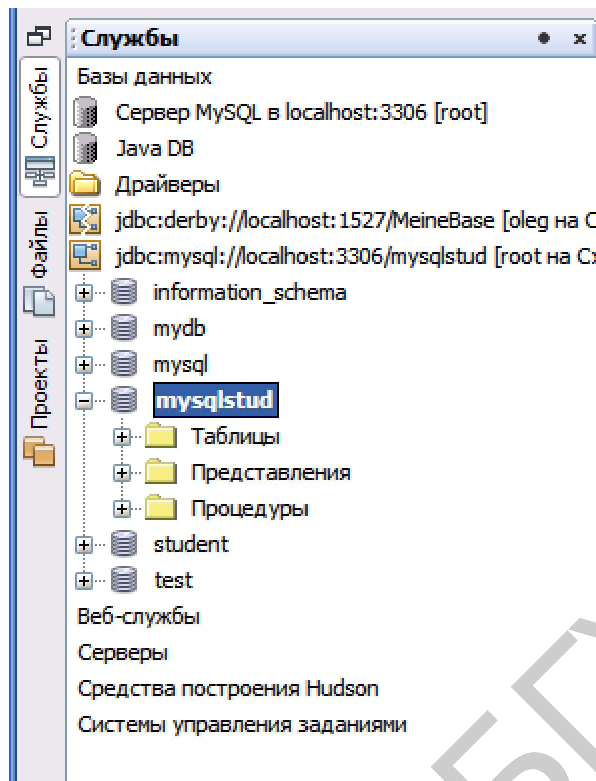


Рисунок 40 – Создание базы данных на сервере MySql

Выполним следующие команды, набрав их в редакторе:

```
use mysqlstud;
create table stud(fio varchar(20) NOT NULL PRIMARY KEY, grup
int(2));
insert into stud values('petrov',10);
insert into stud values('sidirov',10);
insert into stud values('borisov',11);
insert into stud values('mazepa',11);
```

Результат выполнения этих команд показан на рисунке 41.

select * from stud ×

#	fio	grup
1	borisov	11
2	mazepa	11
3	petrov	10
4	sidirov	10

Рисунок 41 – Результат выполнения команд SQL

Библиотека БГУИР

Теперь обратимся к дополнительным возможностям MySQL. Создадим и вызовем хранимую процедуру. Проще создать хранимую процедуру будет из консоли. Запустим MySql и убедимся в наличии заполненной таблицы stud, созданной ранее. Следующий скриншот (рисунок 42) дает пояснение.

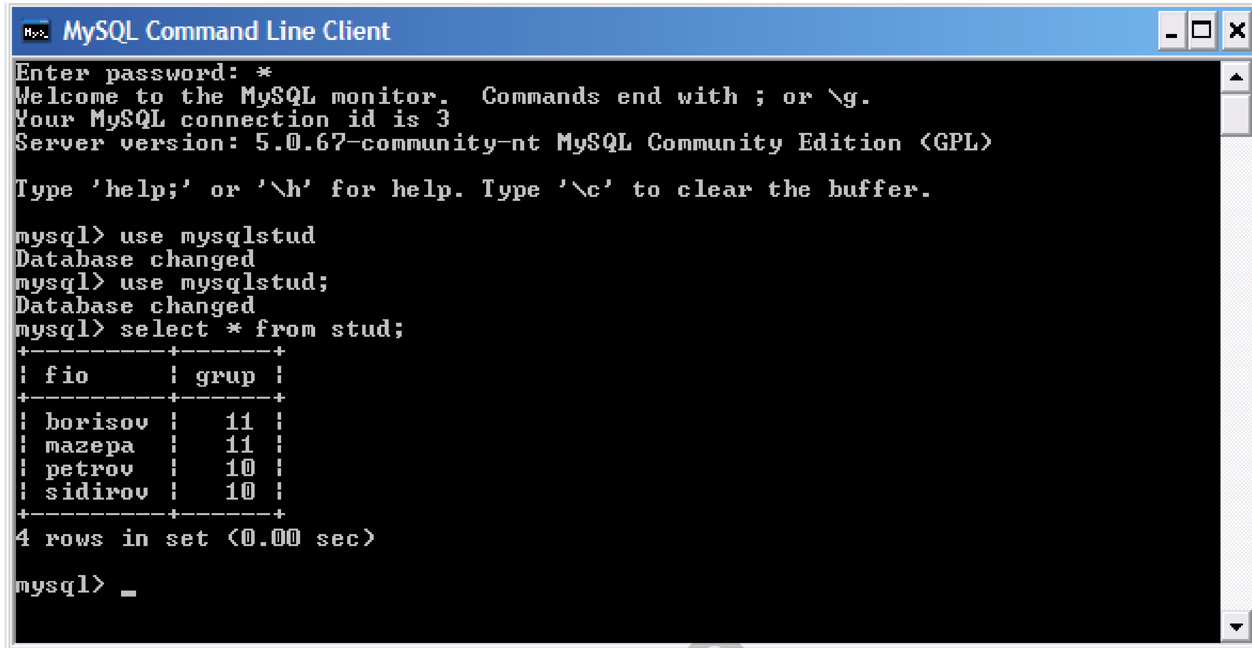


Рисунок 42 – Работа с сервером MySQL через консоль

Для создания хранимой процедуры используем следующие строки, вводимые с клавиатуры

```
delimiter  //
create procedure findgr(IN t varchar(25), OUT g int(2))
  begin
    select grup into g from stud where fio=t;
   end
  //
delimiter ;
```

Иллюстрацию набора текста хранимой процедуры дает следующий скриншот (рисунок 43).

Библиотека

```
mysql> use mysqlstud
Database changed
mysql> use mysqlstud;
Database changed
mysql> select * from stud;
+---------+------+
| fio     | grup |
+---------+------+
| borisov |   11 |
| mazepa  |   11 |
| petrov  |   10 |
| sidirov |   10 |
+---------+------+
4 rows in set (0.00 sec)

mysql> delimiter //
mysql> create procedure findgr(IN t varchar(25), OUT g int(2))
    -> begin
    ->  select grup into g from stud where fio=t;
    -> end
    -> //
Query OK, 0 rows affected (0.02 sec)

mysql> delimiter ;
mysql> _
```

Рисунок 43 – Ввод текста хранимой процедуры

Следующий скриншот показывает вызов процедуры и отображение результата ее работы (команда `select @x;`) (рисунок 44).

```
| sidirov |   10 |
+---------+------+
4 rows in set (0.00 sec)

mysql> delimiter //
mysql> create procedure findgr(IN t varchar(25), OUT g int(2))
    -> begin
    ->  select grup into g from stud where fio=t;
    -> end
    -> //
Query OK, 0 rows affected (0.02 sec)

mysql> delimiter ;
mysql> call findgr('sidirov',@x);
Query OK, 0 rows affected (0.01 sec)

mysql> select @x;
+------+
| @x   |
+------+
| 10   |
+------+
1 row in set (0.00 sec)

mysql>
```

Рисунок 44 – Отображение результата работы хранимой процедуры

Теперь все готово для написания программы на языке `Java`. Вот ее текст:

```
package labmysql;
```

```
import java.sql.*;
public class LabMySql {

        private        static        String        dbURL        =
"jdbc:mysql://localhost:3306/mysqlstud; user=root;password=1";
      //  private static String tableName = "STUD";

        private static Connection conn = null;
        private static  CallableStatement stmt = null;

    public static void main(String[] args) {

   try{
      //Register JDBC driver

      Class.forName("com.mysql.jdbc.Driver").newInstance();
      //Open a connection
      System.out.println("Connecting to database...");

       conn = DriverManager

     .getConnection("jdbc:mysql://localhost:3306/mysqlstud","root
", "1");

      //Execute a query
      System.out.println("Creating statement...");
      String sql = "{call findgr (?, ?)}";
      stmt = conn.prepareCall(sql);

      //Bind IN parameter first, then bind OUT parameter
      String t="petrov";
      stmt.setString(1, t);

      // Because second parameter is OUT so register it
      stmt.registerOutParameter(2, java.sql.Types.INTEGER);

      //Use execute method to run stored procedure.
      System.out.println("Executing stored procedure..." );
      stmt.execute();

      String answer =""+ stmt.getInt(2);
      System.out.println("The group of " +t+"  is- "+answer);

      stmt.close();
      conn.close();
   }catch(SQLException se){
      //Handle errors for JDBC
      se.printStackTrace();
   }catch(Exception e){
      //Handle errors
      e.printStackTrace();
   }finally{
```

Библиотека БГУИР

```
        //finally block used to close resources
        try{
           if(stmt!=null)
              stmt.close();
        }catch(SQLException se2){
        }// nothing we can do
        try{
           if(conn!=null)
              conn.close();
        }catch(SQLException se){
           se.printStackTrace();
        }
     }
     System.out.println("OK");
}}
```

Результат работы программы показан на рисунке 45.

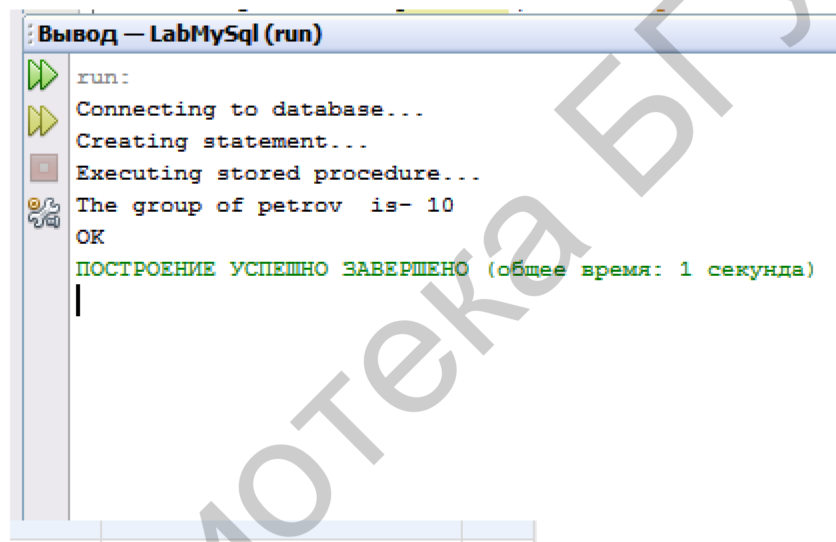


Рисунок 45 – Ввод текста хранимой процедуры из программы Java

Строка для вызова процедуры имеет такой вид:

```
String sql = "{call findgr (?, ?)}";
```

Знаком вопроса отмечены места для подстановки параметров, передаваемых в процедуру. Первый параметр является входным. Он устанавливается таким образом:

```
String t="petrov";
stmt.setString(1, t);
```

Второй параметр является выходным. Его нужно просто зарегистрировать:

Библиотека БГУИР

```
stmt.registerOutParameter(2, java.sql.Types.INTEGER);
```

Процедура выполняется в следующей команде:

```
stmt.execute();
```

Значение результирующего второго параметра получаем в строке

```
String answer =""+ stmt.getInt(2);
```

2.3.3 Взаимодействие Java-Access

Прямой доступ к таблице ACCESS выполняет следующий код:

```
package dumpschema;
import java.sql.*;
import java.io.*;
/**
 *
 * @author Admin
 */

public class DumpSchema
{
  private static final String ODBC_DRIVER =
"sun.jdbc.odbc.JdbcOdbcDriver";
  private static final String ODBC_SOURCE =
"jdbc:odbc:Driver={Microsoft Access Driver
(*.mdb)};DBQ=e:/work5/mydata.mdb";

  public static void main(String[] args)
  {
    new DumpSchema();
  }

  public DumpSchema()
  {

    Connection ODBCConnection=null;
      try
    {
      Class.forName( ODBC_DRIVER);

      ODBCConnection = DriverManager.getConnection(
ODBC_SOURCE);

      Statement query = ODBCConnection.createStatement();
      ResultSet results =
      query.executeQuery("SELECT * from stud");
```


Библиотека БГУИР

```
while (results.next()) {
  String fname = results.getString("fio");
  int lname = results.getInt("groupp");
  System.out.println("Found user \"" + fname + " " + lname +
"\"");
                                  }
         }

       catch(Exception e2)
          {
            System.out.println( "Unable to get column
information for table "+
                e2);
          }
     }
  }
```

Обращаем внимание на строку соединения

```
  private static final String ODBC_SOURCE =
"jdbc:odbc:Driver={Microsoft Access Driver
(*.mdb)};DBQ=e:/work5/mydata.mdb";
```

Результат работы программы помещен на рисунке 46.

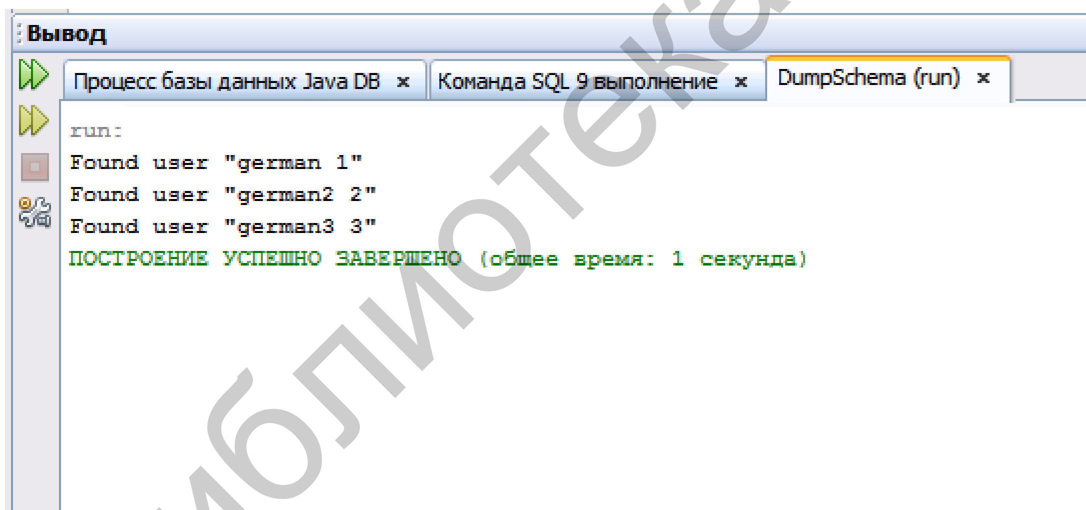


Рисунок 46 – Ввод данных из Access-таблицы в программе Java

Пример. Вывод информации по таблицам базы данных Access. Выводится информация по всем таблицам, содержащимся в указанной базе данных (в примере – это mydata.mdb).

```
package dumpschema;
import java.sql.*;
import java.io.*;
```

Библиотека БГУИР

```
public class DumpSchema
{
  // JDBC Stuff
  private static final String ODBC_DRIVER =
"sun.jdbc.odbc.JdbcOdbcDriver";
  private static final String ODBC_SOURCE =
"jdbc:odbc:Driver={Microsoft Access Driver
(*.mdb)};DBQ=e:/work5/mydata.mdb";

  public static void main(String[] args)
  {
    new DumpSchema();
  }

  public DumpSchema()
  {

    Connection ODBCConnection=null;
      try
    {
      Class.forName( ODBC_DRIVER);

      ODBCConnection = DriverManager.getConnection(
ODBC_SOURCE);
      DatabaseMetaData metadata = ODBCConnection.getMetaData();
      ResultSet tableNames = metadata.getTables( null, null,
null, new String[] {"TABLE"});
        while( tableNames.next())
        {
          String table = tableNames.getString( 3);
          try
          {
            ResultSet columnNames = metadata.getColumns( null,
null, table, null);
            System.out.println( "TABLE: " + table);
            while( columnNames.next())
            {
              System.out.println( "\t" +
columnNames.getString(4));  //Field name
              System.out.println( "\t\tDescription: " +
columnNames.getString(12));
              System.out.println( "\t\tType: " +
columnNames.getString(6));
              System.out.println( "\t\tSize: " +
columnNames.getString(7));
              System.out.println( "\t\tDecimal Digits: " +
columnNames.getString(9));
              System.out.println( "\t\tAllow Nulls? " +
columnNames.getString(18));
              System.out.println( "\t\tDefault Value: " +
columnNames.getString(13));
            }
```

```
            columnNames.close();
          }
          catch(Exception e)
          {
            System.out.println( "Unable to get column
information for table '"
                + tableNames.getString(3) + "'\n" + e);
          }
        }
        tableNames.close();

         }

       catch(Exception e2)
          {
            System.out.println( "Unable to get column
information for table "+
                e2);
          }
 }

 }
```

Эта последняя версия программы дает информацию по таблице `STUD`:

```
run:
TABLE: stud
     fio
          Description: null
          Type: VARCHAR
          Size: 20
          Decimal Digits: null
          Allow Nulls? YES
          Default Value: null
     groupp
          Description: null
          Type: BYTE
          Size: 3
          Decimal Digits: 0
          Allow Nulls? YES
          Default Value: null
ПОСТРОЕНИЕ УСПЕШНО ЗАВЕРШЕНО (общее время: 1 секунда)
```

2.4 Работа с web-ресурсами

Основными ресурсами Интернета являются документы `HTML`, файлы, серверные и сервисные программы и др. Актуальна задача доступа к сайтам и выполнения контентного поиска. Начнем с того, что покажем, как получить содержимое документа:

Библиотека БГУИР

```
     package loadfile;

 import java.io.BufferedInputStream;
 import java.io.BufferedOutputStream;
 import java.io.File;
 import java.io.FileOutputStream;
 import java.io.IOException;
 import java.net.URL;
 import java.net.URLConnection;
 import java.nio.charset.MalformedInputException;

public class LoadFile {
    public static void main(String[] args) {

       URL url = null;
       URLConnection con = null;
       int i;
         try {
          url = new URL("file:/e:/work5/my.html");
          System.out.println("before open URL");
          con = url.openConnection();
          File file = new File("e:/work5/my2.html");
          System.out.println("after my2.html");
   BufferedInputStream bis = new BufferedInputStream(

con.getInputStream());
   BufferedOutputStream bos = new BufferedOutputStream(
                       new FileOutputStream(file));
   String bhtml="";

        while ((i = bis.read()) != -1) {

           bos.write(i);
           bhtml+=(char)i;

                                   }
    bos.flush();
    bis.close();

String htmlcontent= new String(bhtml);
System.out.println("file was copied");
System.out.println("It is as follows\n"+htmlcontent);
                                          }

catch (MalformedInputException malformedInputException)
{
malformedInputException.printStackTrace();
                                                                  }
catch (IOException ioException) {

ioException.printStackTrace();
         }
```

```
        catch(Exception e)
        {
           System.out.println("error "+e.getMessage());
        }
    }
}
```

В данном примере считывается содержимое файла `e:/work5/my.html`. Этот документ содержит следующий текст

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>File Upload</title>
        <meta    http-equiv="Content-Type"    content="text/html;
charset=UTF-8">
    </head>

     <body>
        <form method="POST" action="upload"
enctype="multipart/form-data" >
            File:
            <input type="file" name="file" id="file" /> <br/>
            Destination:
            <input type="text" value="/tmp" name="destination"/>
            </br>
            <input  type="submit"  value="Upload"  name="upload"
id="upload" />
        </form>
    </body>
</html>
```

Для считывания необходимо использовать объект типа `URLConnection con`. Открытие соединения и подключение к файлу выполняется в следующем фрагменте кода:

```
        try
        {
         url = new URL("file:/e:/work5/my.html");
         System.out.println("before open URL");
         con = url.openConnection();
        ...........
```

Далее файл считывается через буферизованный входной поток и записывается в новый файл:

```
   BufferedInputStream bis = new BufferedInputStream(
   con.getInputStream());
   BufferedOutputStream bos = new
   BufferedOutputStream(new FileOutputStream(file));
```


Библиотека БГУИР

```
String bhtml="";
     while ((i = bis.read()) != -1)
  {
        bos.write(i);
        bhtml+=(char)i;
  }
```

Параллельно формируем текстовую строку bhtml, куда добавляем считываемые символы.

В итоге получаем следующий выходной скриншот с результатами работы программы (рисунок 47).

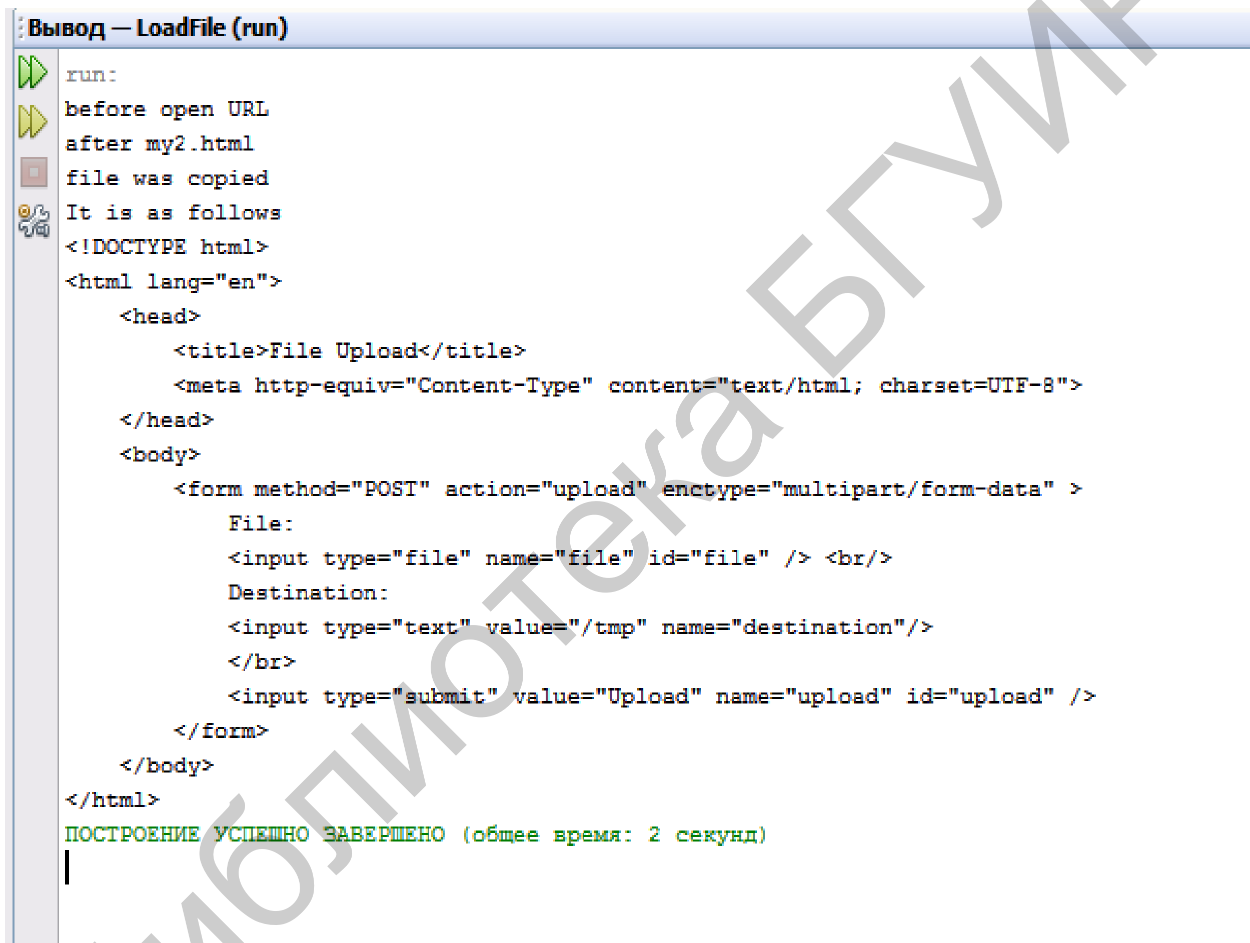


Рисунок 47 – Отображение содержимого считанного html-файла

На первых этапах создания распределенных систем использовали технологию cgi (common gateway interface). Ее суть сводилась к запуску скриптов (например, на языках perl, asp и др.) или exe-файлов (например, написанных на Delphi) на стороне сервера при указании их url. Следует заметить, что Java-сервер не запускает exe-файлы (в его компетенции сервлеты, jsp-страницы, web-сервисы и ejb-компоненты). Используя класс URL Java, можно все же привести пример подобной реализации. Идея

Библиотека БГУИР

заключается в доступе через `url` к `exe`-файлу и его запуску. Фактически мы повторим общую идею предыдущего примера.

```
package cgilib;
import java.net.*;
import java.io.*;

public class CgiLib {

    public static void main(String[] args)
    {
      BufferedReader     rdr=     new     BufferedReader(     new
InputStreamReader(System.in));
      String b_url="file:/e:/work/calendar.exe";
      String adr_url=b_url;
      System.out.println("Sending Request to:"+adr_url);
        try
        {
            URL u=new URL(adr_url);
            InputStream in_stream=u.openStream();
            BufferedInputStream buf=new
BufferedInputStream(in_stream);
            int dat;

               String bhtml="";
               File file = new File("e:/work/calen_COPY.exe");
               BufferedOutputStream bos = new
                BufferedOutputStream(new
FileOutputStream(file));
            while((dat=buf.read())!=-1)
            {
                          bos.write(dat);
                          bhtml+=(char)dat;
            }

            System.out.println();
            System.out.println("File  from  server  got.  Press  any
key to run it");
            bos.close();
            Runtime rt=Runtime.getRuntime();
            Process proc = rt.exec(""+file);
            proc.waitFor();
        }
        catch(Exception e)
        {
          System.out.println("Connection
error:"+e.getMessage());
        }
    }
}
```


Библиотека БГУИР

В этом примере производится считывание исполняемого файла (отображающего календарь) по адресу

```
file:/e:/work/calendar.exe
```

Заметим, что адрес может указывать на удаленный компьютер в общем случае. Выполняемый файл читается в массив байтов и затем создается его копия, которая и запускается на выполнение из строк

```
Runtime rt=Runtime.getRuntime();
Process proc = rt.exec(""+file);
```

Последняя из приведенных команд и запускает приложение календаря (рисунок 48).

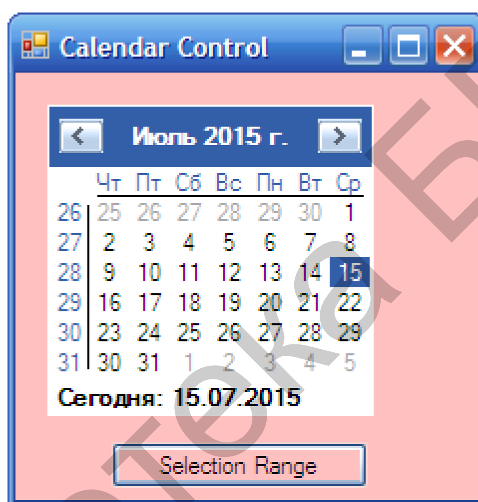


Рисунок 48 – Запуск считанного исполняемого файла

Для просмотра документов можно использовать объект типа браузер, как показано ранее. Можно программно создать объект-браузер или запустить соответствующий файл. Рассмотрим сначала вторую возможность. Реализацию класса `LoadFile`, приведенную выше в этом разделе, сохраним и добавим лишь следующий фрагмент после последнего блока `catch`:

```
         try{
              Runtime rt=Runtime.getRuntime();
              Process proc = rt.exec("c:\\Program
Files\\Google\\Chrome\\Application\\Chrome.exe "+newfile);
              proc.waitFor();
             }
         catch(Exception ew)
         {
 System.out.println("error in start browser  "+ew.getMessage());
         }}}
```


Библиотека БГУИР

Здесь строка `String newfile="e:/work5/my2.html"` дает адрес документа. Объект `Runtime rt` позволяет запускать другие приложения. В примере запускаем браузер `Chrome.exe` и передаем ему в качестве аргумента путь к документу и имя документа.

Следующая программа показывает, как программно создать и использовать браузер в `Java`.

Для создания объекта типа браузера нужно к базовому классу подключить интерфейс `HyperlinkListener`. Браузер будет открывать указываемый ему сайт на панели `JEditorPane`, используя параметр-объект типа `URL`:

```
URL  url=new URL(String initialUrl);
JEditorPane htmlPane= new JEditorPane(url);
htmlPane.setPage(url);
```

Для навигации по гиперссылкам необходимо подключить прослушиватель:

```
htmlPane.addHyperlinkListener(this);
```

Мы будем выполнять навигацию так: набираем в текстовом поле новый `URL`, нажимаем клавишу `Enter` и обрабатываем событие от текстового поля так:

```
              Public void actionPerformed(Actionevent ae)
   {
     String url;
      if(ae.getSource()== urlField)

       {  url= urlField.getText();
           JOptionPane. showMessageDialog(null, "url="+url);
               try
                         {
                            htmlPane.setPage(new URL(url));
                 JOptionPane.showMessageDialog(null,
"ActivatedFrom>"+url);
                          }

     catch (IOException io)
                 {

                 }
else
       ….. …. ….
```

Приводим полный текст программы простого браузера в `Java`

```
package browser;
```


Библиотека БГУИР

```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;


public    class    Browser       extends    JFrame    implements
HyperlinkListener, ActionListener{

     public static void main(String[]args)
       {
      Browser    br=    new    Browser("file:/e:/work5/my.html");
//указываем, какой сайт открыть
      br.setSize(600,800);
      br.setBackground(new Color(100,200,200));
      br.setVisible(true);
       }

   private JButton b1;
   private JTextField urlField;
   private JEditorPane htmlPane;
   private String initialUrl;

   public Browser(String initurl)
    {
      this.initialUrl= initurl;
      JPanel topPanel = new JPanel();
      urlField=new JTextField(40);
      urlField.setText(this.initialUrl);
//адрес сайта в текстовое поле
      urlField.addActionListener(this);
//нажатие кнопки вызывает смену сайта
      topPanel.add(urlField);
      b1= new JButton("EXIT");
      b1.addActionListener(this);
      topPanel.add(b1);
      Container knt=getContentPane();
      knt.add(topPanel,BorderLayout.NORTH);

        try
         {

       JOptionPane.showMessageDialog(null,"Try to create URL");
             URL url = new URL(this.initialUrl);


JOptionPane.showMessageDialog(null,""+url.toExternalForm());
             htmlPane= new JEditorPane(url);
            //htmlPane  отображает сайт
             htmlPane.setEditable(false);
```


Библиотека БГУИР

```
            //содержимое сайта нельзя редактировать
       htmlPane.addHyperlinkListener(this);
            //этот прослушиватель реагирует на
            //смену сайтов
             htmlPane.setBackground(new Color(200,200,200));
             htmlPane.setPage(url);
            //отображаем сайт этой командой
         JScrollPane sp= new JScrollPane(htmlPane);
            //полоса прокрутки для больших //сайтов
         getContentPane().add(sp, BorderLayout.CENTER);
           }

       catch(IOException ioe)
          {}
      }

     public void  actionPerformed(ActionEvent ae)
        //обработчик событий от кнопки
        //и текстового поля
           {

            String url;
            if(ae.getSource()==urlField)
//событие возникло при  нажатии клавиши для
 //активного текстового поля с адресом URL

            {
             url=urlField.getText();
             JOptionPane.showMessageDialog(null,"url="+url);
              try{
               htmlPane.setPage(new URL(urlField.getText()));

//смена сайта по этой
//команде
                   }
              catch (IOException ioe)
                 {}

             }
                 else
         if (ae.getSource()==b1)
              System.exit(0);
               }

        public void hyperlinkUpdate(HyperlinkEvent hle)
                //это обработчик событий от
                //смены сайта.Просто выводим сообщение
             {

             if(hle.getSource()==
HyperlinkEvent.EventType.ACTIVATED)
              {
```

Библиотека БГУИР

```
JOptionPane.showMessageDialog(null,"NewURLLoaded");
                }}
            }
```

После запуска программы вводим URL-адрес документа в текстовом поле и нажимаем клавишу ENTER. Подтверждаем переход (рисунок 49).

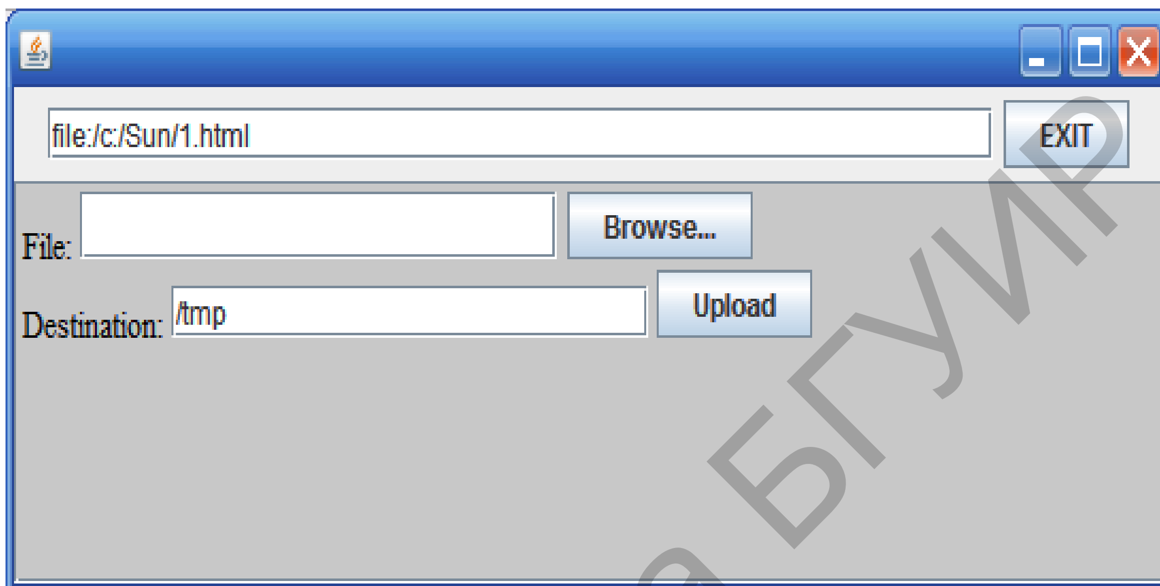


Рисунок 49 – Простой браузер

Результат открытия нового документа представлен на рисунке 50.

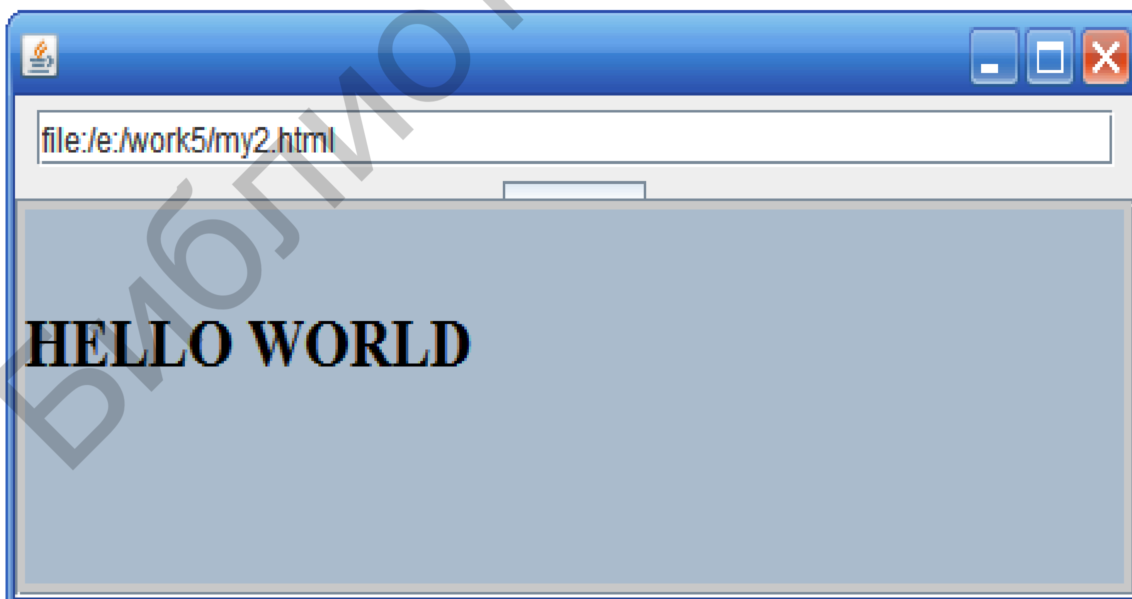


Рисунок 50 – Открытие нового документа в браузере

Библиотека БГУИР

Доступные online-сервисы Интернета предоставляют достаточно широкий спектр возможностей. Например, можно выполнить перевод текста, поиск документа, произвести математические расчеты, «озвучить» текст, узнать погоду и т. п. Пример подобного сервиса дает online-кодирование. Этот сервис дает возможность писать программы, не имея установленного в системе компилятора языка и среды программирования. Воспользуемся таким сервисом по адресу ideone.com. Выберем для написания кода язык Java (скриншот представлен на рисунке 51).

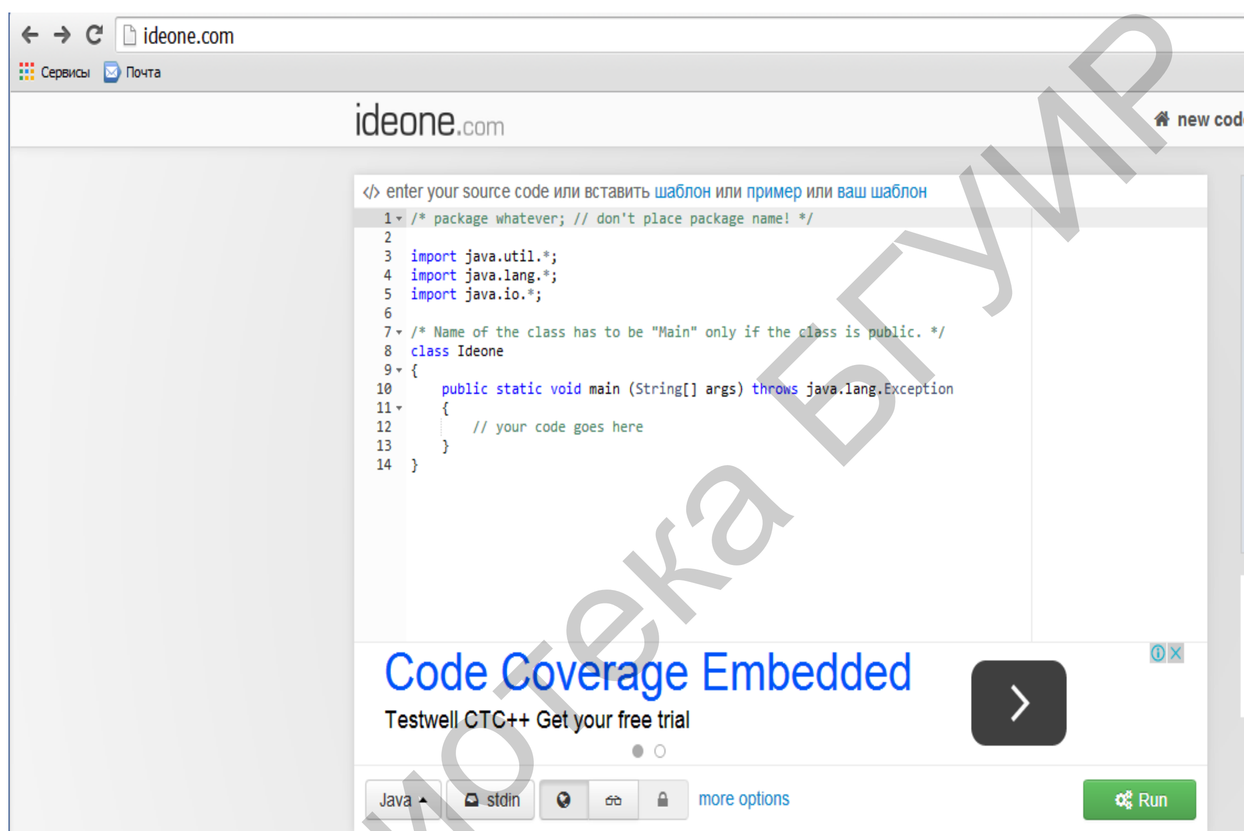


Рисунок 51 – Online-ресурс для набора и выполнения Java-программ

Наберем текст и выполним его (кнопка Run). Результат выводится в нижней части окна (рисунок 52).

Библиотека БГУИР

ideone.com new cod

Повсюду ищут то, что вы предлагаете.
Google

редактировать fork скачать copy

```
1.  /* package whatever; // don't place package name! */
2.
3.  import java.util.*;
4.  import java.lang.*;
5.  import java.io.*;
6.
7.  /* Name of the class has to be "Main" only if the class is public. */
8.  class Ideone
9.  {
10.     public static void main (String[] args) throws java.lang.Exception
11.     {
12.        System.out.println("Hello, once more,once more, once more!!!!");
13.     }
14. }
```

Успешно #stdin #stdout 0.09s 320256KB comments (0)

stdin

```
Standard input is empty
```

stdout copy

```
Hello, once more,once more, once more!!!!
```

Рисунок 52 – Вывод результатов выполнения Java-программы

Богатые возможности online-программирования предоставляет платформа облачных вычислений (правда, эта услуга платная с trial-версией).

2.5 Сервлеты и JSP-страницы

Сервлет – это Java-приложение на стороне сервера, которое вызывается из клиента посредством взаимодействия с web-сервером. JSP-страницы (Java Server Pages) избавляют пользователя от необходимости писать в сервлете фрагмент возвращаемого на сторону клиента HTML-кода. Идея JSP-страниц состоит в том, чтобы в обычный HTML-документ вставить команды языка Java. Для этой цели используют JSP-теги (скриплеты):

Библиотека БГУИР

- <% -- комментарий;
- <%! объявление переменных и методов;
- <%= получение и вставка значения;
- <%@ page import подключение внешнего файла (класса).

JSP-страницы сохраняются в файлах с расширением jsp. Компилировать их не надо. Они, как и сервлеты, вызываются сервером. Приведем пример JSP-страницы:

```
<html>
<body bgcolor=#aabbee>
<H1> Hello from JSP</H1>
<Hr></HR>
<%@  page import="java.util.*" %>
<%@  page import="java.awt.*" %>
<%! String getDate()  {
                     Date dt=new Date();
                     return dt.toLocaleString(); }%>

TODAY IS:<%=getDate() %>
<HR></HR>
</body>
</html>
```

Здесь теги

```
<%@  page import="java.util.*" %>
<%@  page import="java.awt.*" %>
```

подключают пакеты java.util.* и java.awt.*. В тегах

```
<%! String getDate()  {
                     Date dt=new Date();
                     return dt.toLocaleString(); }%>
```

реализуется метод getDate. Вызов этого метода помещен в <%=getDate() %>

Чтобы реализовать данную страницу в проекте NetBeans, нужно создать проект типа web-приложение, добавить к нему эту страницу и сделать ее главной. Подобный проект мы создавали ранее. Поэтому, опуская сопутствующие замечания, создаем проект с именем MyJsp. Перепишем файл index.jsp следующим образом:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
```


Библиотека БГУИР

```
        <meta    http-equiv="Content-Type"    content="text/html;
charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
       <%
        String redirectURL = "jspE.jsp";
        response.sendRedirect(redirectURL);
       %>
    </body>
</html>
```

Здесь производится перенаправление на новую страницу в строках скриплета

```
<%
        String redirectURL = "jspE.jsp";
        response.sendRedirect(redirectURL);
%>
```

Результат обращения к данной странице приведен на рисунке 53.

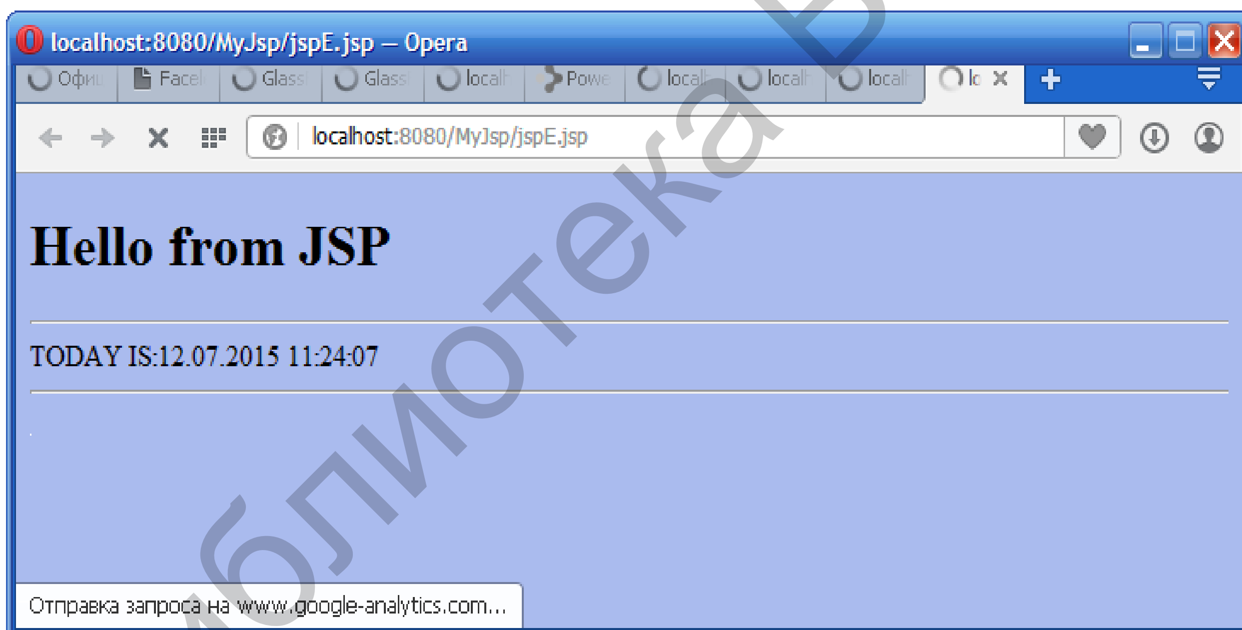


Рисунок 53 – Работа jsp-страницы

Можно изменить стартовую конфигурацию проекта и обойтись без команд перенаправления просмотра. Для этого добавим в проект конфигурационный файл web.xml следующего содержания:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javae
http://java.sun.com/xml/ns/javaee/web-app 3 0.xsd">
```


Библиотека БГУИР

```
 <welcome-file-list>
     <welcome-file>
     jspE.jsp
     </welcome-file>
</welcome-file-list>
     </web-app>
```

Стартовый файл проекта указывается в тегах

```
<welcome-file-list>
      <welcome-file>
      jspE.jsp
      </welcome-file>
 </welcome-file-list>
```

Место создания файла иллюстрируется следующим скриншотом (рисунок 54).

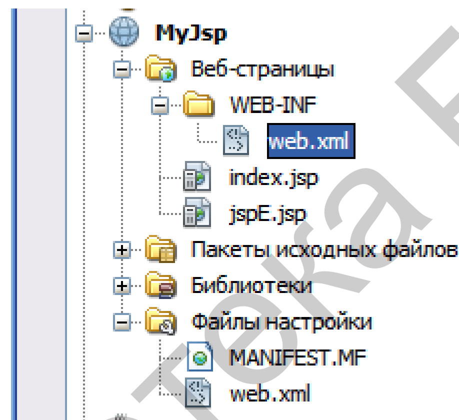


Рисунок 54 – Дислокация файла `web.xml`

Библиотека БГУИР

Этот файл создаем вручную.

В `JSP`-скриптах доступны объекты `out`, `response` и `request` для получения и передачи данных на форму клиента. Следующий пример дает некоторую иллюстрацию.

```
<html>
<body bgcolor=#aabbee>
<H1> Hello from JSP</H1>
<HR></HR>
<%@  page import="java.util.*" %>
<%@  page import="java.awt.*" %>
<%@  page import="java.io.*" %>
<%@  page import ="javax.servlet.*"%>
<%@  page import ="javax.servlet.http.*"%>
<%! String getDate()  {
                     Date dt=new Date();
                     return dt.toLocaleString(); }%>
```

```
<%
PrintWriter ot= response.getWriter();
String s=getDate();
ot.println("TODAY IS:"+s); %>
<HR></HR>
</body>
</html>
```

Обратим внимание на то, как выполняются операторы Java

```
<%
PrintWriter ot= response.getWriter();
String s=getDate();
ot.println("TODAY IS:"+s);
%>
```

Перейдем теперь к сервлетам. Построим сервлет, который будет выдавать возраст студента по фамилии студента. При этом используем ранее созданную базу данных. Итак, создаем web-приложение, даем ему имя ServletStud (рисунок 55).

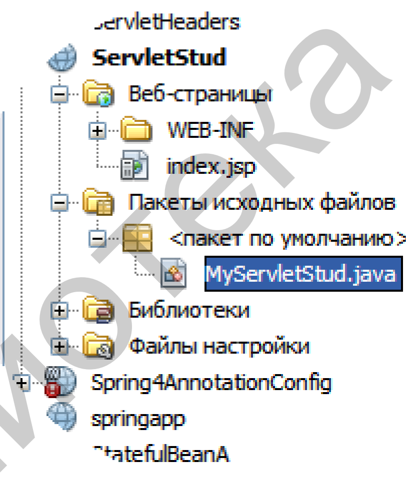


Рисунок 55 – Исходная структура проекта

Добавляем класс сервлета в узле <Пакет по умолчанию> через контекстное меню. Имя класса сервлета – MyServletStud.java. Пока ничего в текст сервлета не добавляем. Создадим далее файл html с именем Start.html. Вызываем контекстное меню на узле web-страницы, выбираем пункт Создать – HTML. Запишем следующий код документа Start.html:

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
```

Библиотека БГУИР

```
        <meta    http-equiv="Content-Type"    content="text/html;
charset=UTF-8">
    </head>
    <body bgcolor="#aaccff">
        <h2> Форма для получения сведений</h2>
        <br>
        <form name="frm" method="Get" action="MyServletStud">
            ФАМИЛИЯ (ввести вручную):<Select name ="sel">
                <option name="o1" value="petrov">petrov</option>
            <option name="o2" value="sidorov">sidorov</option>
            <option name="o3" value="abasov">abasov</option>

            </select>
Ответ СЕРВЛЕТА: <input type="text" name ="tfans" value="    "/>
        <h4>   Кликни   здесь   после   ввода   фамилии   :<Input
type="submit" value="Вызов сервлета"/>
        </h4>
        </form>
    </body>
</html>
```

Если открыть этот документ в браузере, то получим следующий вариант окна, изображенный на рисунке 56.

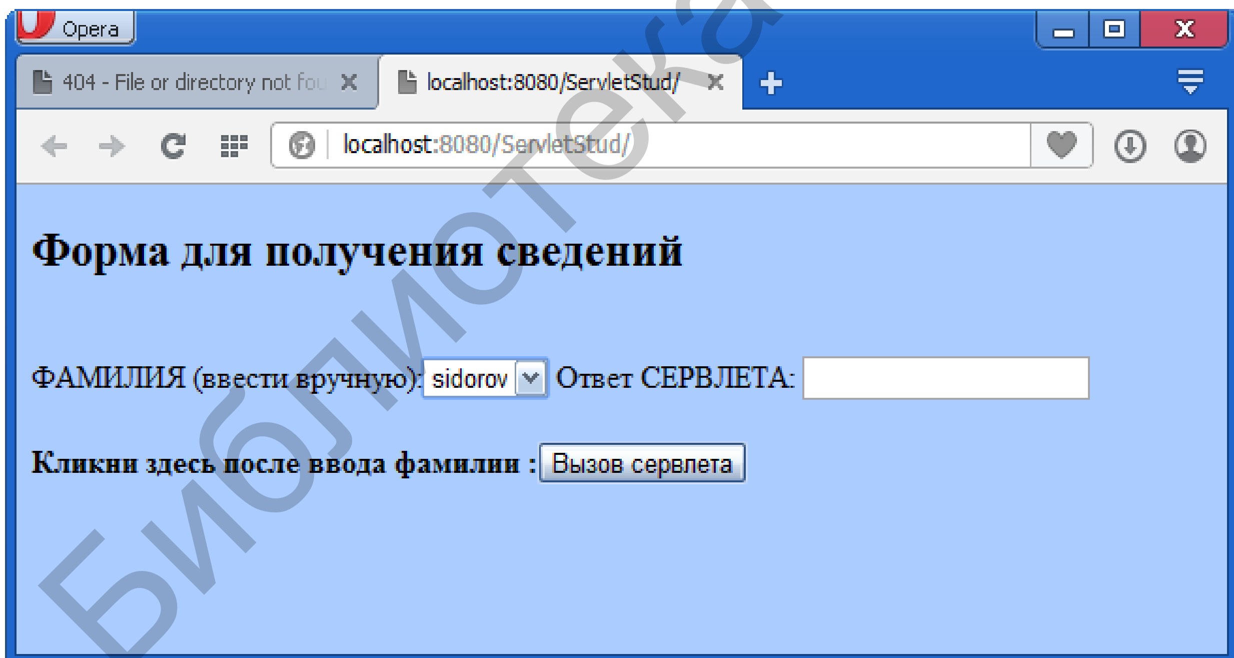


Рисунок 56 – Окно стартового документа

Теперь нам необходимо корректно задать содержимое конфигурационного файла `web.xml`. Оно должно быть таким:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <servlet>
        <servlet-name>MyServletStud</servlet-name>
        <servlet-class>MyServletStud</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>MyServletStud</servlet-name>
        <url-pattern>/MyServletStud</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>
            Start.html
         </welcome-file>
     </welcome-file-list>
     <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
</web-app>
```

Минимальное приложение собрано. Теперь можно его выполнить, для чего активируем контекстное меню на имени проекта, выбираем сначала опцию `Очистить и построить`, затем опцию – `Развернуть`, затем опцию `Выполнить`. Результат показан на рисунке 57 (активизирован сервлет и получен результат его работы в форме страницы `html`).

Рисунок 57 – Выходное окно сервлета

Библиотека БГУИР

В сервлете нужно сделать три вещи: получить данные из формы, обратиться в базу с фамилией студента и получить его возраст и вернуть результат. Заметим, что для установления соединения с базой данных нужно подключить архивный файл с драйвером `derbyClient.jar` к проекту (см. пункт 2.3.1). В соответствии с этими задачами текст сервлета переделаем следующим образом:

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSetMetaData;


public class MyServletStud extends HttpServlet {

       private         static         String         dbURL         =
"jdbc:derby://localhost:1527/MeineBase;user=oleg;password=german
";
       private static String tableName = "STUD";

       private static Connection conn = null;
       private static Statement stmt = null;

    protected  void  processRequest(HttpServletRequest  request,
HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            String fio=""+request.getParameter("sel");
            String ans="";
            ans=createConnection();
             if(ans.length()<=0)
             {
               ans=selectStud(fio.trim());
             }
            shutdown();

            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet MyServletStud</title>");
            out.println("</head>");
```

Библиотека БГУИР

```
            out.println("<body bgcolor='#aaccff'");
            out.println("<form>");
            out.println("<h2>      Форма      для      получения
сведений</h2><br><br>");
            out.println("ФАМИЛИЯ     (ввести     вручную):<input
type='text' name ='tf' value='"+fio+"' />");
            out.println("Ответ СЕРВЛЕТА: <input type='text' name
='tfans' size='40' value='"+ans+ "' />");
            out.println("</form>");
            out.println("</body>");
            out.println("</html>");

        } finally {
            out.close();
        }
    }

    private static String createConnection()
    {
       String res="";
        try
        {

Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance(
);
            conn = DriverManager.getConnection(dbURL);
        }
        catch (Exception except)
        {
         res=except.getMessage();
        }
         return res;
    }

    private static String selectStud(String fio)
    {
        String res="";
        try
        {
            stmt = conn.createStatement();
            ResultSet results = stmt.executeQuery("select * from
" + tableName+" where fio='"+fio+"'");

            while(results.next())
            {
                res =""+ results.getInt(2);
                break;
            }
            results.close();
            stmt.close();
        }
        catch (SQLException sqlExcept)
```

```
        {
            res=""+sqlExcept.getMessage();
        }
            return res;
    }

    private static void shutdown()
    {   try
        {
            if (stmt != null)
            {
                stmt.close();
            }
            if (conn != null)
            {
      DriverManager.getConnection(dbURL + ";shutdown=true");
                conn.close();
            }
        }
        catch (SQLException sqlExcept)
        {
        }
    }
    @Override
    protected     void     doGet(HttpServletRequest     request,
HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    protected     void     doPost(HttpServletRequest     request,
HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
    @Override
    public String getServletInfo() {
        return "Short description";
    }// </editor-fold>
}
```

Поясним основные места этого кода. В сервлете основным обработчиком (методом) является метод `protected void processRequest`. У этого метода есть два системных (встроенных) объекта: `request` и `response`. Первый позволяет получить данные, переданные из клиентской формы (документа `HTML`):

```
String fio=""+request.getParameter("sel");
```


Библиотека БГУИР

Используется метод getParameter, аргументом которого является имя выпадающего списка в клиентской форме. Далее создаем соединение с базой данных

```
ans=createConnection();
```

Если соединение успешно создано, то выполняем обращение к базе через метод

```
ans=selectStud(fio.trim());
```

Выводим результаты в форме документа HTML обратно на сторону клиента:

```
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet MyServletStud</title>");
            out.println("</head>");
            out.println("<body bgcolor='#aaccff'");
            out.println("<form>");
            out.println("<h2>       Форма       для       получения
сведений</h2><br><br>");
            out.println("ФАМИЛИЯ      (ввести      вручную):<input
type='text' name ='tf' value='"+fio+"' />");
            out.println("Ответ СЕРВЛЕТА: <input type='text' name
='tfans' size='40' value='"+ans+ "' />");
            out.println("</form>");
            out.println("</body>");
            out.println("</html>");
```

Теперь приложение готово. Выполним его, введя любую фамилию из списка (рисунок 58).

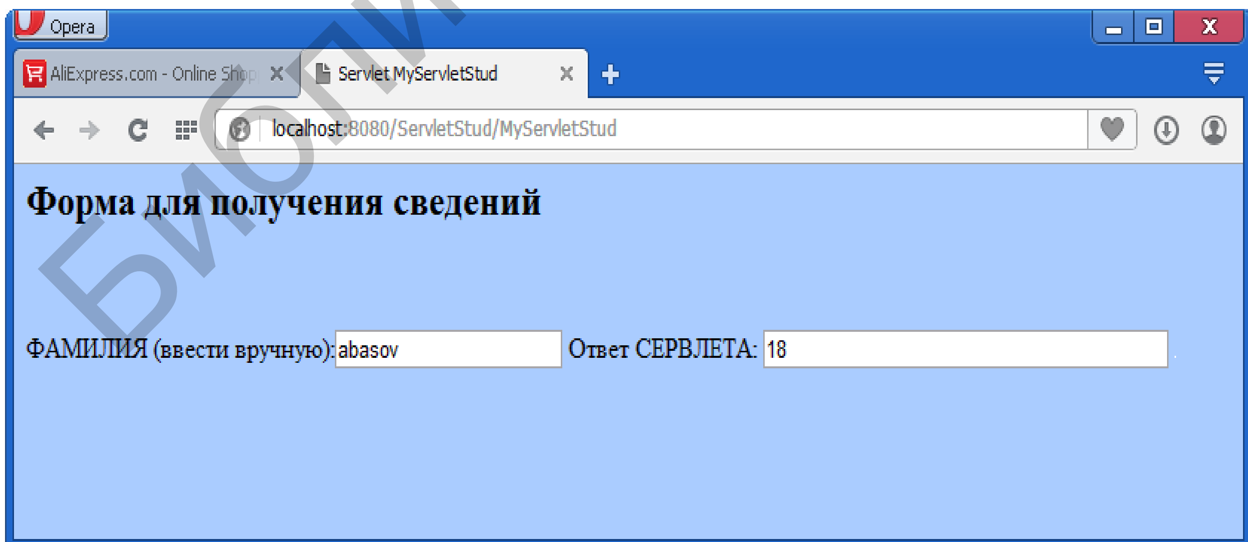


Рисунок 58 – Значение, возвращаемое сервлетом в текстовое поле

Библиотека БГУИР

Выводу тела документа из сервлета должен предшествовать вывод типа документа:

```
response.setContentType("text/html");
```

Это обстоятельство следует иметь в виду всегда, поскольку браузер по-разному интерпретирует различные виды документов.

Значения переменных формы получаем с помощью команды

```
request.getParameter("sel");
```

Здесь `sel` – это имя элемента формы (списка) документа `HTML`.

Переменная класса `HttpServletRequest` `request` позволяет прочитать все данные, формы, переданные браузером, используя команду

```
getParameter().
```

Переменная класса `HttpServletResponse` `response` позволяет вывести данные на форму клиента. Как правило, возвращается обычный `HTML`-документ:

```
out.println ("<HTML>\n"+
                "<HEAD><H1>       Получение       данных       от
формы</H1></HEAD>\n"+
               "<BODY BGCOLOR=34AAFF><BR><BR>");
  … … …
```

Как и апплет, сервлет имеет набор методов, формирующих его цикл жизни. Его составляют следующие методы:

- `init();`
- `service();`
- `destroy();`
- `doGet();`
- `doPost();`
- `doXxx().`

Метод `init` вызывается один единственный раз при инициализации сервлета. Его используют так же, как и в апплетах. Синтаксис метода такой:

```
public void init() throws ServletException
 {
 //код инициализации}
```

Имеется и вторая версия этого метода, которая позволяет считать параметры сервера. Она имеет такой вид:

Библиотека БГУИР

```
public void init(ServletConfig conf)
throws ServletException
 {
 super.init(config);
 //код инициализации
}
```

Параметры сервера помещаются в конфигурационном файле `server.xml`. Примером может служить следующий текст.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems. Inc.//DTD Web Application 2.2//EN"
    http://java.sun.com/j2ee/dtds/web-app_2.2.dtd>

<web-app>
 <servlet>
    <servlet-name>
          ShowMsg
    </servlet-name>
    <servlet-class>
      <init-param>
        <param-name>
             name
        </param-name>
         <param-value>
             ovgerman
          </param-value>
         </init-param>
<init-param>
        <param-name>
             group
        </param-name>
         <param-value>
             112
          </param-value>
         </init-param>
</servlet>
</web-app>
```

В приведенном выше файле определены два параметра: `name` и `group`. Эти параметры имеют значения, определенные в тегах:

```
<param-value>
  …
</param-value>
```

Рассмотрим, как прочитать значение параметров. Это делает следующий код.

Библиотека БГУИР

```
public void init(ServletConfig conf)
throws ServletException
 {
super.init(config);
String nameA=conf.getInitParameter("name";
String groupA=Integer.parseInt(conf.getInitParameter("group"));
… … …
}
```

Для удаления сервлета следует использовать метод `destroy()`.

Метод `service` вызывается всякий раз, когда сервлет получает управление со стороны браузера. Синтаксис этого метода следующий:

```
public void service(HttpServletRequest request,
                                HttpServletResponce response)
              throws ServletException, IOException
   {
     … … …
   }
```

Наконец, метод `doPost`(…) играет ту же роль, что и метод `doGet`. Метод `doPost` используется тогда, когда сервлет активизируется из `HTML`-сайта при указании в параметре `Method` тега `Form` значения `post`, например:

```
<Form name=myfrm  Method =”post” …>
```

В этом случае данные формы передаются серверу отдельно от заголовка запроса. В этом случае можно передавать большие массивы данных.

2.6 Отправка электронной почты

Для отправки электронной почты нужно подключить к проекту библиотечный файл `mail.jar`. Следует скачать с Интернета библиотечный архив `javamail`-1.4.3 (или более новую версию). Приложение для отправки почты приведено ниже:

```
package main;
import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;

public class Main {
    private static String user_log = "ovgerman@mail.ru";
 // Mail.ru user name from which the mail will be sent
    private static String PASSWD = "abracadabra";
// Mail password
    private static String rcpnt= "ovgerman@tut.by";
// to whoom
```

```
    public static void main(String[] args) {
        String from_str = user_log;
        String pass = PASSWD;
        String[] to_str_arr = { rcpnt };
// list of recipient email addresses
        String subject_str = "Java send e-mail example";
     String body_str = "Pleased to sent You e-mail from Java";

        sendFromMailRU (from_str, pass, to_str_arr, subject_str,
body_str);
    }

    private static void sendFromMailRU (String from, String
pass, String[] to, String subject, String body) {
        Properties props = System.getProperties();
        String host = "smtp.mail.ru";
        props.put("mail.smtp.starttls.enable", "true");
        props.put("mail.smtp.host", host);
        props.put("mail.smtp.user", from);
        props.put("mail.smtp.password", pass);
        props.put("mail.smtp.port", "587");
        props.put("mail.smtp.auth", "true");

        Session session = Session.getDefaultInstance(props);
        MimeMessage message = new MimeMessage(session);

        try {
            message.setFrom(new InternetAddress(from));
            InternetAddress[] toAddress =
              new InternetAddress[to.length];

            // To get the array of addresses
            for( int i = 0; i < to.length; i++ ) {
                toAddress[i] = new InternetAddress(to[i]);
            }

            for( int i = 0; i < toAddress.length; i++) {
                message.addRecipient(Message.RecipientType.TO,
toAddress[i]);
            }
            message.setSubject(subject);
            message.setText(body);
            Transport transport = session.getTransport("smtp");
            transport.connect(host, from, pass);
            transport.sendMessage(message,
message.getAllRecipients());
            transport.close();}
        catch (AddressException ae)
     {
            ae.printStackTrace();   }
        catch (MessagingException me)
          {
```


Библиотека БГУИР

```
        me.printStackTrace();
      }
    }
  }
```

Заметим, что данная программа предполагает выход в Интернет без `proxy`-сервера (при наличии `proxy`-сервера возможны отказы в работе). В программе требуется правильно указать отправителя, включая его `login` и `password`, получателя, а также имя почтового сервера. В нашем примере имеем:

```
String from_str = user_log;
String pass = PASSWD;
String[] to_str_arr = { rcpnt };
```

Последняя строка задает список получателей (в примере список `to_str_arr` содержит единственное имя). Указание почтового сервера и задание его свойств выполняется в методе `sendFromMailRU`:

```
Properties props = System.getProperties();
String host = "smtp.mail.ru";
props.put("mail.smtp.starttls.enable", "true");
props.put("mail.smtp.host", host);
props.put("mail.smtp.user", from);
props.put("mail.smtp.password", pass);
props.put("mail.smtp.port", "587");
props.put("mail.smtp.auth", "true");
```

Отметим, что почтовый `host` – это компьютер, на котором расположен почтовый сервер. В нашем примере это `String host = "smtp.mail.ru"`. Отправка сообщения выполняется в строке

```
transport.sendMessage(message, message.getAllRecipients());
```

Команда

```
Transport transport = session.getTransport("smtp");
```

указывает тип используемого транспортного протокола. Собственно текст почтового сообщения помешается в `message.setText(body)`.

Для присоединения к почтовому отправлению файла программу несколько изменим:

```
package main;
import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;
import java.io.*;
import javax.activation.DataHandler;
```


Библиотека БГУИР

```
import javax.activation.DataSource;
import javax.activation.FileDataSource;
import javax.mail.BodyPart;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Multipart;

public class Main {

    private static String user_log = "ovgerman@mail.ru";
// Mail.RU user name)
    private static String PASSWD = "abracadabra";
// Mail password
    private static String rcpnt = "ovgerman@tut.by";

    public static void main(String[] args) {
        String from_str = user_log;
        String pass = PASSWD;
        String[] to_str_arr = { rcpnt };
   // list of recievers
        String subject_str = "Java send e-mail example";
        String body_str = "See attached file!";

        sendFromMailRU    (from_str,    pass_str,    to_str_arr,
subject_str, body_str);
    }

    private static void sendFromMailRU(String from, String pass,
String[] to, String subject, String body) {
        Properties props = System.getProperties();
        String host = "smtp.mail.ru";
        props.put("mail.smtp.starttls.enable", "true");
        props.put("mail.smtp.host", host);
        props.put("mail.smtp.user", from);
        props.put("mail.smtp.password", pass);
        props.put("mail.smtp.port", "587");
        props.put("mail.smtp.auth", "true");

        Session session = Session.getDefaultInstance(props);
        MimeMessage message = new MimeMessage(session);

        try {
            message.setFrom(new InternetAddress(from));
            InternetAddress[] toAddress =
             new InternetAddress[to.length];

            // To get the array of addresses
            for( int i = 0; i < to.length; i++ ) {
                toAddress[i] = new InternetAddress(to[i]);
            }

            for( int i = 0; i < toAddress.length; i++) {
```


Библиотека БГУИР

```
       message.addRecipient(Message.RecipientType.TO,
       toAddress[i]);
             }

             message.setSubject(subject);
             message.setText(body);

      MimeBodyPart messageBodyPart = new MimeBodyPart();
      String filename = "e:/work5/Main/file.txt";
      DataSource source = new FileDataSource(filename);
      messageBodyPart.setDataHandler(new DataHandler(source));
      messageBodyPart.setFileName(filename);
       // Create a multipart message
          Multipart multipart = new MimeMultipart();
          multipart.addBodyPart(messageBodyPart);
       //Next set the multipart in the message as follows:

          message.setContent(multipart);

             //
             Transport transport = session.getTransport("smtp");
             transport.connect(host, from, pass);
             transport.sendMessage(message,
message.getAllRecipients());
             transport.close();}
         catch (AddressException ae) {
             ae.printStackTrace();   }
         catch (MessagingException me) {
             me.printStackTrace();}
   }
}
```

Почтовое отправление нужно объявить как состоящее из нескольких частей (parts). Создаем новую часть так:

```
Multipart multipart = new MimeMultipart();
```

Присоединяем к этой части содержимое

```
multipart.addBodyPart(messageBodyPart);
```

Предварительно в качестве содержимого задаем файл

```
messageBodyPart.setFileName(filename);
```

На скриншоте (рисунок 59) показано окно с отображением содержимого почтового отправления, пришедшего к адресату (почтовый ящик на `mail.ru`).

Библиотека БГУИР

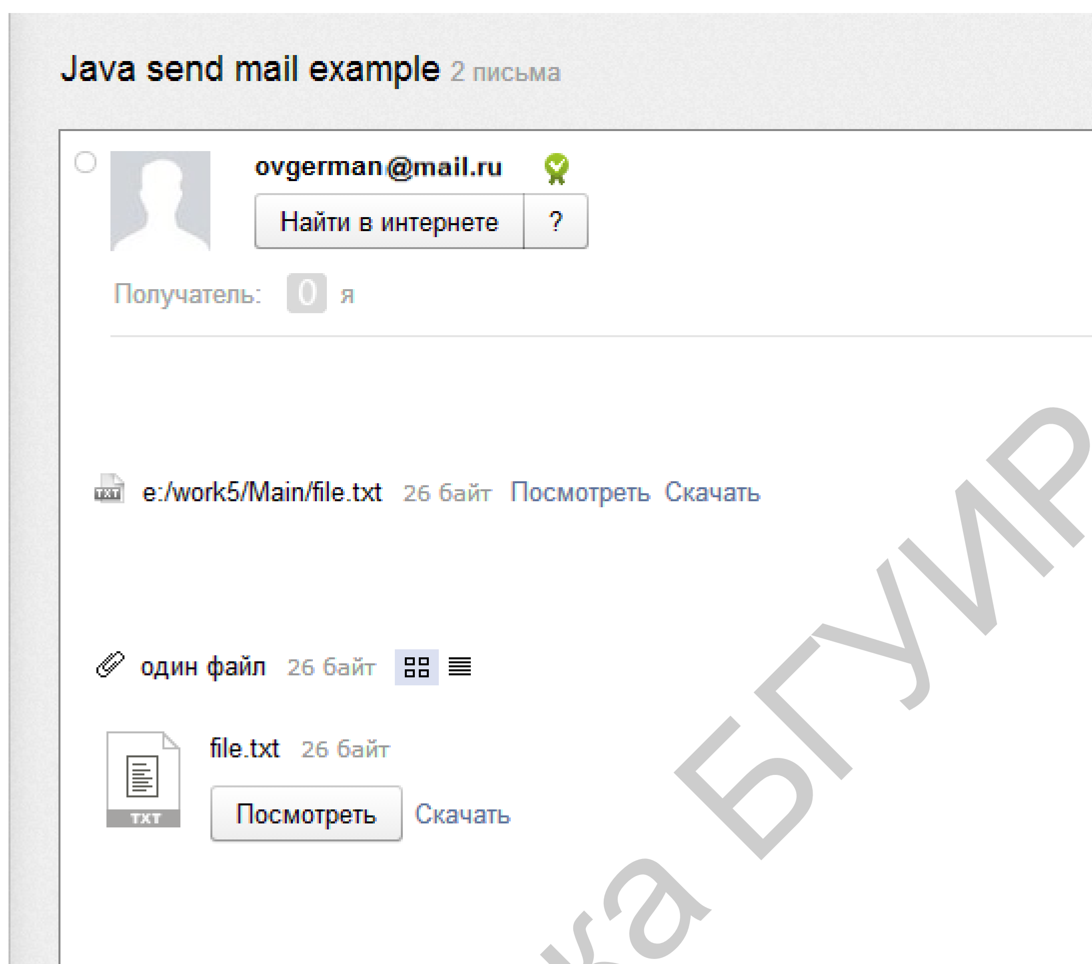


Рисунок 59 – Письмо, отправленное из программы и помещенное в почтовый ящик

2.7 Сериализация объектов и передача их по сети

Практический интерес представляет передача по сети сериализованных объектов. Сериализация означает сохранение объекта в файле на диске. Обратный процесс считывания объекта из файла называется десериализацией. Следует заметить, что сериализация является важнейшим механизмом в системе распределенного программирования, поскольку позволяет организовать хранилище объектов и возможность удаленного доступа к ним (что используется в системе `JNDI` – хранилище объектов `Java`).

Пусть мы хотим реализовать поиск записей по признаку (например, идентификационному номеру персоны). Наша цель построить клиент-серверное приложение. На стороне сервера должны быть предоставлены средства для изменения таблицы (добавление/удаление/поиск записей).

Клиент может считывать сериализованную таблицу и выполнять только поиск. Работающее приложение (запущены и сервер, и клиент) будет иметь следующий вид (рисунок 60).

Библиотека БГУИР

Серверная часть
Номер(идентификатор)
Фамилия
Группа
Даигностика
Сервер стартовал!!!
Очистить | Добавить | Удалить | Найти в базе | Сохранить базу | Загрузить базу | Включить сервер

Клиентская часть
Номер(идентификатор)
3
Фамилия
Shurik
Группа
3
Даигностика
Загружено из сервера!!!
Очистить | Найти в базе | Получить из сервера

Рисунок 60 – Окно сервера и клиента

В серверной части приложения у нас будет три класса: первый класс `Student` «поставляет» экземпляры для сохранения в таблице; второй класс `servThr` представляет поток, выполняющий связь с клиентом и отправку ему массива байтов с сериализованной таблицей; третий – главный класс серверного приложения, реализующий его функционал. Структура планируемого проекта показана на рисунке 61.

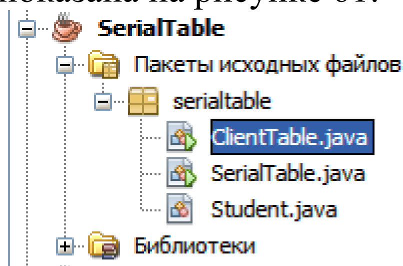


Рисунок 61 – Структура проекта

Библиотека БГУИР

Теперь создадим задуманный проект типа `Приложение Java`, дадим ему имя `SerialTable`. Создадим в проекте следующий сериализуемый класс:

```
package serialtable;
import java.io.Serializable;
public class Student implements Serializable
{ public String id;
  public String lname;
  public String group;

  public Student(String id, String lname,
          String group)

  { this.id=id;
    this.lname=lname;
    this.group=group;}
  }
```

Класс `Student` содержит три переменных-члена:

```
public String id;
public String lname;
public String group;
```

Первоначально мы создадим базу из объектов этого класса и сохраним ее в файле так, чтобы можно было позднее загрузить эту базу и выполнить поиск студента. Главный класс будет основан на форме (`Frame`) (см. приложение А). Форма серверной части показана выше на скриншоте (рисунок 60).

Таблицу с записями класса `Student` реализуем на основе ассоциативного класса `Hashtable`, который позволяет выполнять ассоциативный поиск. Добавление записи в ассоциативную таблицу реализуется так:

```
    Student st=new Student(id,fnamefld.getText(),
                        lnamefld.getText(),
                        groupfld.getText());
     bd.put(id,st);
```

Занесение записи в таблицу реализует команда `put`, в которой два аргумента: идентификационный номер (ассоциативный признак) и объект типа `Student`.

Поиск записи выполняет следующий фрагмент кода:

```
 String id=idfld.getText();
  Object ob=bd.get(id);
   if (ob!=null)
```


Библиотека БГУИР

```
{ lnamefld.setText(((Student)ob).lname);
  groupfld.setText(((Student)ob).group);
  idfld.setText(id);
}
```

Запись отыскивается по команде

```
Object ob=bd.get(id);
```

Если запись существует в ассоциативной таблице, то фамилию студента и номер группы выводим в текстовые поля. Таблица – это объект; сохранение объектов в файле называется, как указывалось ранее, сериализацией, а восстановление объектов из файлов – десериализацией. Сохраняем объект-таблицу таким образом:

```
try{
 fos=new FileOutputStream("tmpserial.dat");
 oos=new ObjectOutputStream(fos);
 oos.writeObject(bd); //write in file the table
 fos.close();
 oos.close();
 infofld.setText("Сохранено!!!");}
catch(Exception ex)
   {infofld.setText("Не может выполнить сохранение:"+
    ex.getMessage());}
```

Для ввода-вывода объектов используют классы `ObjectOutputStream`, `ObjectInputStream`. Занесение таблицы, представленной объектом `bd`, в файл реализует оператор `oos.writeObject(bd)`. Чтение (десериализация) таблицы выполняется таким образом:

```
fis=new FileInputStream(new File("tmpserial.dat"));
ois=new ObjectInputStream(fis);
bd=(Hashtable) ois.readObject(); //reading table
fis.close();
ois.close();
infofld.setText("Загружено!!!");
```

Обратим внимание на необходимость приведения типов при десериализации:

```
bd=(Hashtable) ois.readObject();
```

Заметим, кроме того, что сериализация/десериализация требует подключить библиотечные пакеты

```
import java.io.*;
import java.io.Serializable;
```

Сериализуемый класс должен быть объявлен с интерфейсом `Serializable`:

Библиотека БГУИР

```
class Student implements Serializable.
```

Теперь рассмотрим класс потока, выполняющего связь с клиентом и отправку ему сериализованной таблицы. Этот класс имеет следующий вид:

```
class servThr extends Thread
{
     static ServerSocket sc;
         servThr()
           {
              try
                   {
             sc=new ServerSocket(3001); }
             catch(Exception er) {}
}
     public void run()
{
    try
    {
    RandomAccessFile f = new
    RandomAccessFile("tmpserial.dat", "r");
    byte[] b = new byte[(int)f.length()];
    f.read(b);
    f.close();

    while(true)
{
    Socket s=null;
      try{
             s=sc.accept();
             if( s!=null)
          {
             DataOutputStream dOut =
 new DataOutputStream(s.getOutputStream());
             dOut.writeInt(b.length);
             dOut.write(b);
             s.close();
          }}
      catch(Exception ex)
          {JOptionPane.showMessageDialog(null, ex.getMessage());
          }
        }
      }
     catch(Exception exx)
     {}
   }
}
```

Формирование массива байтов, содержащего сериализованную таблицу, выполняется в следующем фрагменте кода:

```
RandomAccessFile f = new RandomAccessFile("tmpserial.dat", "r");
```


Библиотека БГУИР

```
    byte[] b = new byte[(int)f.length()];
    f.read(b);
    f.close();
```

Отправка этого массива клиенту реализуется таким образом:

```
            DataOutputStream dOut =
 new DataOutputStream(s.getOutputStream());
            dOut.writeInt(b.length);
            dOut.write(b);
```

К данному моменту основной класс `SerialTable` приложения готов и имеет следующий вид:

```
package serialtable;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;
import java.io.Serializable;
import java.net.*;
import javax.swing.JOptionPane;

class servThr extends Thread
{
     static ServerSocket sc;
         servThr()
           {
              try
                   {sc=new ServerSocket(3001); }
             catch(Exception er) {}
}
     public void run()
{
    try
    {
    RandomAccessFile  f  =  new  RandomAccessFile("tmpserial.dat",
"r");
    byte[] b = new byte[(int)f.length()];
    f.read(b);
    f.close();
     while(true)
   {
    Socket s=null;
      try{
             s=sc.accept();
             if( s!=null)
          {
             DataOutputStream dOut =
 new DataOutputStream(s.getOutputStream());
```


Библиотека БГУИР

```
                dOut.writeInt(b.length);
                dOut.write(b);
                s.close();
             }
}
      catch(Exception ex)
          {JOptionPane.showMessageDialog(null, ex.getMessage());
          }
        }
      }

     catch(Exception exx)
     {}
   }
}

public class SerialTable extends Frame implements
ActionListener{

    static boolean serverIsrunning=false;
    Panel dat;
    Panel but;
    Hashtable bd;
    TextField idfld;
    TextField lnamefld;
    TextField groupfld;
    TextField infofld;
    Button clearbtn;
    Button addbtn;
    Button deletebtn;
    Button searchbtn;
    Button saveTab;
    Button loadTab;
    Button runserver;

    ObjectOutputStream oos;
    FileOutputStream fos;
    ObjectInputStream ois;
    FileInputStream fis;

     SerialTable()
   {
    super("Серверная часть");
    bd=new Hashtable();
    setLayout(new BorderLayout());
    dat=new Panel();
    dat.setLayout(new GridLayout(10,2));
    dat.add(new Label("Номер(идентификатор)"));
    dat.add(idfld=new TextField());
    dat.add(new Label("Фамилия"));
    dat.add(lnamefld=new TextField());
    dat.add(new Label("Группа"));
```


Библиотека БГУИР

```
 dat.add(groupfld=new TextField());
 dat.add(new Label("Даигностика"));
 dat.add(infofld=new TextField());
 but=new Panel();
 but.setLayout(new GridLayout(1,6));
 but.add(clearbtn=new Button("Очистить"));
 but.add(addbtn=new Button("Добавить"));
 but.add(deletebtn=new Button("Удалить"));
 but.add(searchbtn=new Button("Найти в базе"));
 but.add(saveTab=new Button("Сохранить базу"));
 but.add(loadTab=new Button("Загрузить базу"));
 but.add(runserver=new Button("Включить сервер"));

 add("Center",dat);
 add("South",but);
clearbtn.addActionListener(this);
addbtn.addActionListener(this);
deletebtn.addActionListener(this);
searchbtn.addActionListener(this);
saveTab.addActionListener(this);
loadTab.addActionListener(this);
runserver.addActionListener(this);

}

 public void clearFields()
{
  idfld.setText("");
  lnamefld.setText("");
  groupfld.setText("");
  infofld.setText("");
 }

public void actionPerformed(ActionEvent ae)
{
 Object source=ae.getSource();
 if (source==runserver)
 {
   if(!serverIsrunning)
   {
       serverIsrunning=true;
        infofld.setText("Сервер стартовал!!!");
        servThr sf=new servThr();
        sf.start();
   }
   else
   {
    infofld.setText("Сервер уже запущен!!!");
   }
 }
   else
 if (source==loadTab)
```


Библиотека БГУИР

```
    {
       try{

        fis=new FileInputStream(new File("tmpserial.dat"));
        ois=new ObjectInputStream(fis);
        bd=(Hashtable) ois.readObject();
        fis.close();
        ois.close();
        infofld.setText("Загружено!!!");
          }
       catch(Exception ex)
          {
       infofld.setText("Ошибка:"+ex.getMessage());
          }
    }
    else
    if (source==saveTab)
    {
       try{
        fos=new FileOutputStream("tmpserial.dat");
        oos=new ObjectOutputStream(fos);
        oos.writeObject(bd); //write in file the table
        fos.close();
        oos.close();
        infofld.setText("Сохранено!!!");
          }

       catch(Exception ex)
          {
infofld.setText("Не может сохранить таблицу :"+
ex.getMessage());
          }
    }
   else
  if (source==clearbtn)
    {clearFields();}
   else

   if(source==addbtn)
    {String id=idfld.getText();
  if(id!=null)
    {Student st=new Student(id,
                           lnamefld.getText(),
                           groupfld.getText());
     bd.put(id,st);
     infofld.setText("Запись добавлена!!!");

} else

{ infofld.setText("Требуется номер(ID)!!!");}
}
   else
```

Библиотека БГУИР

```
   if (source==deletebtn)
  {
   String id=idfld.getText();
  if(bd.remove(id)!=null)
  {
 clearFields();
  infofld.setText("Запись удалена !!!");

}

}
  else  if (source==searchbtn)
  {
  String id=idfld.getText();
  Object ob=bd.get(id);
   if (ob!=null)
 {
  lnamefld.setText(((Student)ob).lname);
  groupfld.setText(((Student)ob).group);
  idfld.setText(id);
}  else
{ infofld.setText("ID не найден");}
}
}

    public static void main(String[] args) {
           SerialTable ff=new SerialTable();
           ff.addWindowListener(new WindowAdapter(){
          public void windowClosing(WindowEvent evt) {
             System.exit(0);
       }});
           ff.resize(700,300);
           ff.setBackground(Color.PINK);
           ff.setVisible(true);
           ff.setLocationRelativeTo(null);
    }
}
```

Теперь реализуем клиентскую часть. Она в значительной степени дублирует серверную, но лишена ряда функциональных возможностей: можно только просматривать записи. Текст клиентского приложения такой:

```
package serialtable;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.*;
import java.util.Hashtable;
import java.io.Serializable;
```

Библиотека БГУИР

```
import java.net.*;

public class ClientTable extends Frame implements ActionListener,Serializable{
    static boolean serverIsrunning=false;
    Panel dat;
    Panel bute;
    Hashtable bd;
    TextField idfld;
    TextField lnamefld;
    TextField groupfld;
    TextField infofld;
    Button clearbtn;
    Button searchbtn;
    Button getserver;
    ObjectOutputStream oos;
    FileOutputStream fos;
    ObjectInputStream ois;
    FileInputStream fis;

     ClientTable()
  {
    super("Клиентская часть");
    bd=new Hashtable();
    setLayout(new BorderLayout());
    dat=new Panel();
    dat.setLayout(new GridLayout(10,2));
    dat.add(new Label("Номер(идентификатор)"));
    dat.add(idfld=new TextField());
    dat.add(new Label("Фамилия"));
    dat.add(lnamefld=new TextField());
    dat.add(new Label("Группа"));
    dat.add(groupfld=new TextField());
    dat.add(new Label("Даигностика"));
    dat.add(infofld=new TextField());
    bute=new Panel();
    bute.setLayout(new GridLayout(1,3));
    bute.add(clearbtn=new Button("Очистить"));
    bute.add(searchbtn=new Button("Найти в базе"));
    bute.add(getserver=new Button("Получить из сервера"));
    add("Center",dat);
    add("South",bute);
   clearbtn.addActionListener(this);
   searchbtn.addActionListener(this);
   getserver.addActionListener(this);

   }

    public void clearFields()
   {
     idfld.setText("");
     lnamefld.setText("");
```


Библиотека БГУИР

```
     groupfld.setText("");
     infofld.setText("");
    }

   public void actionPerformed(ActionEvent ae)
   {
       Object source=ae.getSource();

  if (source==getserver)
    {
      if(!serverIsrunning)
      {
          Socket socket=null;
          serverIsrunning=true;
          infofld.setText("Сервер подключен к клиенту!!!");
          try
              {
          InetAddress addr = InetAddress.getByName(null);
          socket= new Socket(addr,3001);
          DataInputStream dIn =
new DataInputStream(socket.getInputStream());

int length = dIn.readInt();// read length of incoming array
          if(length>0)
               {
          byte[] tabbuf = new byte[length];
          dIn.readFully(tabbuf, 0, tabbuf.length);
FileOutputStream fos =
new FileOutputStream("tmpserialclient.dat");
          fos.write(tabbuf);
          fos.close();
          //десериализуем принятую от сервера таблицу!!!
          fis=new FileInputStream(new
          File("tmpserialclient.dat"));
           ois=new ObjectInputStream(fis);
           bd=(Hashtable) ois.readObject();
           fis.close();
           ois.close();
           infofld.setText("Загружено из сервера!!!");
           socket.close();
               }
              }

          catch(Exception er)
          {
           infofld.setText("Ошибка :"+er.getMessage());
          }
          }
    else
      {
       infofld.setText("Сервер уже запущен!!!");
```

Библиотека БГУИР

```
      }
    }
    else

  if (source==clearbtn)
    {clearFields();}
   else
    if (source==searchbtn)
  {
  String id=idfld.getText();
  Object ob=bd.get(id);
   if (ob!=null)
{
  lnamefld.setText(((Student)ob).lname);
  groupfld.setText(((Student)ob).group);
  idfld.setText(id);

}  else
{ infofld.setText("ID не найден или таблица не подключена");}
}
}

    public static void main(String[] args) {
           ClientTable ff=new ClientTable();
           ff.addWindowListener(new WindowAdapter(){
          public void windowClosing(WindowEvent evt) {
             System.exit(0);
       }
         });
           ff.resize(700,300);
           ff.setBackground(Color.GREEN);
           ff.setVisible(true);
           ff.setLocationRelativeTo(null);
    }
}
```

Подключение и считывание таблицы из сервера выполняется таким образом:

```
          try
              {
          InetAddress addr = InetAddress.getByName(null);
          socket= new Socket(addr,3001);
          DataInputStream dIn =
 new DataInputStream(socket.getInputStream());

 int length = dIn.readInt();// read length of incoming array
          if(length>0)
               {
          byte[] tabbuf = new byte[length];
          dIn.readFully(tabbuf, 0, tabbuf.length);
               . . . . . . . . .
```


Библиотека БГУИР

```
        }

      catch(Exception er)
      {
       infofld.setText("Ошибка :"+er.getMessage());
      }
```

Принятый массив байтов записываем в файл:

```
FileOutputStream fos =
     new FileOutputStream("tmpserialclient.dat");
     fos.write(tabbuf);
     fos.close();
```

Затем выполняем десериализацию

```
fis=new FileInputStream(new
      File("tmpserialclient.dat"));
      ois=new ObjectInputStream(fis);
      bd=(Hashtable) ois.readObject();
      fis.close();
      ois.close();
```

Несмотря на достаточно внушительный размер описанного здесь проекта, его «внутренняя логика» вполне очевидна.

3 РАСПРЕДЕЛЕННОЕ ПРОГРАММИРОВАНИЕ НА ОСНОВЕ КОМПОНЕНТОВ

3.1 Понятие компонента. Примеры компонентов. Создание библиотечного компонента

Компонент представляет собой библиотечный модуль. В языке `Java` библиотечные модули упаковываются в архивные файлы (`jar`). Такой файл можно подключить к проекту и использовать содержащиеся в компоненте методы. Это легко пояснить примером. Создадим новый проект в `NetBeans` и выберем в качестве шаблона проекта библиотечный класс (рисунок 62).

Библиотека БГУИР

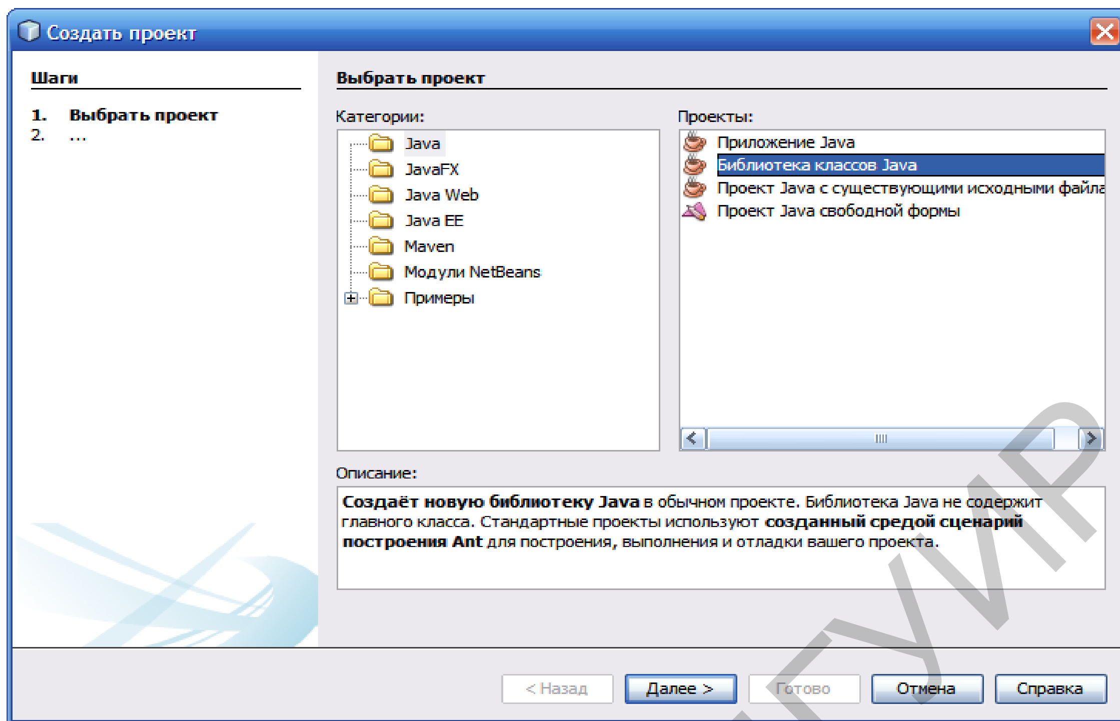


Рисунок 62 – Создание библиотечного класса

Дадим имя проекту `JavaLibA`. В окне проекта (рисунок 63) добавим класс в узле `Пакеты исходных файлов`.

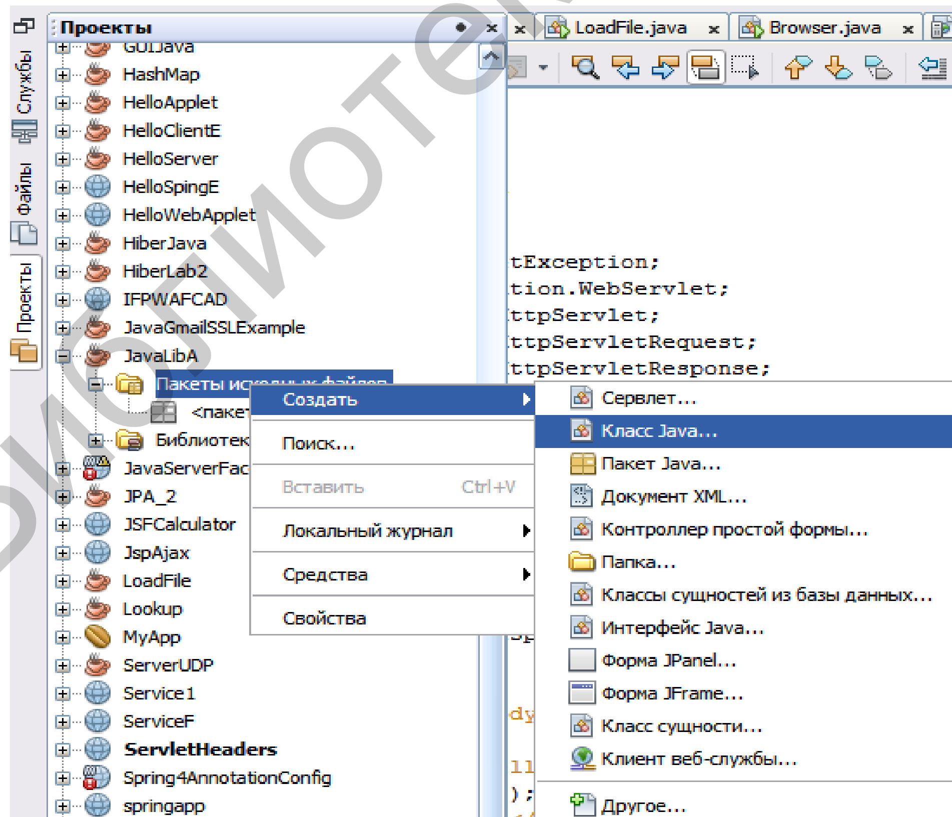


Рисунок 63 – Добавление класса

Библиотека БГУИР

В новом классе реализуем один-единственный метод

```
package com;

public class ClassLib {

    public static void sayHello(String name)
    {
      System.out.println("Hello, You old Buddy -"+name);

    }
  }
```

Чтобы построить jar-архив, достаточно выполнить опцию контекстного меню, активируемого на имени проекта – Очистить и Построить. Откроем вкладку Файлы и раскроем узел dist (там и будет находиться архивный файл, как показано на рисунке 64).

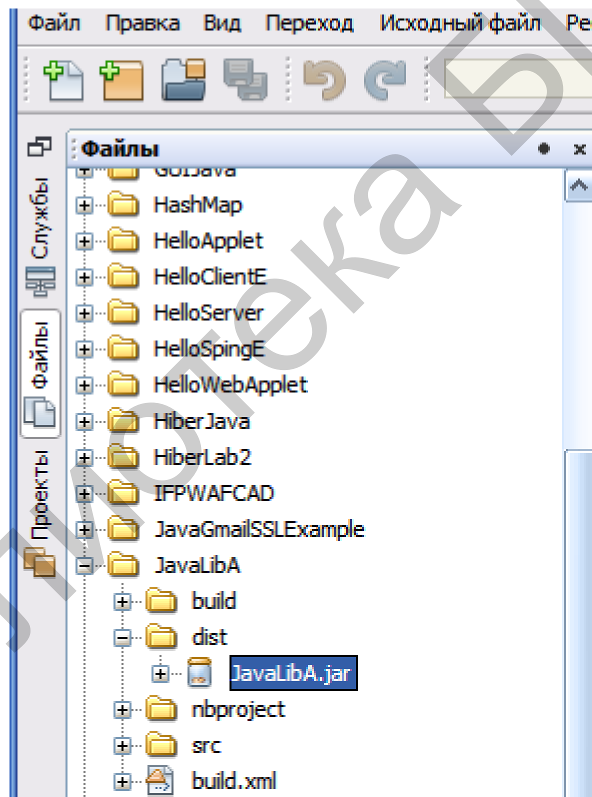


Рисунок 64 – Размещение архивного файла

Мы видим созданный архивный (библиотечный) файл. Этот файл можно подключить в любой другой проект. Создадим какой-нибудь новый проект на основе шаблона Java Application – testLib. Для добавления архивного файла в проект активизируем контекстное меню на узле Библиотеки и выберем пункт Добавить архив jar (рисунок 65).

Библиотека БГУИР

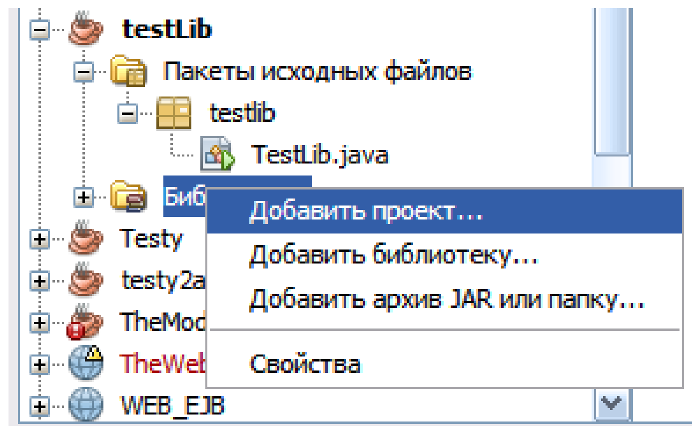


Рисунок 65 – Добавление архива в библиотеку

Отыскиваем наш архивный файл и добавляем его в новый проект (рисунок 66).

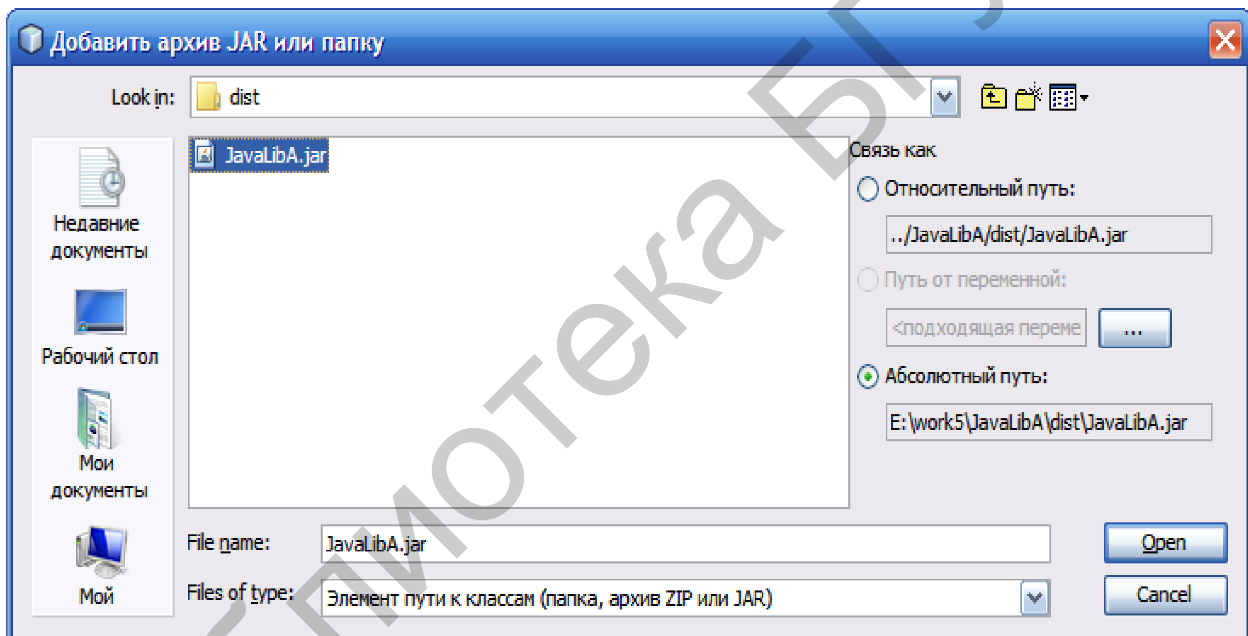


Рисунок 66 – Окно мастера для добавления архивного файла в библиотеку

Наберем следующий текст программы в окне редактора кода:

```
package testlib;
import com.ClassLib.*;

public class TestLib {
    public static void main(String[] args) {
        com.ClassLib.sayHello("New Programmer!");
     }
    }
```

Библиотека БГУИР

Библиотечный компонент подключен в строке

```
import com.ClassLib.*;
```

Результат работы программы представлен следующим выходом:

```
run:
Hello, You old Buddy - New Programmer!
ПОСТРОЕНИЕ УСПЕШНО ЗАВЕРШЕНО (общее время: 0 секунд)
```

Таким образом, работа с компонентами, включая их создание и добавление в другой проект, нами разъяснена.

Развитие компонентного программирования в Java является одной из фундаментальных особенностей этого языка. Пожалуй, в этом отношении Java не имеет себе равных. Мы перейдем к рассмотрению платформ, явным образом работающих с компонентами. Первой такой платформой являются web-сервисы.

3.2 Реализация web-сервисов

Приложение на основе web-сервиса состоит из двух частей (серверной части, представленной собственно web-сервисом, и клиентской части, представленной web-клиентом, который, используя ссылку на web-сервис, получает доступ к его методам). Как отмечалось ранее, web-сервис представляет собой бинарный класс, сохраненный в файле на стороне сервера. При вызове web-сервиса этот бинарный файл скачивается в оперативную память, где используется как обычный класс. Ссылки на объекты web-сервиса передаются по сети клиенту. Таким образом, клиент манипулирует ссылками – реальные вычисления производятся на стороне сервера. Взаимодействие между web-клиентом и web-сервисом выполняется по протоколу SOAP. Конфигурационный файл web-сервиса представляет собой документ XML, в котором содержится информация о методах и параметрах, передаваемых в методы данного web-сервиса.

Итак, начнем создавать серверную часть. Для этого последовательно проходим по скриншотам, иллюстрирующим основные шаги (рисунки 67, 68, 69).

Библиотека БГУИР

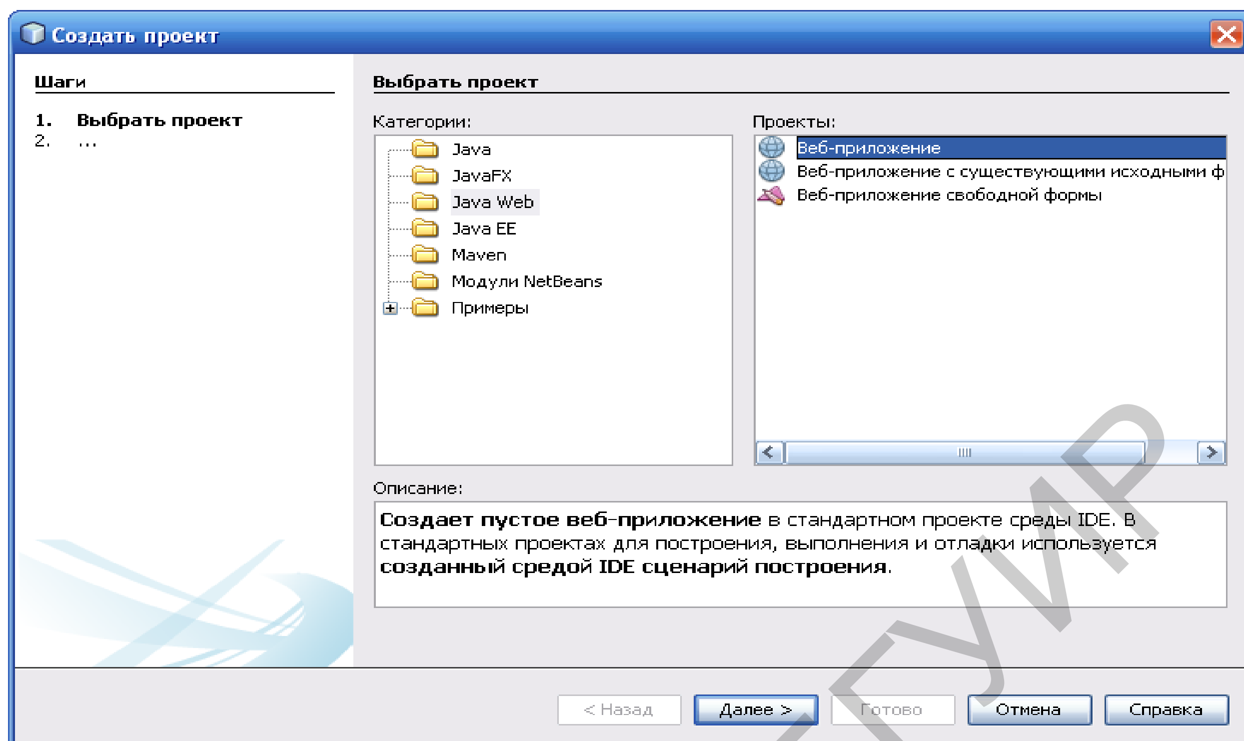


Рисунок 67 – Создание web-сервиса

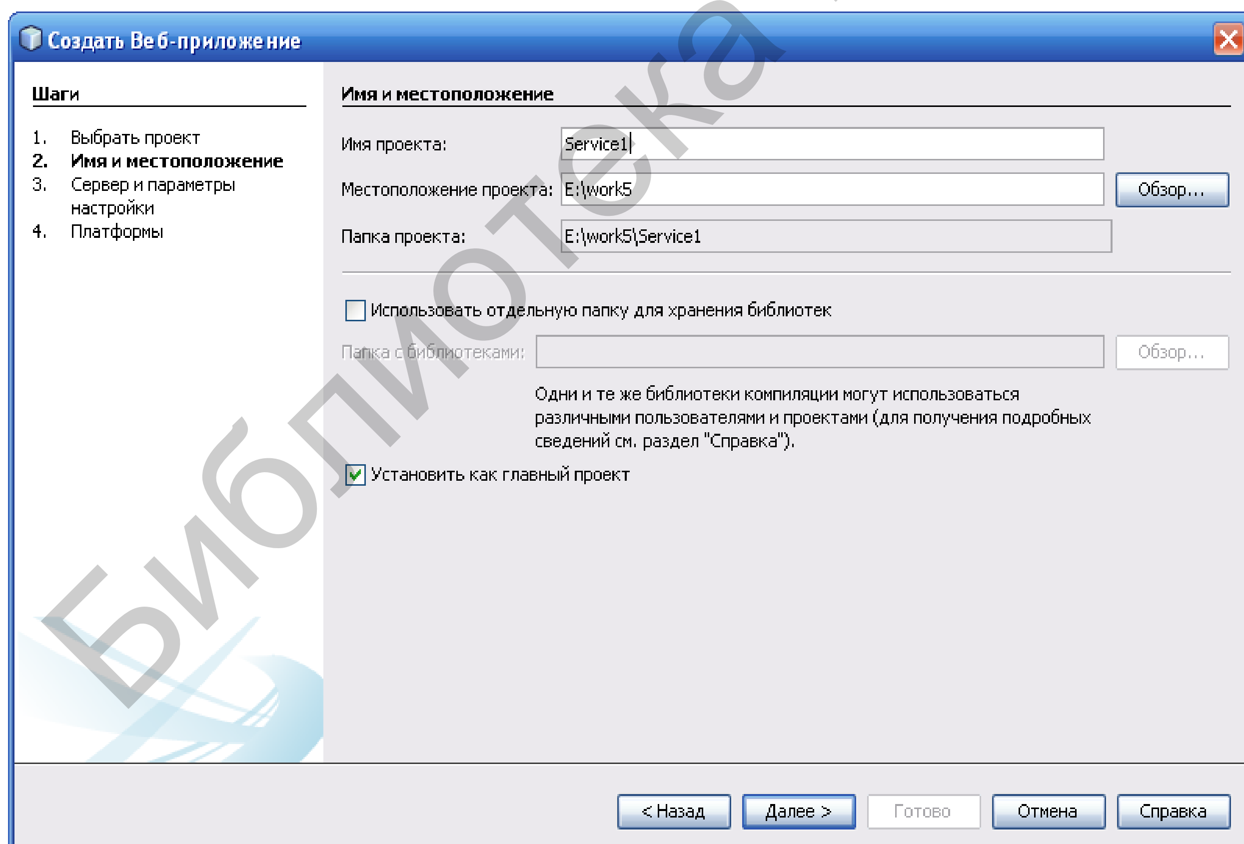


Рисунок 68 – Имя проекта, где создается web-сервис

Библиотека БГУИР

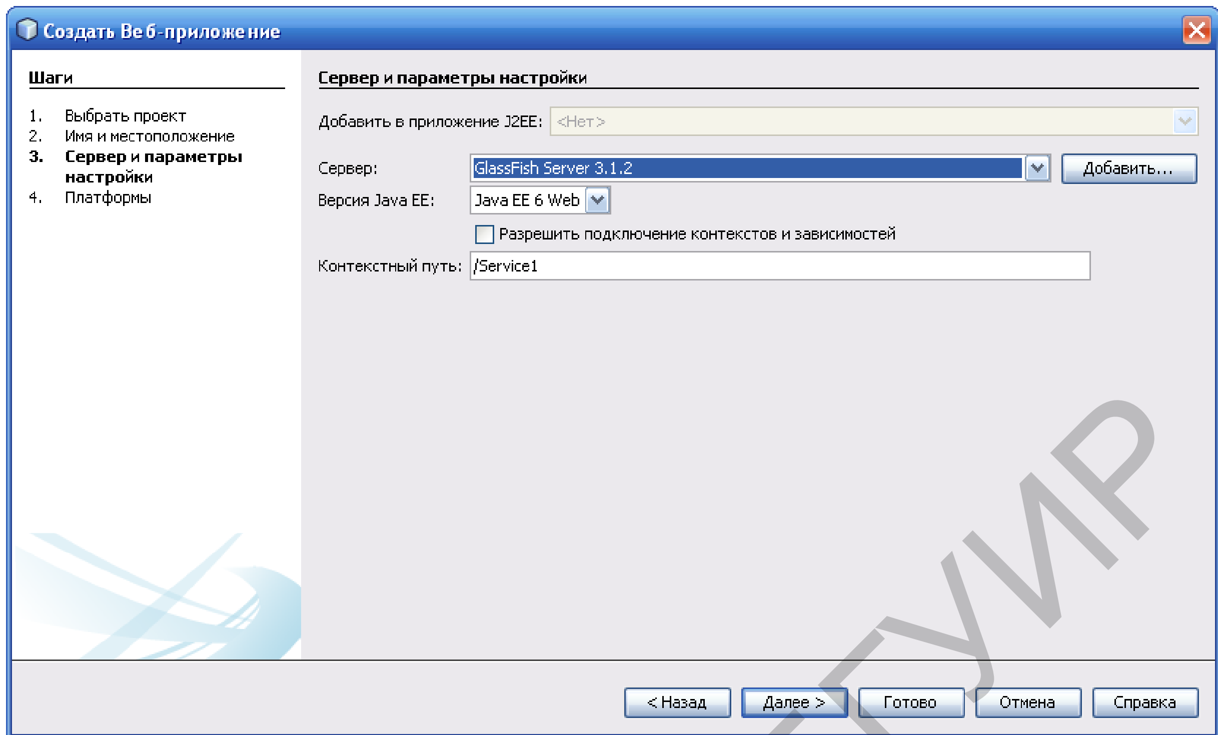


Рисунок 69 – Указание сервера, где развертывается web-сервис

Активизируем контекстное меню щелчком правой кнопки мыши и создаем web-службу (без привязки к WSDL – рисунок 70).

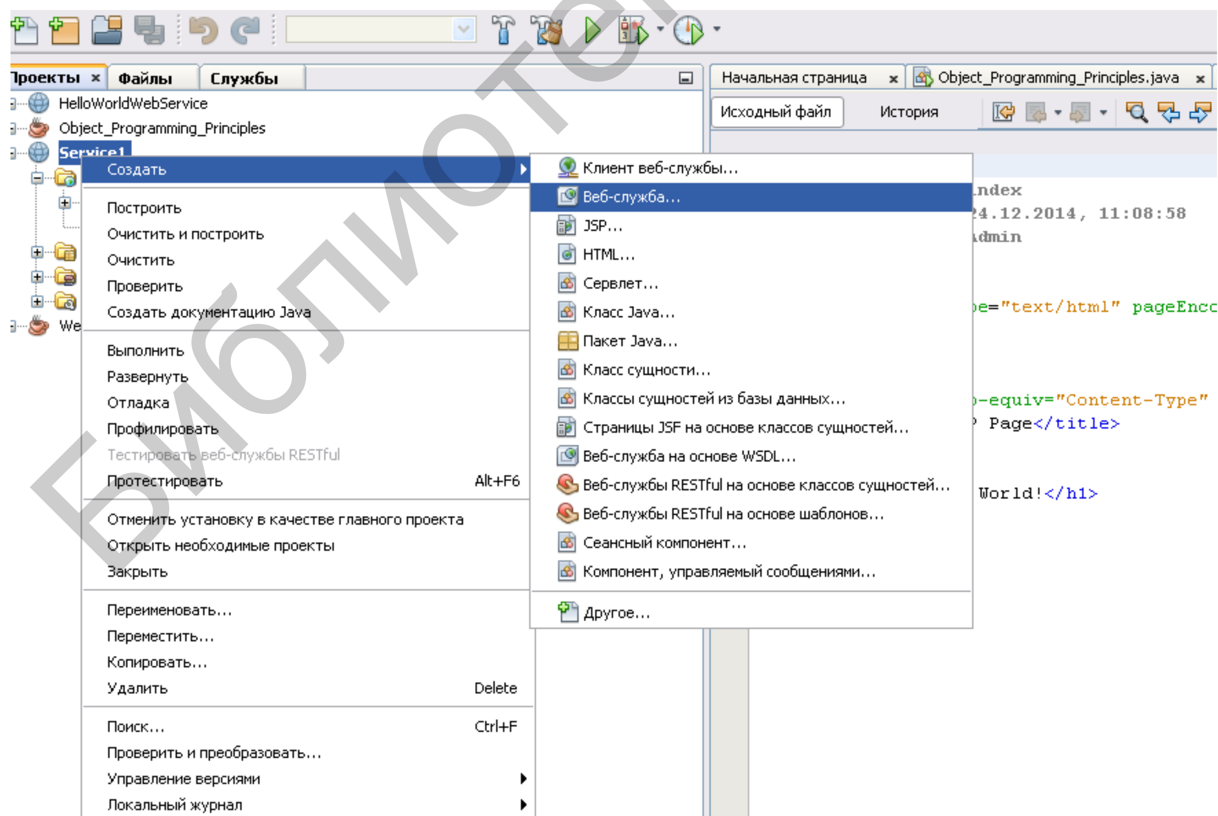


Рисунок 70 – Добавление web-службы

Библиотека БГУИР

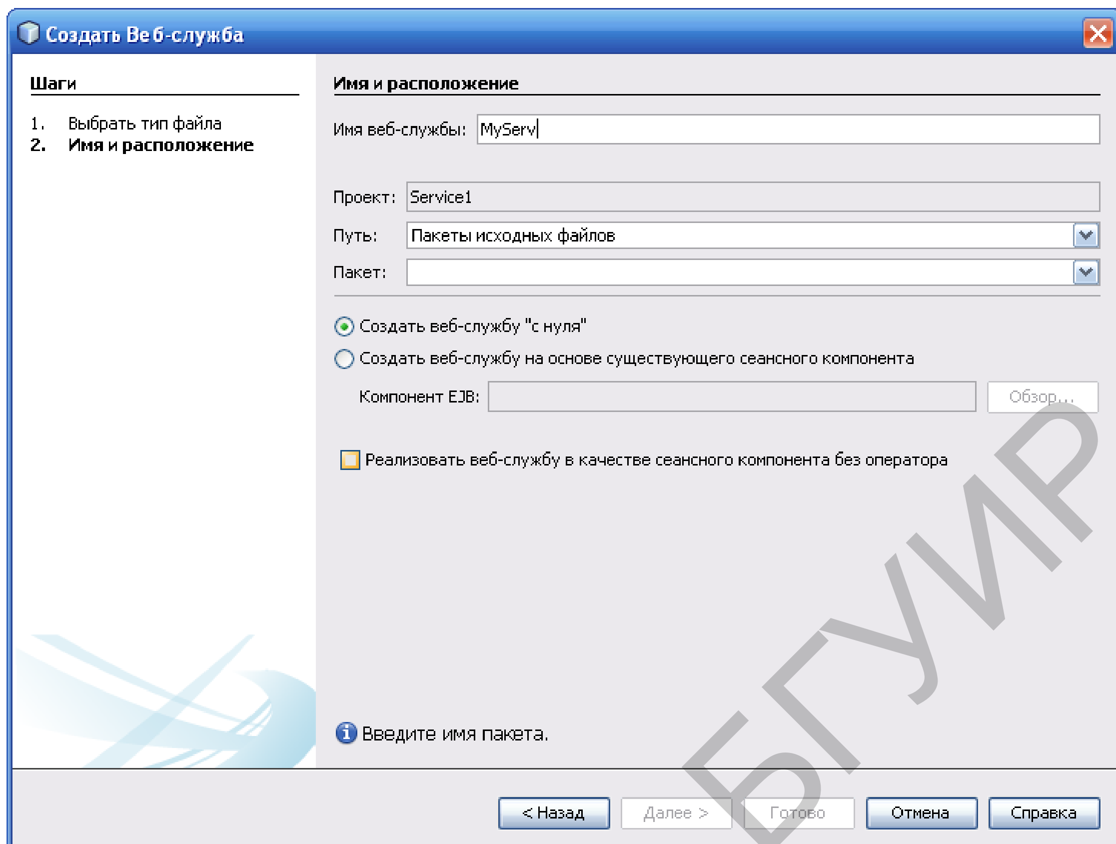


Рисунок 71 – Параметры web-службы

Открывается окно для редактирования кода, в которое внесем несколько правок.

```
package com;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;

@WebService(serviceName = "MyServ")
public class MyServ {

    /**
     * This is a sample web service operation
     */
    @WebMethod(operationName = "hello")
    public  String  sayGritings(@WebParam(name  =  "name")  String
txt) {
        return "Hello " + txt + " !";
    }
}
```

Оттенком выделено измененное название метода.

Библиотека БГУИР

Выполним сервис и разместим его (deploy), используя опции меню. Сначала из контекстного меню выберем Очистить и построить, затем – Развернуть. Затем – Тестировать. Окно тестирования показано на рисунке 72.

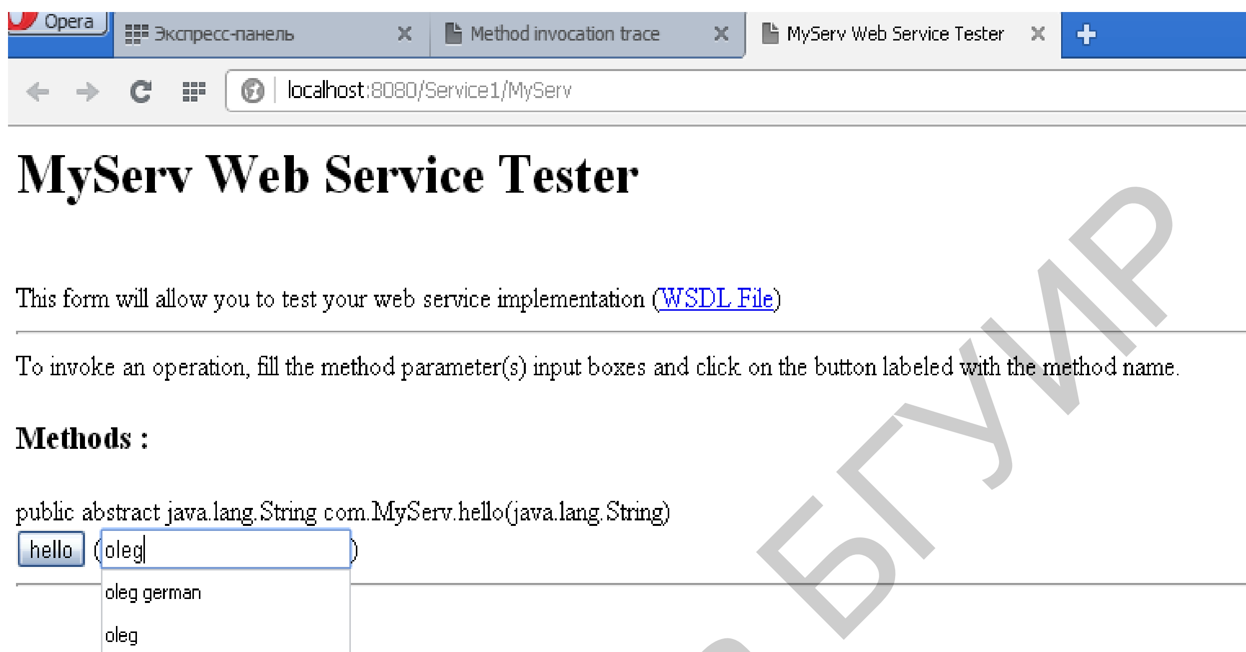


Рисунок 72 – Окно тестирования

Итак, серверная часть готова.

Создаем приложение-клиент как обычное Java-приложение (Java Application), а затем добавляем в него ссылку на созданный сервис. Создание клиента на основе Java Application тривиально – опускаем. Теперь вставляем в проект web-клиента (приводимая далее последовательность скриншотов на рисунках 73, 74 показывает, как это делается).

Библиотека БГУИР

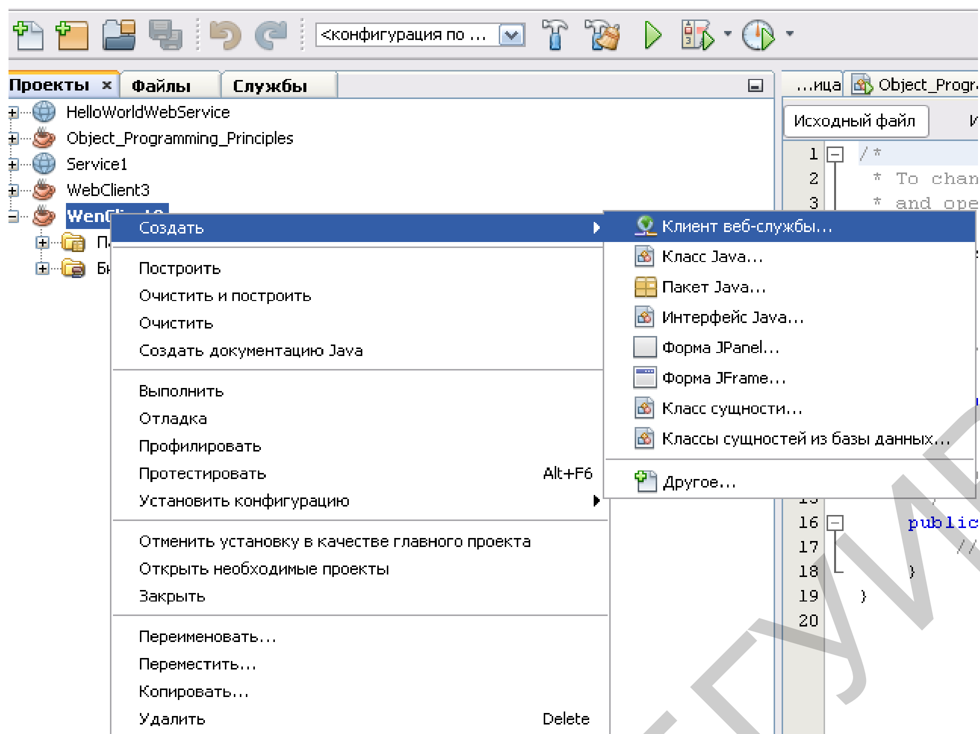


Рисунок 73 – Создание клиентской части

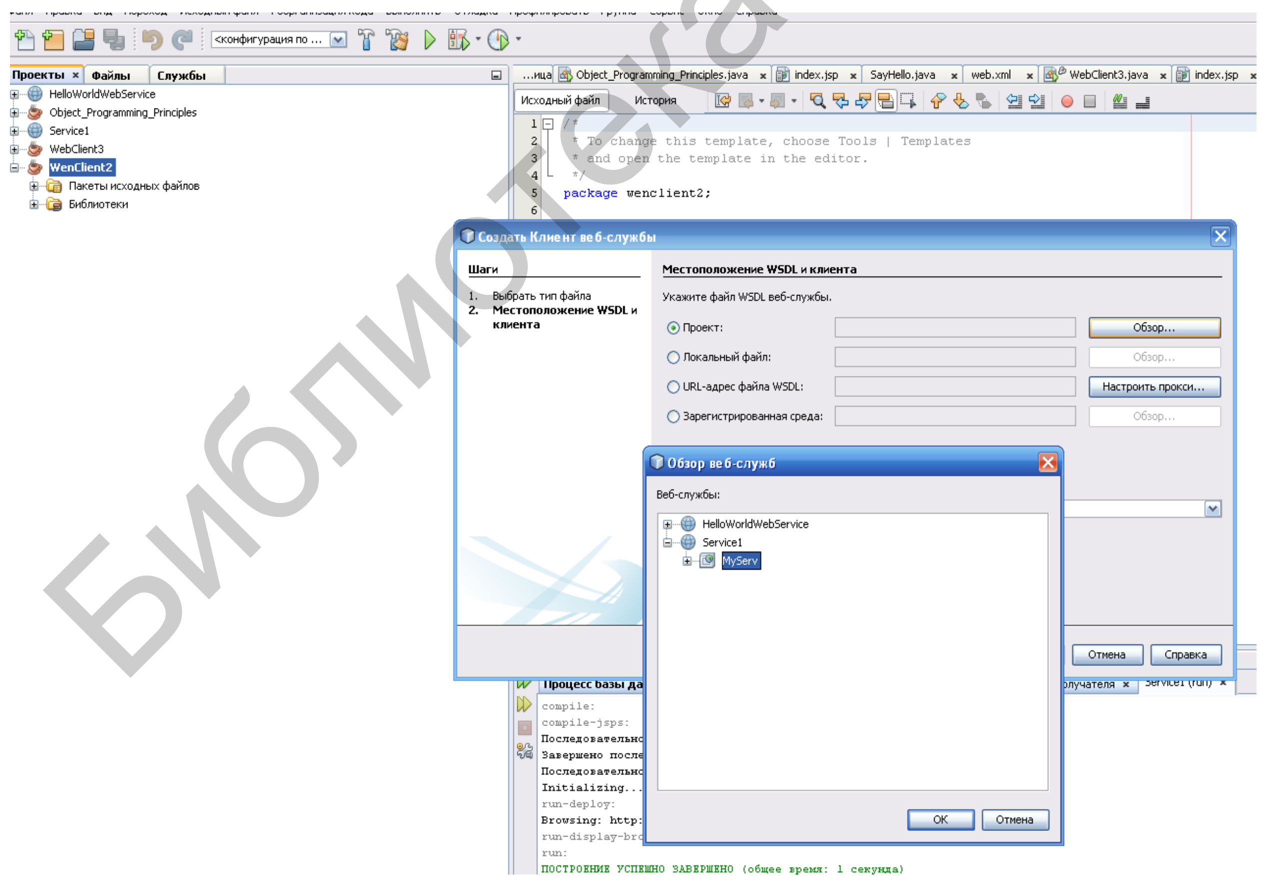


Рисунок 74 – Создание клиентской части (продолжение)

Библиотека БГУИР

Программируем клиентскую часть таким образом:

```
package wenclient2;
import com.MyServ;
import com.MyServ_Service;

public class WenClient2 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here

        MyServ_Service service = new MyServ_Service();
        System.out.println(service.getMyServPort().hello("OLEG
GERMAN"));

    }
}
```

Серверная часть подключается в строках импорта

```
import com.MyServ;
import com.MyServ_Service;
```

В программном коде клиентской стороны создаем объект, представляющий сервис, и вызываем его метод:

```
MyServ_Service service = new MyServ_Service();
        System.out.println(service.getMyServPort().hello("OLEG
GERMAN"));
```

Выходные данные на стороне клиента приведены ниже:

```
files are up to date
wsimport-client-generate:
Created dir: E:\work5\WenClient2\build\classes
Created dir: E:\work5\WenClient2\build\empty
Created   dir:   E:\work5\WenClient2\build\generated-sources\ap-
source-output
Compiling 7 source files to E:\work5\WenClient2\build\classes
Copying 3 files to E:\work5\WenClient2\build\classes
compile:
run:
Hello OLEG GERMAN !
ПОСТРОЕНИЕ УСПЕШНО ЗАВЕРШЕНО (общее время: 1 секунда)
```

Библиотека БГУИР

3.3 Компонентное программирование в Java EE

Компоненты, создаваемые на платформе Java EE, называются бобами (beans). Работа с компонентами и является основной характеристикой этой платформы (в стандартной платформе Java SDK работа с бобами не предусмотрена). Бобы могут быть локальными и удаленными.

Локальный боб находится в текущей папке проекта (т. е. является составной частью проекта). Добавлять ссылку на локальный боб в клиентской части не надо.

Удаленный боб следует подключать как архивный файл (jar-file).

Имеется три типа бобов – message driven, session и entity. Наиболее практически часто используемым является боб типа session. Мы рассмотрим бобы только этого типа. Бобы типа message driven используются для запуска путем отправки/получения сообщений; бобы категории entity служат для работы с объектными базами данных.

Различают два подтипа бобов категории session: sessionless и sessionfull. Последние отличаются от первых только тем, что позволяют возобновить сеанс (session) взаимодействия с бобом с точки прерывания. Первые всегда начинают сеанс сначала. Сеанс взаимодействия с бобом организует клиентское приложение, которое должно получить ссылку на компонент с тем, чтобы использовать его методы.

Каждый боб представлен двумя «составляющими»: интерфейсом, в котором объявлены методы боба, и собственно классом боба, где этот интерфейс реализован. Для создания боба нужно сначала создать пустой библиотечный класс, в который мы затем поместим интерфейс боба. Создание библиотечного класса нам уже знакомо. В NetBeans библиотечному классу соответствует одноименная категория проекта.

Так, создадим библиотечный класс с именем EJBInterfaceE (пока интерфейс пуст).

Следующим шагом является создание модуля (класса) боба. Для этого создаем новый проект и выбираем категорию проекта Java EE (Модуль EJB) (рисунок 75).

Введем имя проекта EJBAppE.

Библиотека БГУИР

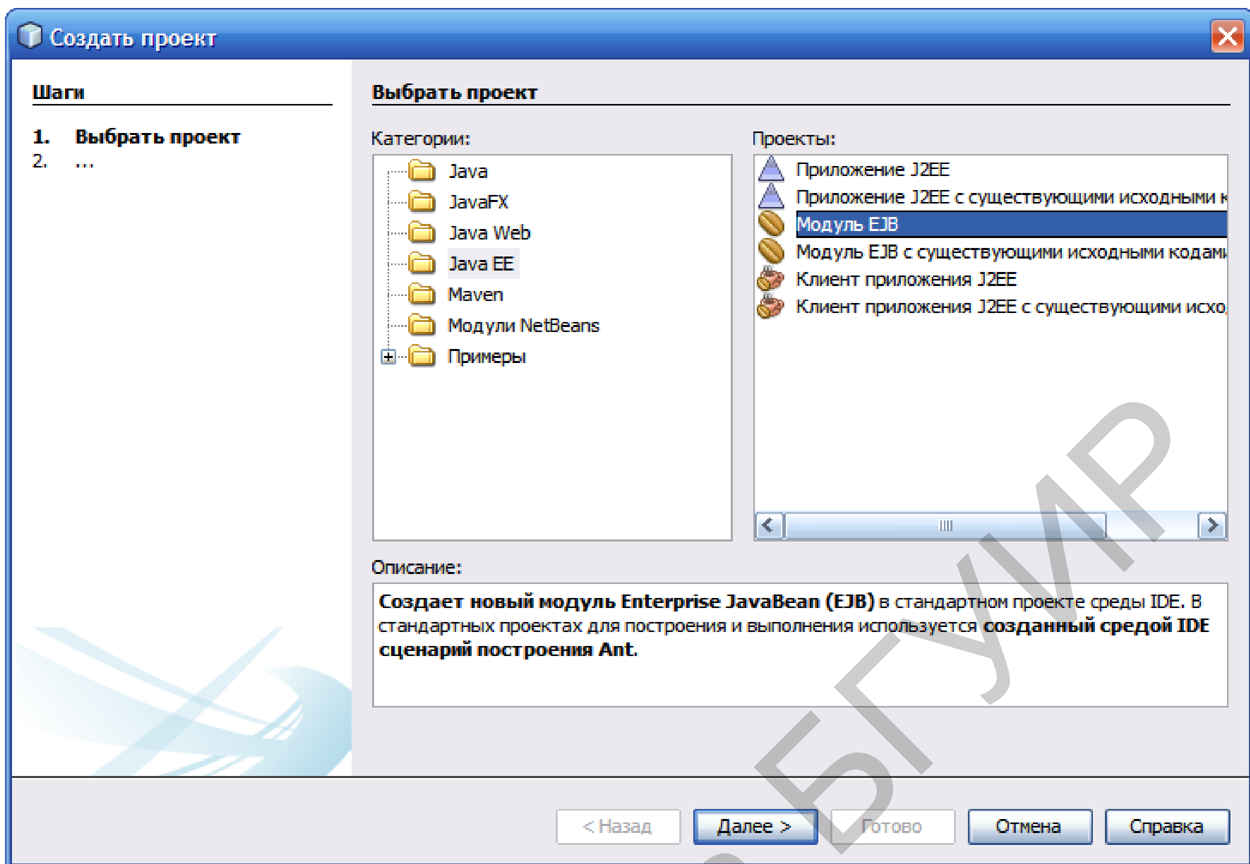


Рисунок 75 – Создание модуля боба

Создаем сеансовый компонент. Для этого активизируем контекстное меню на узле `Компоненты EJB` нашего проекта и выбираем пункт `Сеансовый компонент`. Откроется окно, показанное на рисунке 76. В нем следует задать следующие поля:

- имя `EJB` – `NewSessionBean` (разумеется, можно было выбрать другое имя);
- имя пакета – `com` (имя пакета произвольное, указывать имя пакета настоятельно рекомендуется);
- тип сеанса – без сохранения состояния (`Statelless`);
- тип интерфейса – удаленный интерфейс в проекте (в качестве имени проекта выбираем из списка ранее созданный пустой библиотечный класс).

Нажимаем `Готово`.

Библиотека БГУИР

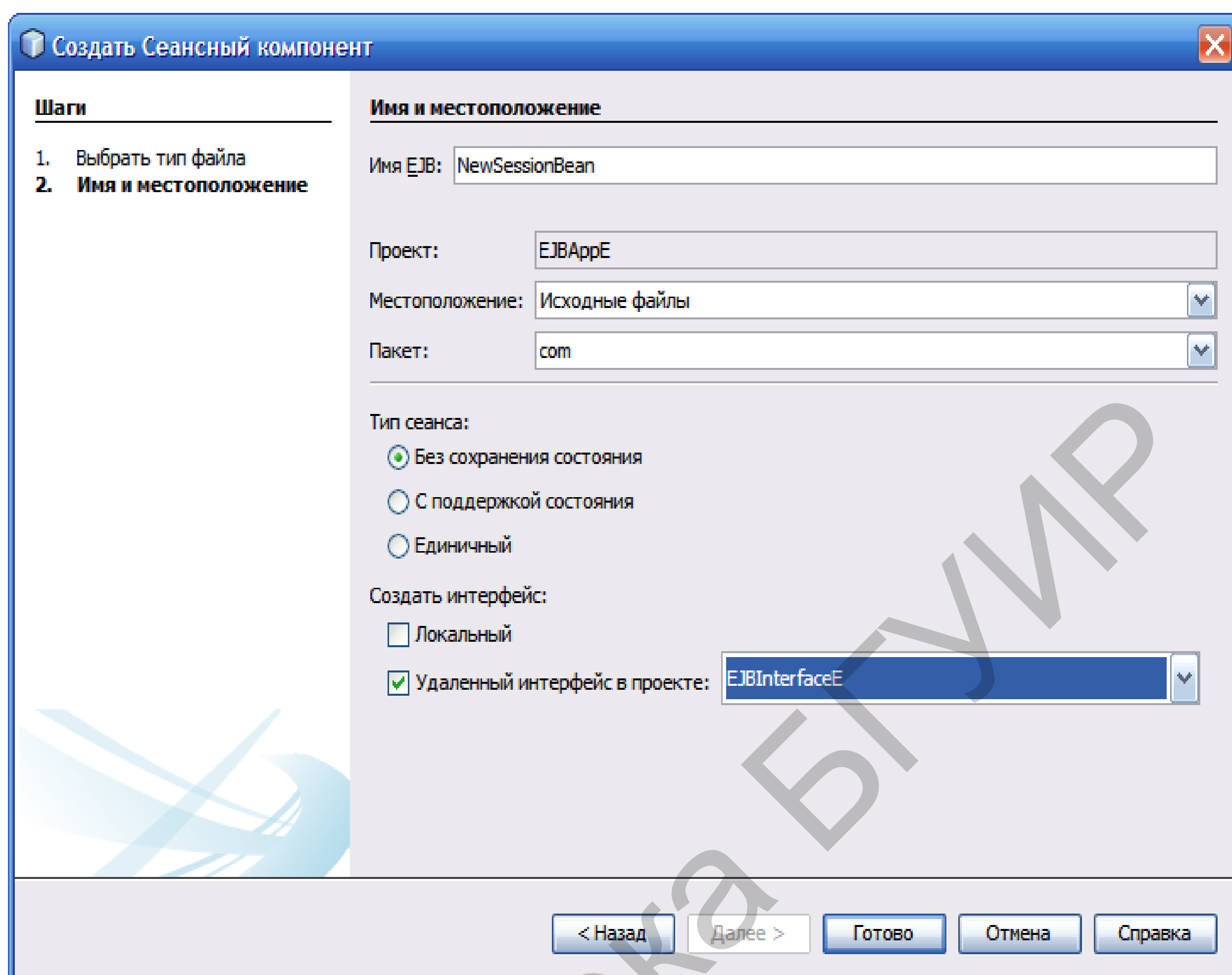


Рисунок 76 – Задание параметров боба

Создается следующая заготовка кода сеансового компонента:

```
package com;
import javax.ejb.Stateless;

@Stateless
public class NewSessionBean implements NewSessionBeanRemote
{
}
```

В настоящий момент мы имеем класс сеансового компонента и пустой интерфейс, помещенный в библиотечный архив. Нам нужно добавить методы в компонент – они называются бизнес-методами. Для добавления бизнес-метода активизируем контекстное меню щелчком правой кнопки мыши в окне редактора кода компонента (рисунок 77).

Библиотека БГУИР

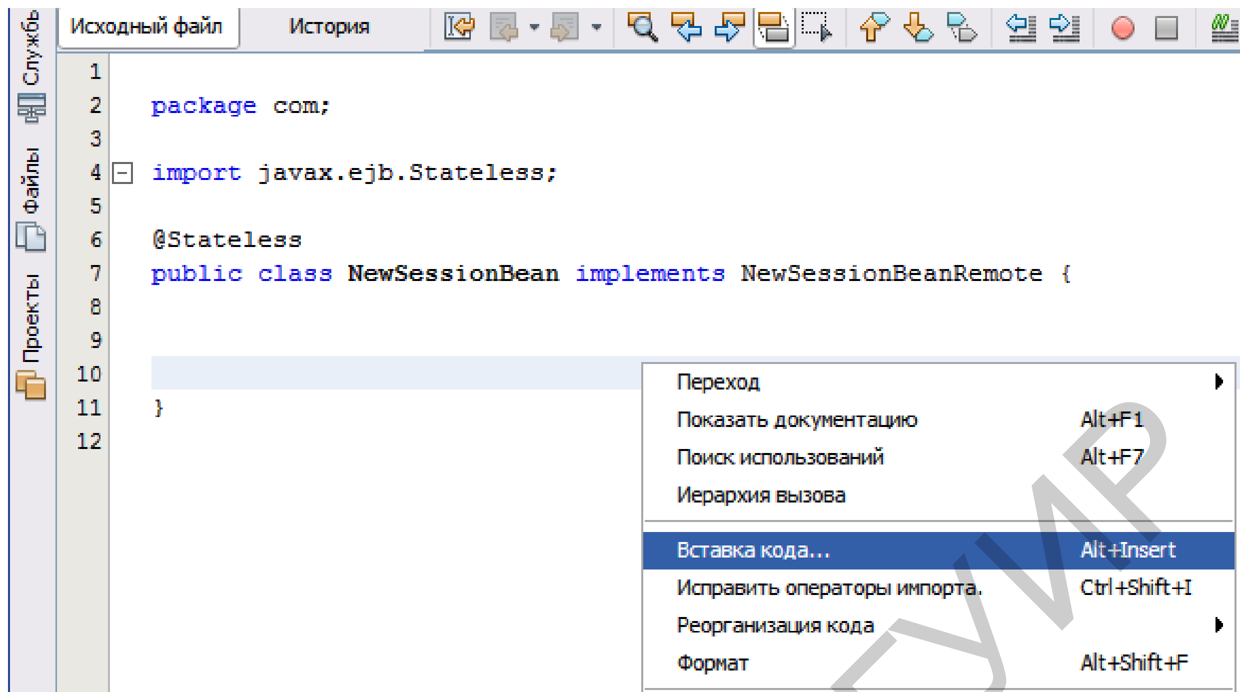


Рисунок 77 – Добавление бизнес-метода

Выбираем пункт `Вставка кода`. Затем выбираем пункт `Бизнес-метод`. Открывается следующее окно, в котором заполняем нужные поля: имя бизнес-метода, возвращаемый тип и параметры (рисунок 78).

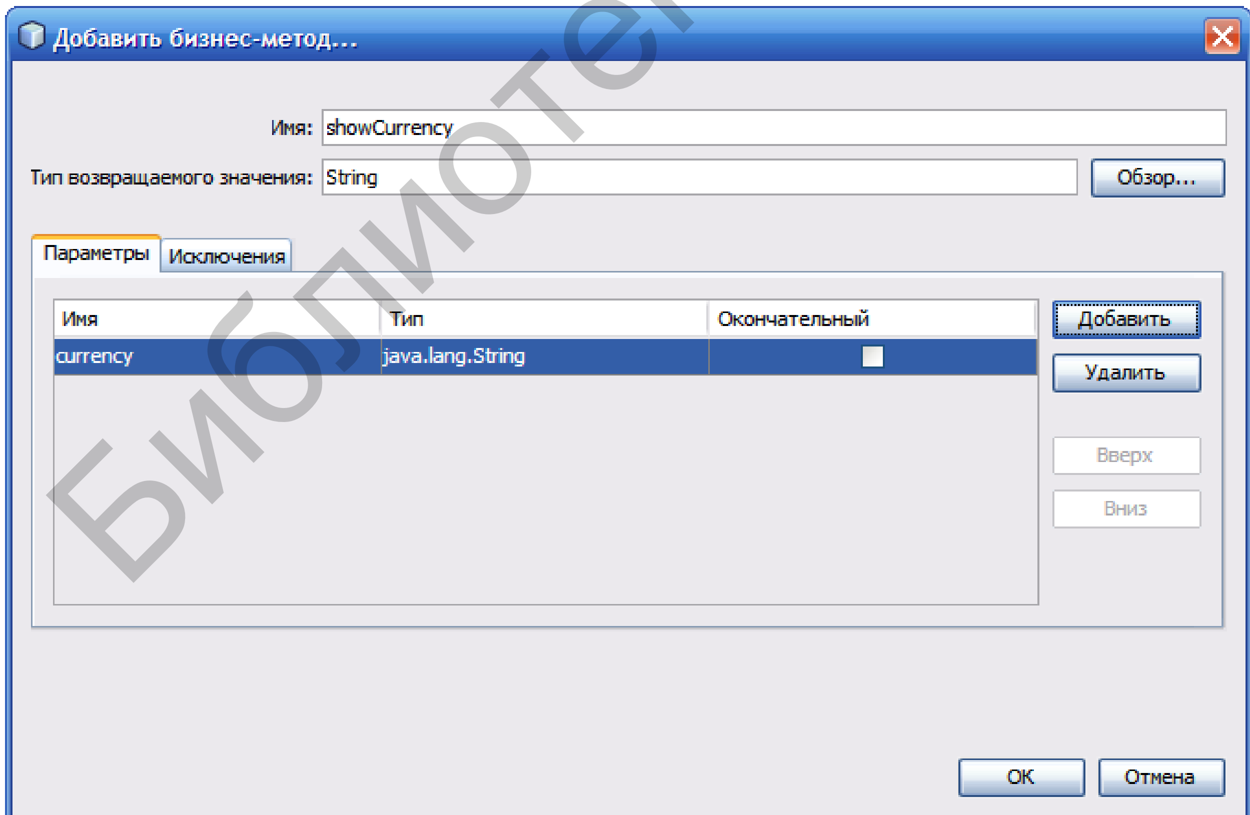


Рисунок 78 – Задание параметров бизнес-метода

Библиотека БГУИР

Нажимаем OK и дописываем код нашего метода:

```
package com;

import javax.ejb.Stateless;

@Stateless
public class NewSessionBean implements NewSessionBeanRemote {

    @Override
    public String showCurrency(String currency) {

        String answer="unknown";
        if(currency.indexOf("dollar")>=0)
        {
            answer="15900";

        }
        else
          if(currency.indexOf("euro")>=0)
        {
            answer="17200";

        }

        return answer;
    }
}
```

Заметим, что система автоматически добавила в интерфейс наш бизнес-метод. Интерфейс имеет такой вид:

```
package com;

import javax.ejb.Remote;
@Remote
public interface NewSessionBeanRemote {

    String showCurrency(String currency);}
```

Теперь у нас есть корпоративное приложение с простым компонентом EJB, предоставляемым через удаленный интерфейс. У нас также имеется независимая библиотека классов, содержащая интерфейс EJB, которую можно разослать другим разработчикам. Разработчики могут добавлять библиотеку к своим проектам, если им нужна связь с EJB, предоставляемым через удаленный интерфейс, но не нужны исходные коды для EJB. При изменении кода для EJB достаточно распространить JAR обновленной библиотеки классов, если изменения затронули любой из интерфейсов.

Библиотека БГУИР

При использовании диалогового окна Добавить бизнес-метод среда IDE автоматически включает метод в удаленный интерфейс.

Теперь корпоративное приложение можно собрать и запустить. При запуске приложения среда IDE развернет архив EAR на сервере.

Щелкните правой кнопкой мыши на корпоративном приложении EJBAppE и выберите Deploy (Развернуть).

После выбора Развернуть среда IDE собирает корпоративное приложение и разворачивает архив EAR на сервере. Если взглянуть в окно Files (Файлы), можно заметить, что файл JAR EJBRemoteInterface развернут вместе с приложением.

Если развернуть узел Приложения сервера GlassFish в окне Службы, можно увидеть, что компонент EJBAppE развернут (рисунок 79).

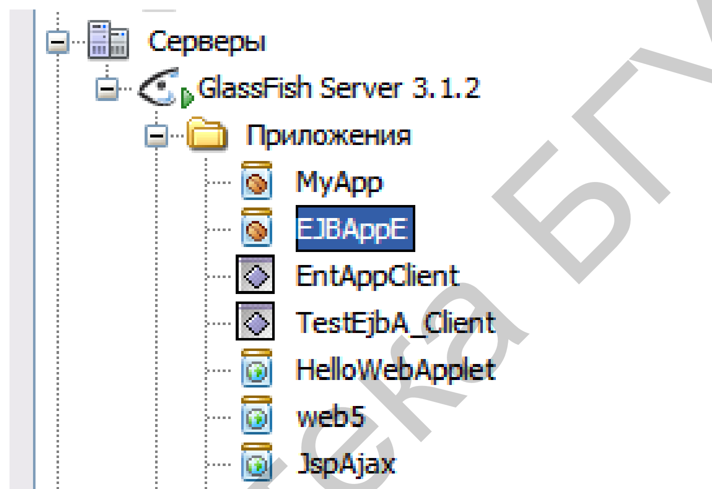


Рисунок 79 – Развернутые приложения на сервере

Итак, нам осталось создать клиентское приложение и связаться с сеансовым компонентом. Для этого необходимо выполнить следующее.

1 Выберите Файл > Создать проект и затем выберите Клиент корпоративного приложения в категории Java EE. Нажмите кнопку Далее.

2 Введите EJBClientE в поле Project Name (Имя проекта). Нажмите кнопку Далее.

3 Выберите GlassFish Server в качестве сервера. Нажмите кнопку Завершить.

Теперь следует добавить библиотеку классов, содержащую удаленный интерфейс. Выберите узел Библиотеки, откройте пункт Добавить проект в контекстном меню (рисунок 80).

Библиотека БГУИР

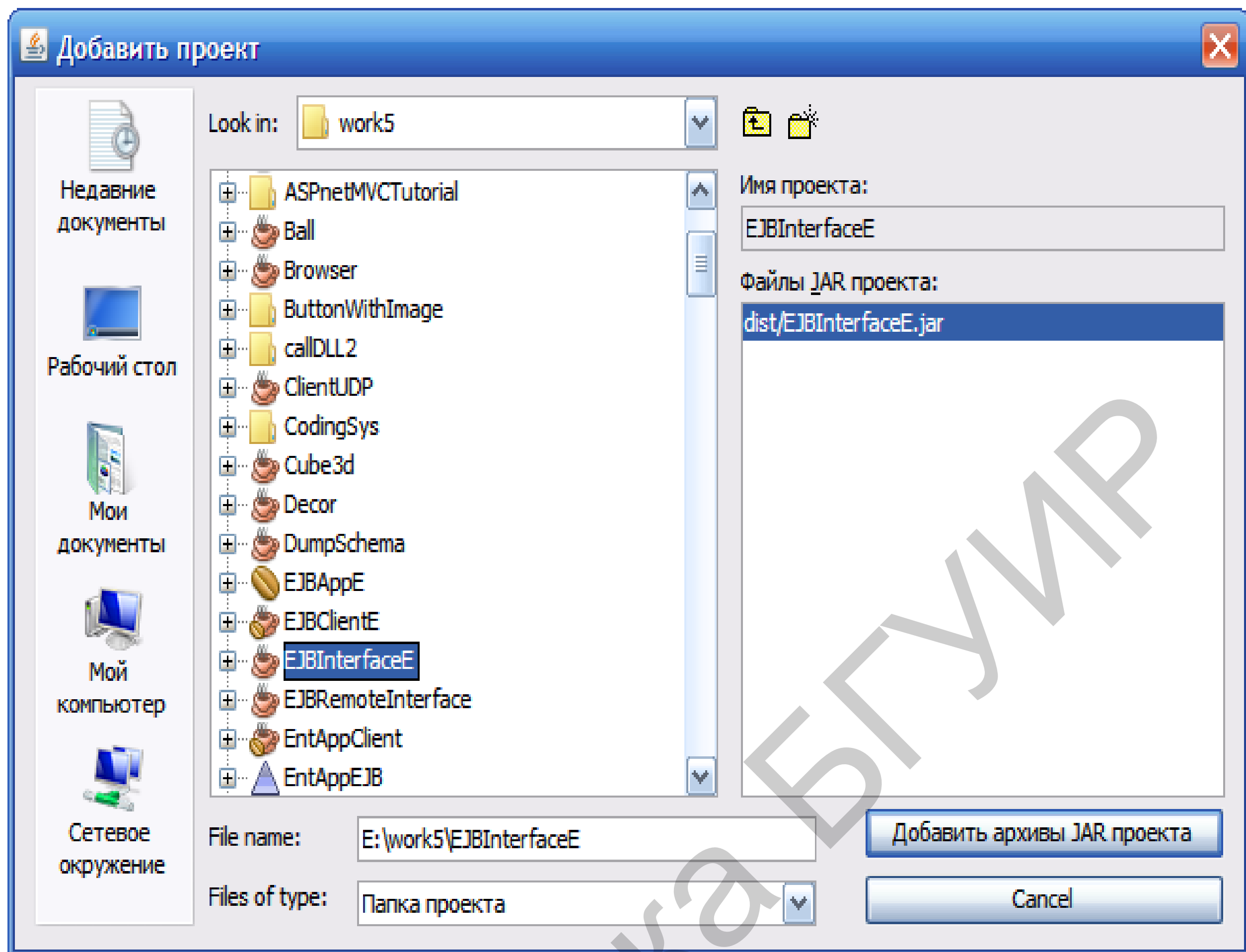


Рисунок 80 – Добавление проекта

Нажмите кнопку `Добавить архивы JAR проекта`.

Откройте исходный файл `Main.java` клиентского приложения в редакторе. Для этого дважды щелкните левой кнопкой мыши на узле дерева проекта с этим именем.

Добавьте в файл `Main.java` ссылку на сеансовый компонент следующим образом.

1 Щелкните правой кнопкой мыши исходный код и выберите `Insert Code` (`Вставить код`) (`Alt-Insert`), затем выберите `Call Enterprise Bean` (`Вызвать компонент корпоративного уровня`), чтобы открыть диалоговое окно вызова компонента корпоративного уровня.

2. Выберите узел проекта `EJBAppE`, затем `NewSessionBean`. Нажмите `OK` (рисунок 81).

Библиотека БГУИР

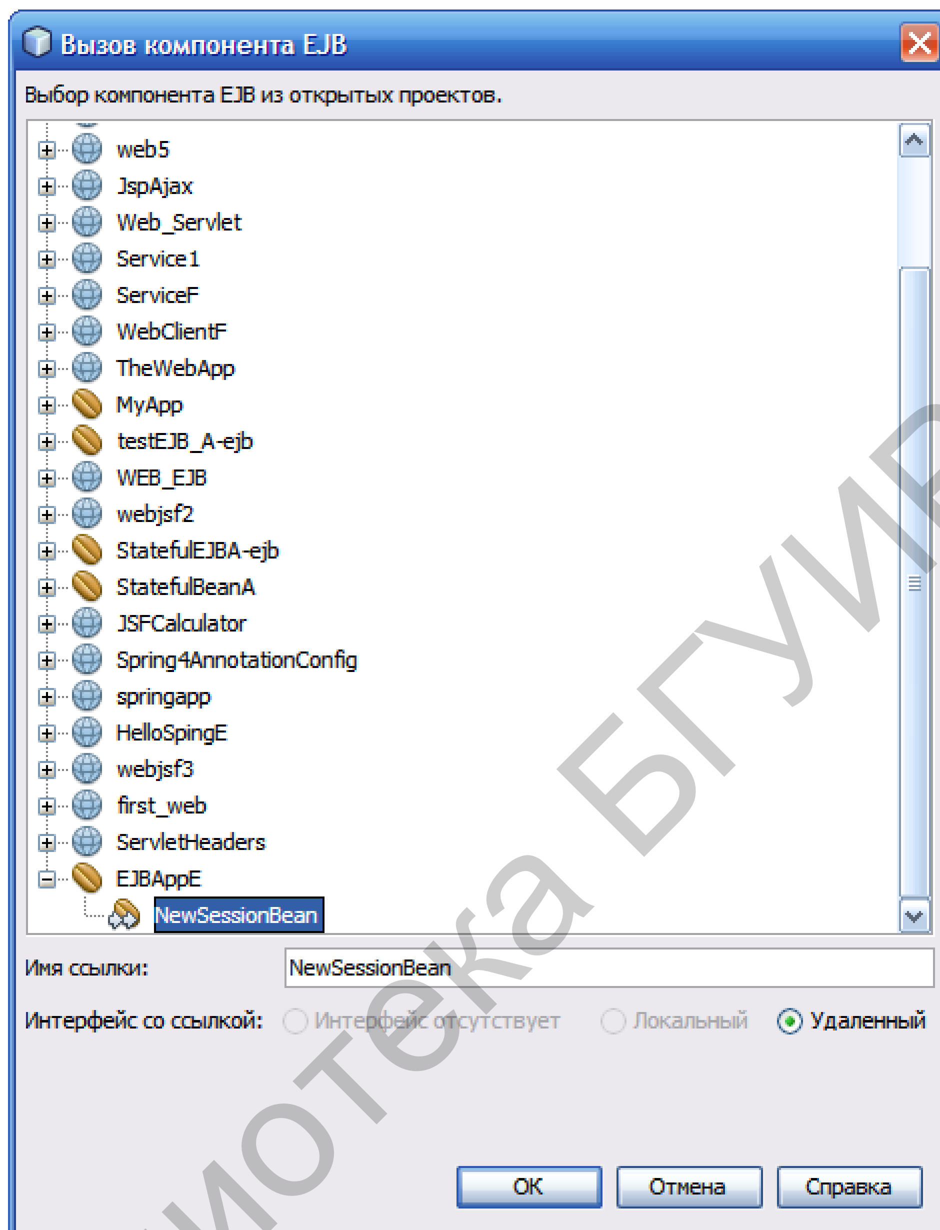


Рисунок 81 – Выбор компонента EJB

Библиотека БГУИР

Клиентское приложение запишем таким образом:

```
package ejbcliente;

import com.NewSessionBeanRemote;
import javax.ejb.EJB;

public class Main {
    @EJB
    private static NewSessionBeanRemote newSessionBean;


    public static void main(String[] args) {
```

```
        System.out.println("Currency ratings for dollar: " +
newSessionBean.showCurrency("dollar"));
        System.out.println("Currency ratings for euro: " +
newSessionBean.showCurrency("euro"));
    }
}
```

Итак, для выполнения клиентского приложения нужно предварительно развернуть серверное приложение, представленное проектом EJBAppE. После этого выполнить файл Main.java клиентского проекта. В окне вывода результатов получим следующие строки:

```
Currency ratings for dollar: 15900
Currency ratings for euro: 17200
run-single:
ПОСТРОЕНИЕ УСПЕШНО ЗАВЕРШЕНО (общее время: 43 секунд)
```

Если необходимо создать дополнительные EJB, можно просто добавить новые удаленные интерфейсы EJB к проекту библиотеки классов EJBInterfaceE.

Рассмотрим теперь создание локального боба. Такой боб доступен непосредственно в проекте, где он создан. Создаем модуль EJB (рисунок 82).

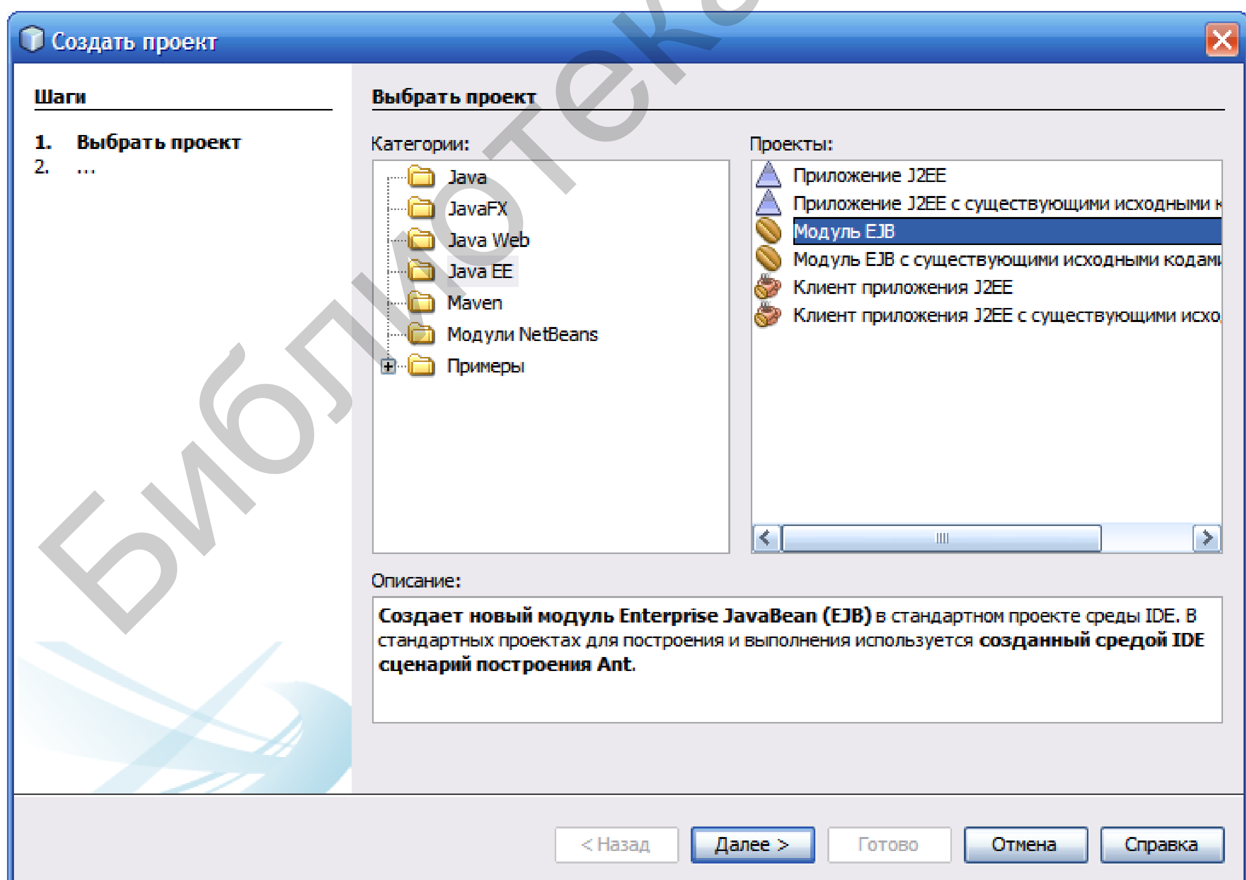


Рисунок 82 – Создание локального боба

Библиотека БГУИР

В дереве проектов (рисунок 83) на имени модуля `EjbLLocalE` в узле `Компоненты EJB` вызываем контекстное меню и создаем сеансовый компонент.

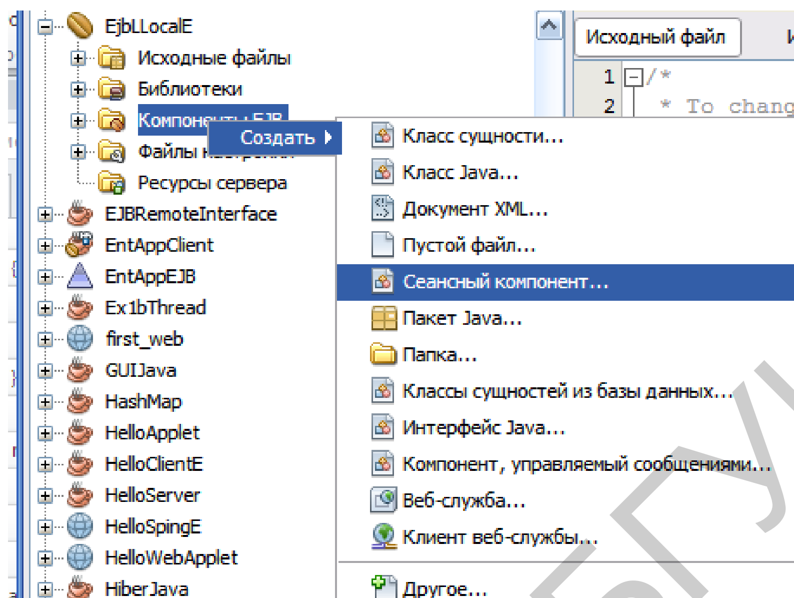


Рисунок 83 – Создание сеансового компонента

Определяем имя сеансового компонента, название пакета и его тип – `Локальный` (рисунок 84).

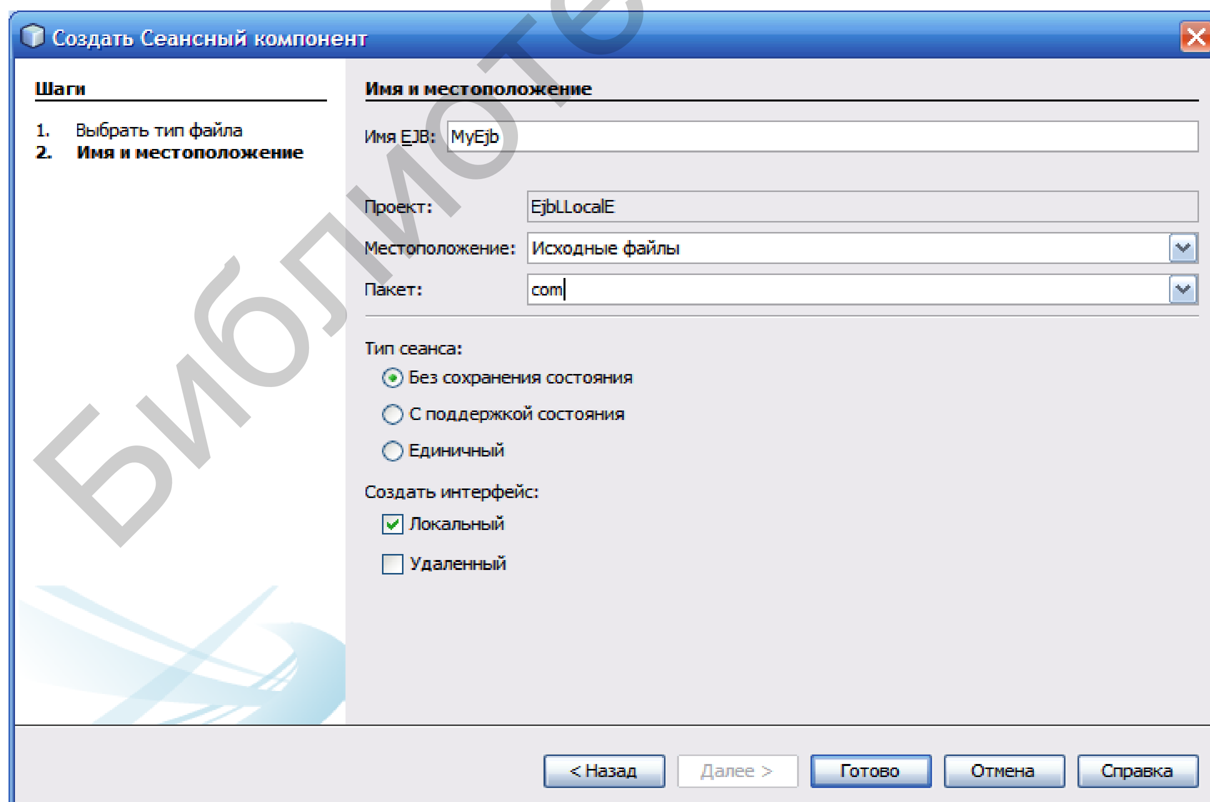


Рисунок 84 – Задание свойств сеансового компонента

Библиотека БГУИР

Расширяем функциональность компонента добавлением бизнес-метода в класс компонента. С этой целью двойным щелчком мыши на имени `MyEjb.java` открываем окно редактора, вызываем контекстное меню правой кнопкой мыши и выбираем пункт `Вставка кода`, а затем `Добавить бизнес-метод` (рисунок 85).

```
@Stateless
public class MyEjb implements MyEjbLocal {

    @Override
    public String sayDateX() {
        return ""+(new Date());
    }

}
```

Создать
Добавить бизнес-метод...
Конструктор...
toString()...
Переопределение метода...
Добавить свойство...
Вызов компонента EJB...
Использовать базу данных...
Отправка сообщения JMS...

Рисунок 85 – Расширение функциональности компонента

Задаем свойства бизнес-метода (имя, типы параметров (рисунок 86)).

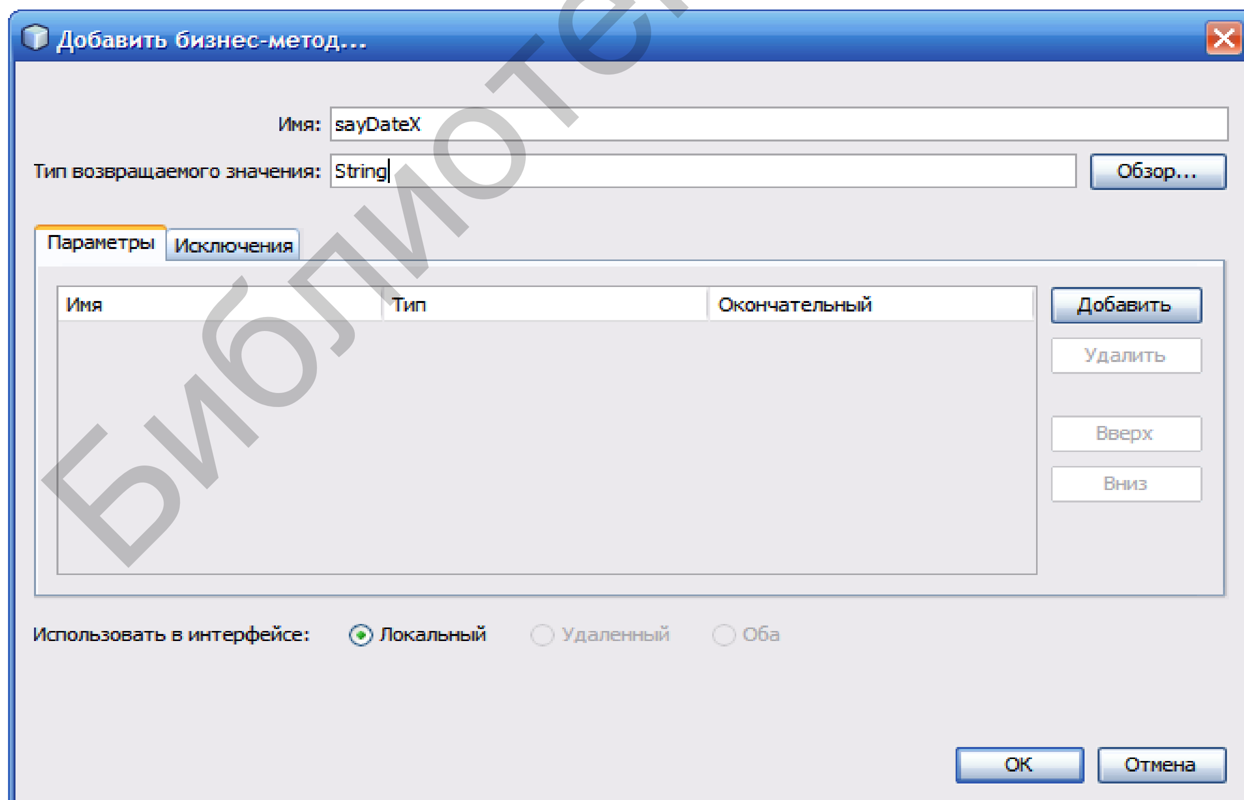


Рисунок 86 – Задание имени, типа и параметров метода

Библиотека БГУИР

Вводим следующий текст бизнес-метода в редакторе кода:

```
package com;
import java.util.*;

import javax.ejb.Stateless;
@Stateless
public class MyEjb implements MyEjbLocal {

    @Override
    public String sayDateX() {
        return ""+(new Date());
    }
}
```

Заметим, что система автоматически поместит объявление бизнес-метода в файл интерфейса:

```
package com;
import javax.ejb.Local;

@Local
public interface MyEjbLocal {

    public String sayDateX();
}
```

Папка `com` содержит два файла: файл интерфейса `MyEjbLocal.java` и файл локального сеансового компонента `MyEjb.java` (рисунок 87).

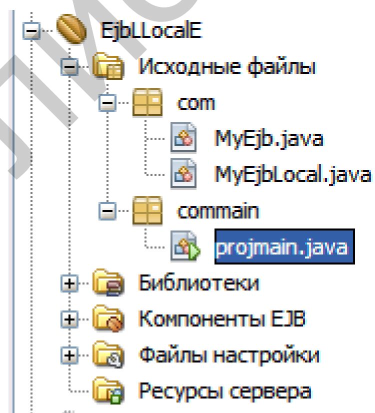


Рисунок 87 – Структура папки `com`

Создадим новый пакет `commain` и добавим в него основной файл проекта:

```
package commain;
```


Библиотека БГУИР

```
import com.*;

public class projmain {
    public static void main(String [] args)
    {
      com.MyEjbLocal ml=(com.MyEjbLocal) new com.MyEjb();
System.out.println("The Date from local session bean is:"+
ml.sayDateX());

    }
}
```

Запустим на выполнение файл `commain`:

```
run:
The Date from local session bean is:Wed Jul 22 12:45:53 MSK 2015
ПОСТРОЕНИЕ УСПЕШНО ЗАВЕРШЕНО (общее время: 1 секунда)
```

Программа использовала созданный локальный сеансовый компонент и вывела дату и время с помощью метода `sayDateX()`.

3.4 Служба JNDI

Служба JNDI существенным образом используется в технологии J2EE. Ее суть в том, что она позволяет хранить объекты в памяти, организованной по принципу файловой структуры операционной системы. JNDI представляет собой дерево контекстов. Каждый узел дерева можно связать с каталогом, и такая ассоциация вполне отражает суть дела. Для того чтобы найти объект в дереве, следует установить начальный контекст (узел дерева – `InitialContext`). Доступ к контекстам реализует провайдер (специальный сервис или, проще говоря, класс `java`). Имеется несколько различных провайдеров. Мы в качестве иллюстрации воспользуемся провайдером `RefFSContextFactory`. Соответствующий java-класс находится в архивном файле `fscontext.jar` в дистрибутиве. Отметим, что содержимое jar-архивов можно просматривать с помощью программы `WinRAR.exe`. Для иллюстрации работы JNDI нам потребуется создать два `java`-файла. Главный из них как раз и демонстрирует работу JNDI в части создания дерева контекстов, состоящего, правда, только из одного начального контекста. Вот его содержимое:

```
package lookup;

import java.util.Properties;
import javax.naming.*;
```


Библиотека БГУИР

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.Binding;
import javax.naming.NamingEnumeration;
import javax.naming.NamingException;
import java.util.Hashtable;

public class Lookup {
  public static void main(String[] args) {

   try {
  Hashtable props = new Hashtable ();
  props.put(Context.INITIAL_CONTEXT_FACTORY,
    "com.sun.jndi.fscontext.RefFSContextFactory");
  props.put(
    Context.PROVIDER_URL,
       "file:/");
 Object z0=new Fruit("orange");
 Object z1=new Fruit("tangerine");
 Context ctx = new InitialContext(props);

try
{
 ctx.bind("z0", z0);
}
catch(Exception e){
  ctx.unbind("orange");
}

try
{
 ctx.bind("z1", z1);
}
catch(Exception e2)
{ctx.unbind("tangerine");
}

       Object object = ctx.lookup("z0");
       System.out.println(""+object.toString());
       object = ctx.lookup("z1");
       System.out.println(""+object.toString());
       NamingEnumeration iter=ctx.listBindings("");
  while (iter.hasMore()) {
    Binding binding=(Binding)iter.next();
    Object cf=binding.getObject();
     if(cf.toString().indexOf("Fruit")>=0)

     {
         String ans=""+cf;
         int k=ans.indexOf("Content");
         if(k>=0)
```


Библиотека БГУИР

```
         System.out.println("\n*** binding
item:"+ans.substring(k+8));}

  }

  }
      catch (NamingException nex) {
     System.out.println("Error ="+ nex.toString());
    }
  }
}
```

Итак, запоминаем следующее. Во-первых, чтобы создать начальный контекст дерева нужно создать объект типа `Hash`-таблицы. Такая таблица состоит из двух столбцов: один (первый) столбец является именем объекта (переменной), второй – ее значением. Для чего это надо? Нужно понимать, что контекст, вообще говоря, это и есть набор пар «имя − значение». Таким образом, чтобы в будущем найти контекст, потребуется указать некоторые из значений в множестве «имя − значение». Нашу `Hash`-таблицу создаем таким образом:

```
Hashtable props = new Hashtable ();
  props.put(Context.INITIAL_CONTEXT_FACTORY,
    "com.sun.jndi.fscontext.RefFSContextFactory");
  props.put(
    Context.PROVIDER_URL,
       "file:/");

 Context ctx = new InitialContext(props);
```

Занесение значений в таблицу выполняет оператор `put`, первым аргументом которого является имя, вторым – значение.

Во-вторых, когда `Hash`-таблица сформирована, строим начальный контекст:

```
     ctx = new InitialContext(props);
```

В-третьих, теперь можно вставить в дерево объекты для хранения. Такая операция называется связыванием объекта (`bind` – связывать). Делаем это так:

```
 Object z0=new Fruit("orange");
 Object z1=new Fruit("tangerine");

try
{
 ctx.bind("z0", z0);
}
catch(Exception e){
```


Библиотека БГУИР

```
  ctx.unbind("orange");
}

try
{
 ctx.bind("z1", z1);
}
catch(Exception e2)
{ctx.unbind("tangerine");
}
```

Здесь мы сначала создаем объект(ы) класса `Fruit`, а затем вставляем его (их) в дерево контекстов под именем «`z0`» и «`z1`» соответственно. Теперь, чтобы получить объект из дерева контекстов, мы используем оператор `lookup`:

```
       Object object = ctx.lookup("z0");
       System.out.println(""+object.toString());
       object = ctx.lookup("z1");
       System.out.println(""+object.toString());
```

При этом выполняется поиск объекта в дереве контекстов. Разумеется, начальный контекст удерживается в переменной `initctx`. Теперь возникает вопрос, что из себя представляет объект класса `Fruit`? Этот объект должен быть ссылкой (`Reference`) или ссылочным объектом (`Referenceable`). Класс `Fruit` представляет второй `java`-файл и имеет такой вид:

```
package lookup;

import javax.naming.*;


public class Fruit  implements Referenceable
{
  String fruit;

  public Fruit(String f)
  {
     fruit=f;
  }

    public Reference getReference() throws NamingException
  {
      return new Reference(
              Fruit.class.getName(),
              new StringRefAddr("fruit",fruit));

     }
  }
```

Библиотека БГУИР

Во-первых, класс Fruit наследует абстрактный класс Referenceable и должен реализовать его единственный метод getReference. В конструкторе класса Fruit просто инициализируется объект данного класса. Метод getReference() возвращает ссылку на объект, вызывая конструктор new Reference(). В этот конструктор передаем имя класса Fruit и адрес создаваемого объекта, который получаем по команде new StringRefAddr("fruit", fruit). Заметим, что имеется несколько вариантов конструкторов Reference(). В наиболее полном варианте требуется передать название класса-фабрики объектов и его URL. Класс-фабрика порождает объекты по адресным ссылкам. В этом случае нам потребуется третий java-файл. Файл Fruit.java в этом случае следует представить так:

```
import javax.naming.*;

public class Fruit implements Referenceable {
    String fruit;

    public Fruit(String f) {
     fruit = f;
    }

       public Reference getReference() throws NamingException {
     return new Reference(
         Fruit.class.getName(),
         new StringRefAddr("fruit", fruit),
         FruitFactory.class.getName(),
         null);
  }
}
```

Фабрика FruitFactory реализована в следующем java-файле:

```
import javax.naming.*;
import javax.naming.spi.ObjectFactory;
import java.util.Hashtable;

/* создает объект класса Fruit по ссылке */

public class FruitFactory implements ObjectFactory {
    public FruitFactory() {
                                    }

    public  Object  getObjectInstance(Object  obj,  Name  name,
Context ctx,
     Hashtable env) throws Exception {
     if (obj instanceof Reference) {
         Reference ref = (Reference)obj;
         if (ref.getClassName().equals(Fruit.class.getName())) {
```

```
            RefAddr addr = ref.get("fruit");
            if (addr != null) {
                return new Fruit((String)addr.getContent());
            }
          }
      }
      return null;
    }
}
```

Мы видим, что объект создается по адресной ссылке таким образом:

```
return new Fruit((String)addr.getContent());
```

Итак, остается скомпилировать приведенные `java`-файлы и посмотреть их работу:

```
run:
Reference Class Name: lookup.Fruit
Type: fruit
Content: orange

Reference Class Name: lookup.Fruit
Type: fruit
Content: tangerine


 *** binding item: orange


 *** binding item: tangerine

ПОСТРОЕНИЕ УСПЕШНО ЗАВЕРШЕНО (общее время: 0 секунд)
```

Обратим внимание на то, что поиск объекта выполняется по имени, под которым он был ранее сохранен в дереве контекстов.

Итак, мы рассмотрели введение в `JNDI`. Показали, что `JNDI` выполняет роль хранилища объектов и имеет структуру дерева каталогов. Уже саму `JNDI` можно рассматривать в качестве баз данных. `JNDI` используется в системе `J2EE` для доступа к объектам интерфейса (проще говоря, объектам сервера, сохраненным в системе `JNDI`). Доступ к дереву контекстов в `J2EE` выполняет собственный сервер.

Библиотека БГУИР

3.5 Современные технологии, использующие компоненты `Hibernate, Spring`

Технологии `Hibernate` и `Spring` являются весьма «продвинутыми» реализациями компонентно-объектного программирования. Начнем с `Hibernate`.

3.5.1 Технология `Hibernate` и персистентные классы

Эта технология предназначена для работы с базой данных как с классом (такой класс называется персистентным – persistent (устойчивый)). Поля таблицы представляются объектами. Доступ к полям выполняется с помощью методов `set` и `get`.

Создадим проект на базе обычного приложения Java. Назовем его `HiberLabE`. Подключим к нему библиотеку `Hibernate JPA` (рисунок 88).

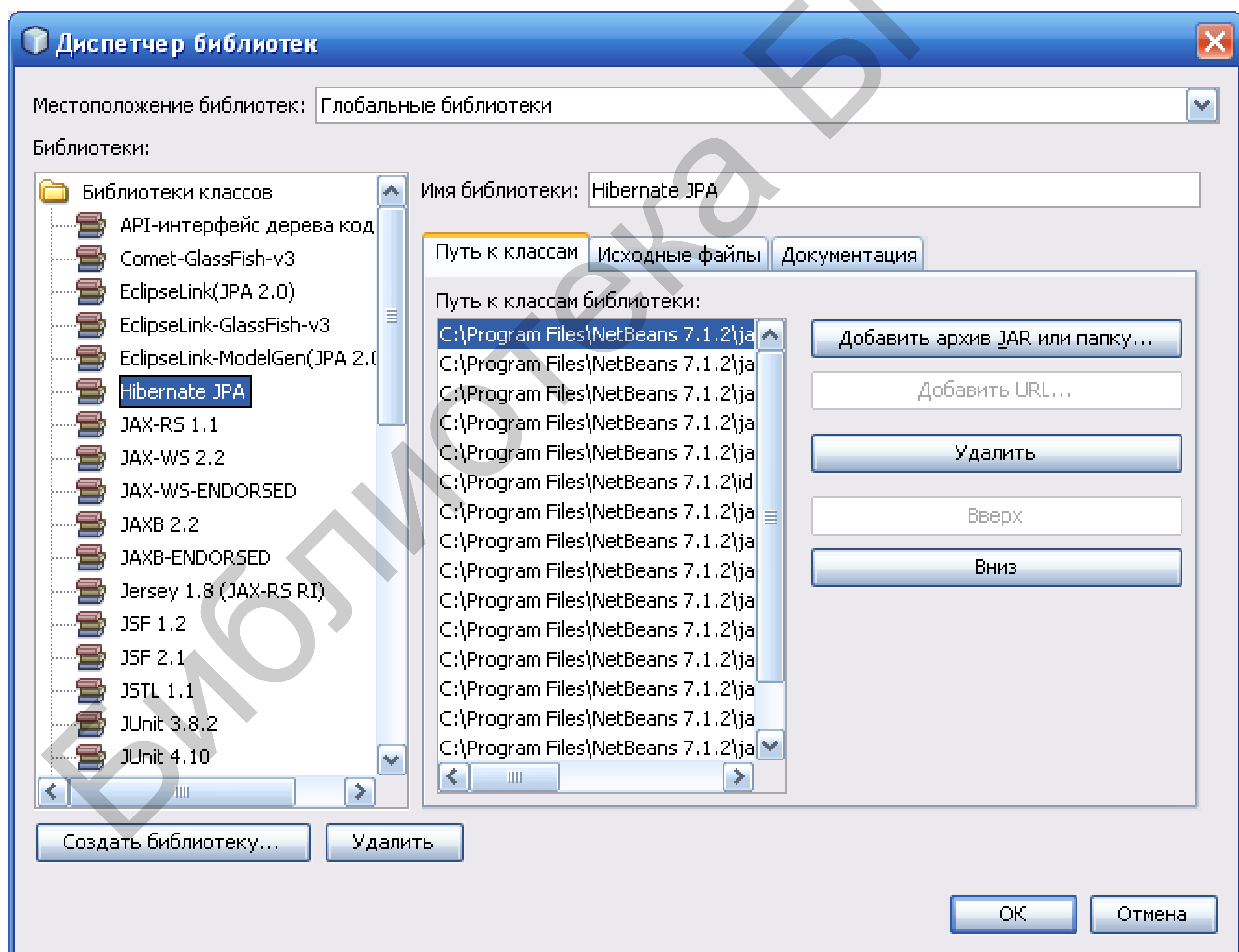


Рисунок 88 – Подключение библиотеки `Hibernate JPA`

Библиотека БГУИР

Кроме того, поскольку будем работать с базой данных derby, подключаем драйвер derbyClient.jar из папки lib инсталляции GlassFish.

Наберем следующий текст.

```
package hiberlabe;

import org.hibernate.annotations.GenericGenerator;
import java.util.*;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import org.hibernate.Session;
 class stud{

   private String fio;
   private int age;

   public stud() {}

   public stud(String name, int g) {


       this.fio = name;
       this.age = g;

   }

    public void setFio(String name) {
        this.fio = name;
    }

    public int getAge() {
        return age;
    }

    public String getFio() {
        return fio;
    }

    public void setAge(int g) {
        this.age = g;
```


Библиотека БГУИР

```
    }
}

public class HiberLabE {

      @SuppressWarnings("unused")

      private static SessionFactory sessionFactory;
      static Session session = null;
      static  Connection connection = null;
      static  Statement statement = null;
      static  ResultSet rs = null;
      private static String dbURL =
"jdbc:derby://localhost:1527/MeineBase;user=oleg;password=german
";


      private static Connection conn = null;
   public static void main(String[] args) {

        System.out.println("Start building Factory");
        sessionFactory = new
Configuration().configure().buildSessionFactory();
        System.out.println("Session Factory has been built
successfully");
        System.out.println("Make a connection to database");
        try
        {

Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance(
);
            conn = DriverManager.getConnection(dbURL);
        }
        catch (Exception except)
        {
         System.out.println("Error in connection:
"+except.getMessage());
        }

           System.out.println("Connection has been established
with Derby database");
           session = sessionFactory.openSession();
           session.beginTransaction();
           List<stud> result=session.createQuery("from
stud").list();
            for(stud x:result)
            {
             System.out.println(">>> "+x.getFio()+" ***
"+x.getAge()) ;

            }
           stud novice=new stud();
```


Библиотека БГУИР

```
            novice.setFio("Golsworthy");
            novice.setAge(30);

            session.save(novice);
            System.out.println("new collection disposition");

              for(stud x:result)
             {
              System.out.println(">>> "+x.getFio()+" ***
"+x.getAge()) ;

              }

              session.getTransaction().commit();
              try
              {
              conn.close();
              }

         catch(Exception ex)
         {
          System.out.println("Connection has NOT been
established!!!");
          }
        }
     }
```

Какова общая идея? `Hibernate` работает с базами данных через классы. Класс должен быть настроен на структуру таблицы базы данных. В нашем примере таким классом является следующий:

```
class stud
  {
   private String fio;
   private int age;
   public stud() {}

   public stud(String name, int g) {
       this.fio = name;
       this.age = g;
   }

    public void setFio(String name) {
        this.fio = name;
    }

    public int getAge() {
        return age;
    }

    public String getFio() {
```


Библиотека БГУИР

```
        return fio;
    }

    public void setAge(int g) {
        this.age = g;
    }
}
```

В классе описаны поля `fio`, `age`. Эти поля должны быть сопоставлены с колонками таблицы. Для этого мы используем специальный `map`-файл. Вот его вид:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE   hibernate-mapping   PUBLIC   "-//Hibernate/Hibernate
Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="hiberlabe">
    <class name="stud" table="stud">
       <id name="fio" column="fio">
            <generator class="native"/>
        </id>

        <property name="age" column="age"/>
    </class>
</hibernate-mapping>
```

Мы видим, что таблица привязана к классу `stud`. Имя таблицы также `stud`. Имена полей класса сопоставлены со столбцами таблицы. Заметим, что один столбец должен быть ключевым (в нашем примере – это столбец `fio`). Ключевой столбец объявляется в теге `<id>`. Наконец, очень важно: нужно в классе `stud` написать `get` и `set` для полей. Внимание! Имя поля указывается в `get` и `set`:

```
    public void setFio(String name) {
        this.fio = name;
    }

    public int getAge() {
        return age;
    }

    public String getFio() {
        return fio;
    }

    public void setAge(int g) {
        this.age = g;
    }
```


Библиотека БГУИР

Этот файл имеет имя `stud.hbm.xml`. `Stud` – это имя класса (вообще имя произвольно). Файл нужно добавить в папку `src` проекта. Нужно создать еще один `xml`-файл: `hibernate.cfg.xml`. Он стандартный и имеет такой вид:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-
3.0.dtd">

<hibernate-configuration>

    <session-factory>

      <!-- Database connection settings -->
        <property
name="connection.driver_class">org.apache.derby.jdbc.ClientDrive
r</property>
        <property
name="connection.url">jdbc:derby://localhost:1527/MeineBase</pro
perty>
        <property name="connection.username">oleg</property>
        <property name="connection.password">german</property>

      <!-- SQL dialect -->
        <property
name="dialect">org.hibernate.dialect.DerbyDialect</property>

      <!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">1</property>

      <!-- Enable Hibernate's automatic session context
management -->
        <property
name="current_session_context_class">thread</property>

      <!-- Disable the second-level cache -->
        <property
name="cache.provider_class">org.hibernate.cache.NoCacheProvider<
/property>

      <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">true</property>


      <!-- Mapping files -->
        <mapping resource="stud.hbm.xml"/>

    </session-factory>

</hibernate-configuration>
```

Библиотека БГУИР

В этом файле объявлен класс драйвера:

```
<property
name="connection.driver_class">org.apache.derby.jdbc.ClientDrive
r</property>
```

Объявлена строка соединения:

```
        <property
name="connection.url">jdbc:derby://localhost:1527/MeineBase</pro
perty>,
```

а также имя пользователя и пароль. Одна из наиболее важных строк

```
          <mapping resource="stud.hbm.xml"/>
```

показывает связь с файлом `map`, описанным ранее. Размещение этих файлов иллюстрируется рисунком 89.

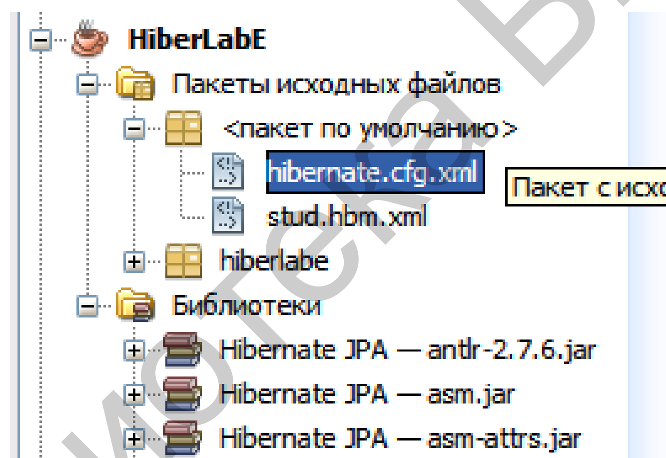


Рисунок 89 – Размещение конфигурационных файлов

И вот, наконец, использование `Hibernate`:

```
           List<stud> result=session.createQuery("from
stud").list();
            for(stud x:result)
            {
             System.out.println(">>> "+x.getFio()+" ***
"+x.getAge()) ;

          }
           stud novice=new stud();
           novice.setFio("Golsworthy");
           novice.setAge(30);
           session.save(novice);
```


Библиотека БГУИР

```
            System.out.println("new collection disposition");

              for(stud x:result)
             {
              System.out.println(">>> "+x.getFio()+" ***
"+x.getAge()) ;

             }
```

Список студентов формируется в виде коллекции

```
List<stud> result=session.createQuery("from stud").list();
```

Затем производится вывод списка на консоль

```
             for(stud x:result)
             {
              System.out.println(">>> "+x.getFio()+" ***
"+x.getAge()) ;
```

Окно с выходными результатами показано на следующем скриншоте (рисунок 90).

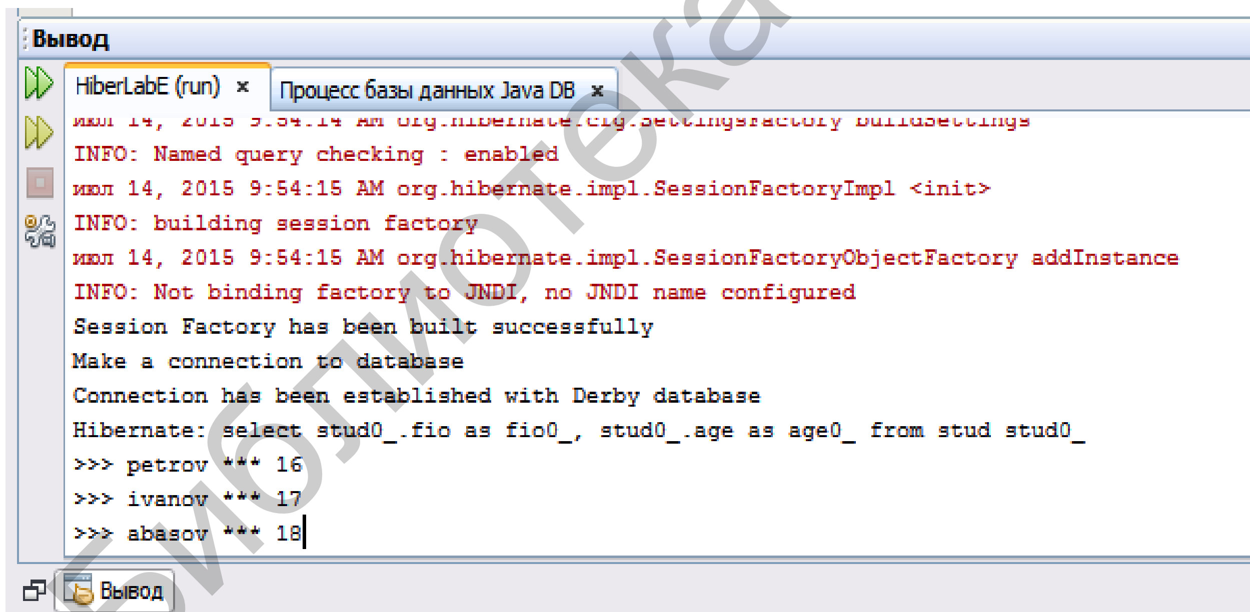


Рисунок 90 – Выходное окно со списком студентов

Рассмотрим, как задавать условия отбора записей при вызове типа

```
List<stud> result=session.createQuery("from stud").list();
```

Перепишем метод `Main` следующим образом:

Библиотека БГУИР

```
public static void main(String[] args) {
       System.out.println("Start building Factory");
        sessionFactory                    =                  new
Configuration().configure().buildSessionFactory();
        System.out.println("Session  Factory  has  been  built
successfully");
        System.out.println("Make a connection to database");

        try
        {

Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance(
);
            conn = DriverManager.getConnection(dbURL);
        }
        catch (Exception except)
        {
         System.out.println("Error         in         connection:
"+except.getMessage());
        }
    System.out.println("Connection  has  been  established  with
Derby database");
           session = sessionFactory.openSession();
           session.beginTransaction();
           String hql = "from stud where AGE=16";
           org.hibernate.Query query = session.createQuery(hql);
           List<stud> result= query.list();
            for(stud x:result)
            {
             System.out.println(">>>      "+x.getFio()+"      ***
"+x.getAge()) ;
       }
             session.getTransaction().commit();
             try
             {
             conn.close();
             }
        catch(Exception ex)
        {
         System.out.println("Connection     has     NOT     been
established!!!");
        }
      }
    }
```

Фильтрация записей реализуется заданием условия отбора в строке

```
String hql = "from stud where AGE=16";
```

Выходное окно имеет теперь следующее содержимое:

```
Connection has been established with Derby database
```


Библиотека БГУИР

```
Hibernate: select stud0_.fio as fio0_, stud0_.age as age0_ from
stud stud0_ where AGE=16
>>> petrov *** 16
ПОСТРОЕНИЕ УСПЕШНО ЗАВЕРШЕНО (общее время: 1 секунда)
```

Следующий пример дает некоторый выход за рамки `Hibernate`, хотя «идеологически» выполнен в том же ключе. Он использует так называемый `entity`-класс для связи с базой данных (`entity` – сущность). Создадим обычное `java`-приложение и назовем его `LabEntity`. Добавим к нему `jar`-файл с классом драйвера `Derby`. На узле `Исходные пакеты` создадим `entity`-класс из контекстного меню в папке `com` (рисунок 91).

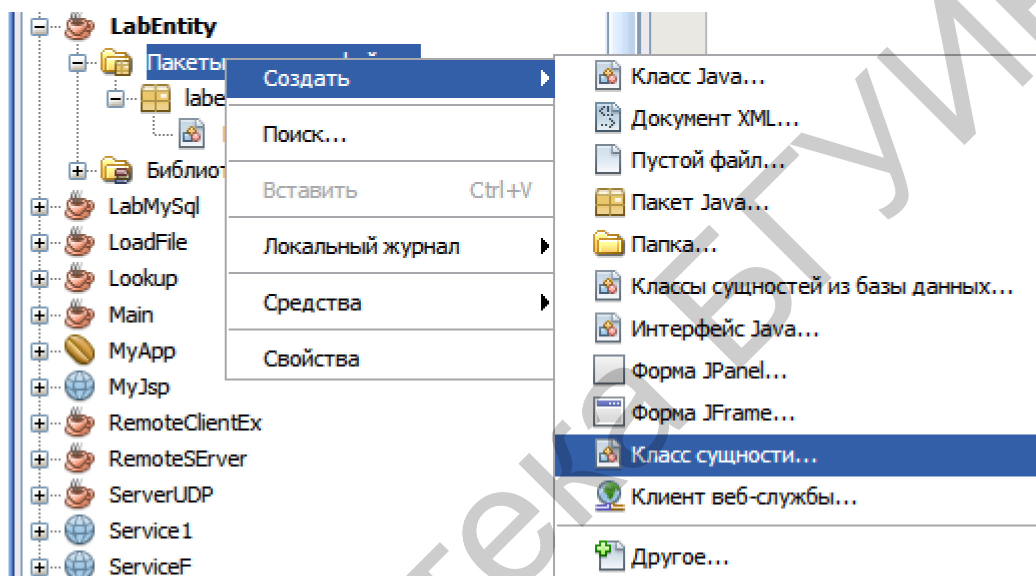


Рисунок 91 – Добавление класса `Entity`

На скриншоте (рисунок 92) указывается физическая база данных, с которой связывается `entity`-класс.

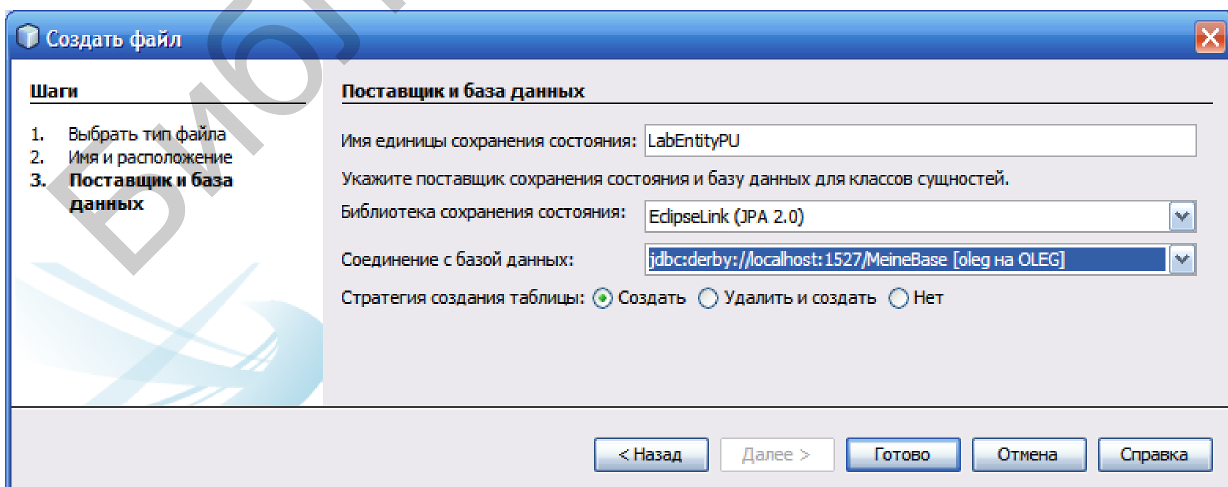


Рисунок 92 – Подключение к существующей базе данных

В последующем программировании нам потребуется значение единицы сохранения состояния (`persistence unit`) – `LabEntityPU`.

Первоначально созданный класс имеет такой вид:

```
package com;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Person implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object)
       {
        // TODO: Warning - this method won't work in the case
the id fields are not set
        if (!(object instanceof Person)) {
            return false;
        }
        Person other = (Person) object;
        if ((this.id == null && other.id != null) || (this.id !=
null && !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
```


Библиотека БГУИР

```
    public String toString() {
        return "com.Person[ id=" + id + " ]";
    }

}
```

Изменим этот класс следующим образом:

```
package com;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Person implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private String name;
    private int age;

    public String getId() {
        return name;
    }

    public void setId(String id) {
        this.name = id;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age)
    {
        this.age = age;
    }

    @Override
    public int hashCode()
    {
        int hash = 0;
        hash += (name != null ? name.hashCode() : 0);
        return hash;
    }

    @Override
```

Библиотека БГУИР

```
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case
the id fields are not set
        if (!(object instanceof Person)) {
            return false;
        }
        Person other = (Person) object;
        if ((this.name == null && other.name != null) ||
(this.name != null && !this.name.equals(other.name))) {
            return false;
        }
        else
          if (this.age!=other.age) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "com.Person[ name=" + name + " ]";
    }

}
```

В данном классе объявлены свойства `name` и `age`, соответствующие столбцам таблицы базы данных. Для каждого из этих свойств определен метод `set/get`. Обратим внимание на то, что за ключевым словом `set/get` следует имя свойства (например `getAge()`). В таблице обязано быть ключевое поле. Такое поле объявляется с атрибутом `@Id`. В нашем примере роль ключевого поля играет `name`. Отсюда имеем соответствующие методы `set/get`:

```
public String getId() {
        return name;
    }

    public void setId(String id) {
        this.name = id;
    }
```

Атрибут `@GeneratedValue(strategy = GenerationType.AUTO)` указывает, что значение индекса (если он целочисленный) наращивается автоматически при вставке новой записи в таблицу. Кроме того, необходимо переписать методы `hashCode()`,`equals()`,`toString()`. В рассматриваемом примере эти методы не играют какой-либо роли и их можно было оставить пустыми. Приведенный `entity`-класс и играет роль класса, представляющего связь с реляционной таблицей базы данных. Чтобы манипулировать этим

Библиотека БГУИР

классом (вставлять и просматривать записи), нам нужен основной класс приложения. Теперь перепишем основной класс `LabEntity`:

```
package labentity;
import com.Person;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import java.util.List;
import javax.persistence.PersistenceContext;
import javax.persistence.TypedQuery;

public class LabEntity {


    public static void main(String[] args) {
     Person p1= new Person();
     p1.setId("johny");
     p1.setAge(20);
     LabEntity le= new LabEntity();
     le.persist(p1);
     Person p2= new Person();
     p2.setId("kitty");
     p2.setAge(21);
     le.persist(p2);
     EntityManagerFactory emf=
javax.persistence.Persistence.createEntityManagerFactory("LabEnt
ityPU");
     EntityManager em=emf.createEntityManager();
     TypedQuery<Person>   query=em.createQuery("Select   g   from
Person g ORDER BY g.age",Person.class);
     List<Person> result= query.getResultList();
            for(Person x:result)
            {
             System.out.println(">>>      "+x.getId()+"       ***
"+x.getAge()) ;
       }
    }

    public void persist(Object ob)
    {
      EntityManagerFactory emf=
javax.persistence.Persistence.createEntityManagerFactory("LabEnt
ityPU");
      EntityManager em=emf.createEntityManager();
      em.getTransaction().begin();
      try
      {
          em.persist(ob);
          em.getTransaction().commit();

      }
```


Библиотека БГУИР

```
        catch(Exception e)
        {
            em.getTransaction().rollback();
        }
        finally
        {
            em.close();
        }
      }
      }
```

Результат работы программы такой:

```
[EL Info]: 2015-07-20 23:15:36.312--ServerSession(18178439)--
EclipseLink, version: Eclipse Persistence Services -
2.3.0.v20110604-r9504

[EL Info]: 2015-07-20 23:15:36.609--ServerSession(18178439)--
file:/E:/work5/LabEntity/build/classes/_LabEntityPU login
successful

>>> johny *** 20

>>> kitty *** 21

ПОСТРОЕНИЕ УСПЕШНО ЗАВЕРШЕНО (общее время: 3 секунды)
```

Для работы с `entity`-классом мы создаем `entity`-менеджер:

```
EntityManagerFactory emf=
javax.persistence.Persistence.createEntityManagerFactory("LabEnt
ityPU");
      EntityManager em=emf.createEntityManager();
```

Этот менеджер может сохранять записи в физической таблице базы данных:

```
em.persist(ob); //объект ob – запись, вставляемая в таблицу
```

Выборку и просмотр записей из таблицы базы данных осуществляет следующий фрагмент кода:

```
TypedQuery<Person> query=em.createQuery("Select g from Person g
ORDER BY g.age",Person.class);
     List<Person> result= query.getResultList();
            for(Person x:result)
            {
             System.out.println(">>>      "+x.getId()+"      ***
"+x.getAge()) ;
```


Библиотека БГУИР

```
        }
}
```

В результате выполнения данного приложения в базе данных появилась таблица `Person` (рисунок 93).

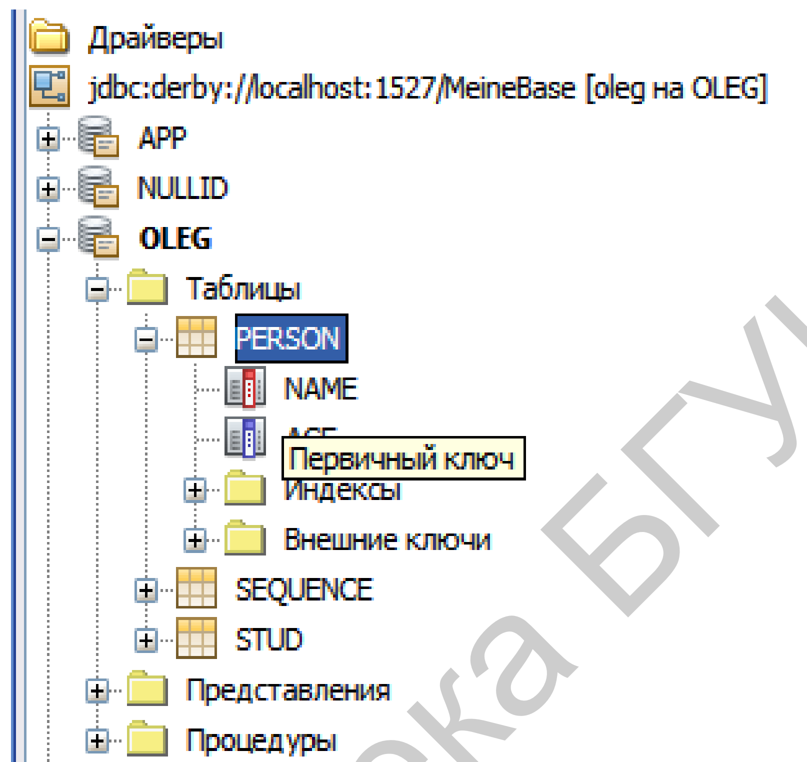


Рисунок 93 – Создание таблицы `Person`

3.5.2 Технология `Spring`

Данная технология включает триаду `<Model – View – Controller>`. Часть `Model` представляет класс(ы), ассоциированный с базой данных (такой класс ранее в тексте определен как `entity`-класс или персистентный класс). Часто используется `Hibernate`. Часть `View` служит для визуализации данных. Представляет `jsp`- или `html`-файлы. Часть `Controller` представляет бизнес-логику, т. е. методы для обработки данных. Создадим учебное приложение `Spring`.

Используем тип проекта `web`-приложение. Дадим ему имя `SpringApp`. В процессе создания указываем платформу `Spring Web MVC` (см. скриншот на рисунке 94).

Библиотека БГУИР

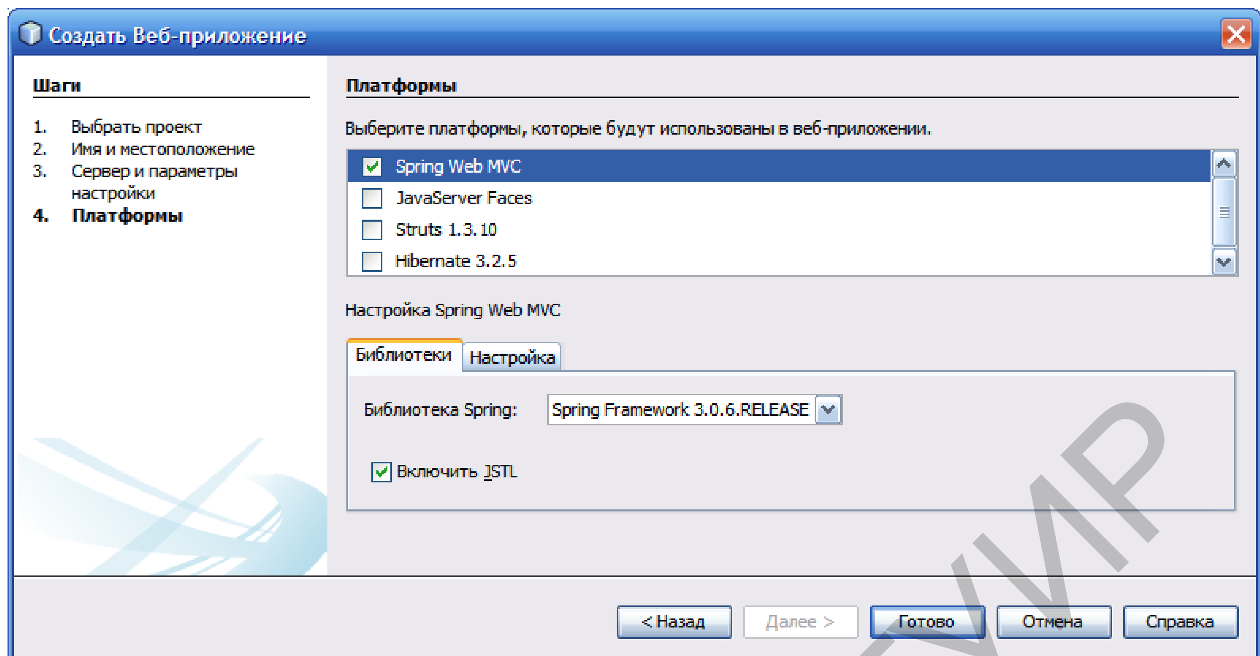


Рисунок 94 – Задание платформы `Spring MVC`

Конечная структура проекта показана на рисунке 95.

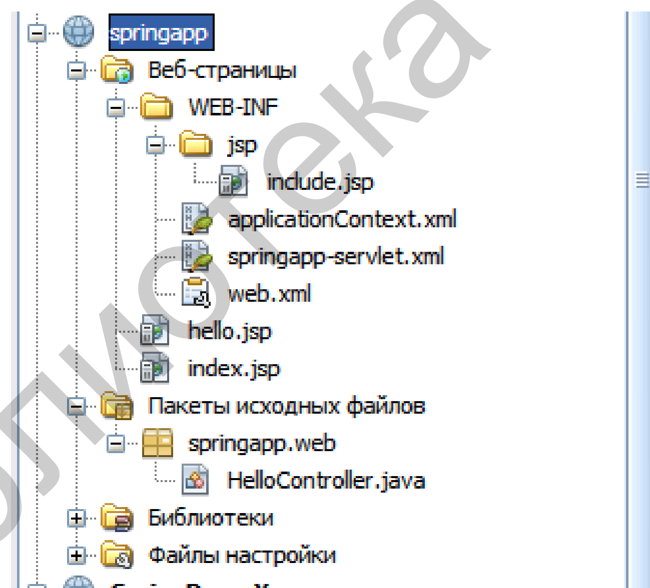


Рисунок 95 – Структура проекта для приложения `Spring MVC`

Отредактируем файл `web.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
         xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
         http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" >
  <servlet>
```

Библиотека БГУИР

```
    <servlet-name>springapp</servlet-name>
    <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet
-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>springapp</servlet-name>
    <url-pattern>*.htm</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>
      index.jsp
    </welcome-file>
  </welcome-file-list>
</web-app>
```

Здесь указывается, что начальным файлом проекта является `index.jsp`. Кроме того, контроллер представлен сервлетом, а `view` – документами `htm`. Один сервлет – `DispatcherServlet` является системным (служит для перенаправления вызовов от объектов `view` в связанные с ними контроллеры).

Далее изменяем содержимое файла `index.jsp` следующим образом:

```
<%@ page session="false"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"
%>

<html>
  <head><title>Example :: Spring Application</title></head>
 <c:redirect url="/hello.htm"/>
</html>
```

Здесь происходит простая переадресация на новый `url`: `hello.htm`. Создаем файл `hello.jsp`:

```
<%@ page session="false"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"
%>
<html>
  <head><title>Hello :: Spring Application</title></head>
  <body>
    <h1>Hello - Spring Application</h1>
    <p>Greetings, it is now <c:out value="${now}"/> Bye-Bye</p>
  </body>
</html>
</html>
```


Библиотека БГУИР

Именно на этот файл и производится переадресация при запуске приложения (хотя расширение у него jsp, а не html – это один и тот же файл). Здесь есть одно важное для нас место (вспомним JSF):

```
<c:out value="${now}"/>
```

В это место класс контроллера (сервлет) подставляет значение переменной now, которая из самого контроллера и передается. Создаем контроллер. Это обычный класс java-сервлета (по сути). Размещаем его в узле springapp.web Пакета исходных файлов:

```
package springapp.web;
import org.springframework.web.servlet.mvc.Controller;
import org.springframework.web.servlet.ModelAndView;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import java.io.IOException;
import java.util.*;

public class HelloController implements Controller
    {

     protected final Log logger = LogFactory.getLog(getClass());
     public HelloController()
    {

    }
    public ModelAndView handleRequest (HttpServletRequest
request, HttpServletResponse response)

            throws ServletException, IOException
    {
        logger.info("Returning hello view");
        String now = (new Date()).toString();
        return new ModelAndView("hello.jsp", "now", now);
    }
  }
```

Метод handleRequest является ключевым. Мы можем использовать его переменную request для получения значения из документа (view) и вернуть результат:

```
return new ModelAndView("hello.jsp", "now", now);
```

Здесь как раз и указывается, что возвращается переменная now по месту с таким же именем, как и в документе hello.jsp, т. е. в позицию

Библиотека БГУИР

```
<c:out value="${now}"/>
```

Конфигурационный файл applicationContext.xml должен иметь такой вид:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
       http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
       http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.0.xsd">

    <!--bean id="propertyConfigurer"

class="org.springframework.beans.factory.config.PropertyPlacehol
derConfigurer"
          p:location="/WEB-INF/jdbc.properties" />

    <bean id="dataSource"

class="org.springframework.jdbc.datasource.DriverManagerDataSour
ce"
          p:driverClassName="${jdbc.driverClassName}"
          p:url="${jdbc.url}"
          p:username="${jdbc.username}"
          p:password="${jdbc.password}" /-->

    <!-- ADD PERSISTENCE SUPPORT HERE (jpa, hibernate, etc) -->
</beans>
```

Наконец, конфигурационный файл springapp-servlet.xml – это переименованный файл dispatcher-servlet.xml. Его содержимое следует скорректировать таким образом:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-
2.5.xsd">
  <!-- the application context definition for the springapp
DispatcherServlet -->
```


Библиотека БГУИР

```
  <bean name="/hello.htm"
class="springapp.web.HelloController"/>
</beans>
```

Здесь важным является привязка документа hello.htm к классу контроллера springapp.web.HelloController (т. е. закрепление контроллера за представлением view). Привязка реализуется через тег <bean>:

```
  <bean name="/hello.htm"
class="springapp.web.HelloController"/>
```

Итак, проект Spring создан. Выполним его (Очистить и построить, затем Развернуть, затем Выполнить). Результат показан на рисунке 96.

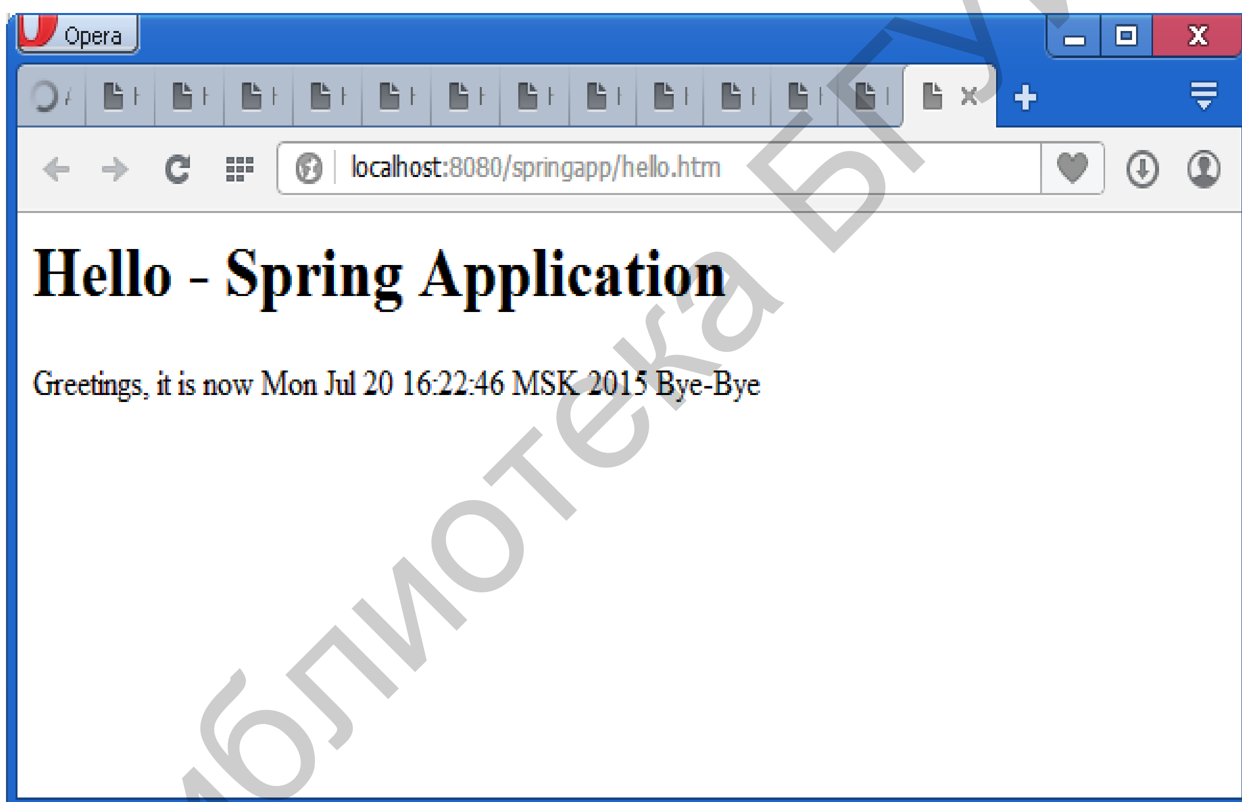


Рисунок 96 – Окно приложения Spring

Очевидным образом данный учебный пример можно развивать в части усложнения обработки, выполняемой контроллером, добавления новых представлений и новых контроллеров.

Библиотека БГУИР

4 ЛАБОРАТОРНЫЙ ПРАКТИКУМ

4.1 Работа с web-ресурсами

Цель работы: познакомиться с возможностями поиска информации в удаленных URL-ресурсах.

Краткое теоретическое содержание

В подразделе 2.4 представлены сведения по работе с URL-ресурсами. Нас интересует поиск различных информационных документов по набору ключевых слов. В настоящей лабораторной работе следует создать небольшое приложение, которое позволяет ввести ключевые слова и найти любой (один) html-документ соответствующий (или, как говорят, релевантный) введенному набору ключевых слов. Исходную форму для поисковика можно реализовать в соответствии с рисунком 97.

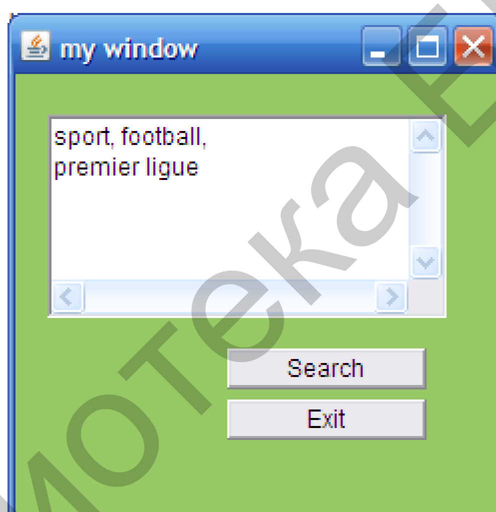


Рисунок 97 – Окно приложения

В текстовой области вводим ключевые слова. Нажимаем кнопку Search. Выполняется поиск по документам, записанным в пакете source_html (рисунок 98).

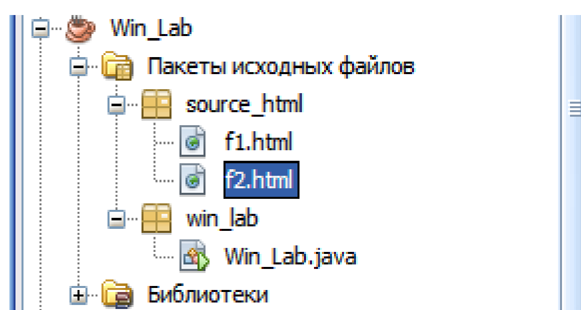


Рисунок 98 – Определение области поиска

Библиотека БГУИР

Найденный документ следует открыть в браузере (например Opera), как показано на рисунке 99.

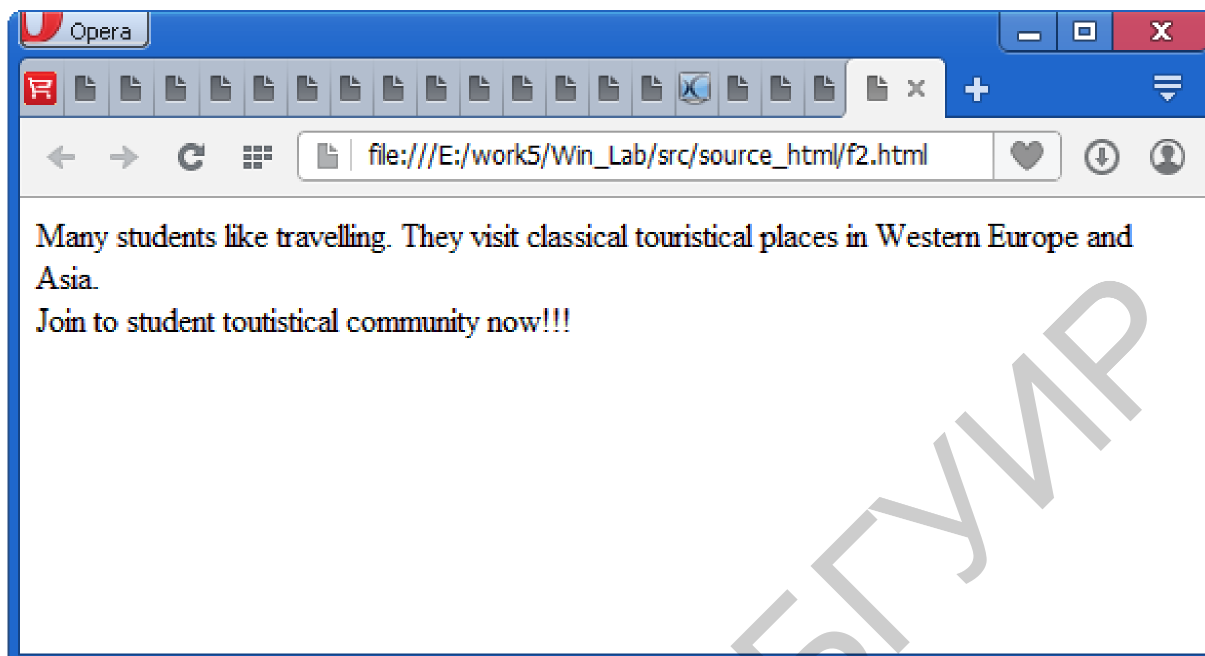


Рисунок 99 – Открытие документа в браузере

Помещаем «заготовку» приложения, которую следует доработать.

```
package win_lab;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.net.URL;
import java.net.URLConnection;
import java.nio.charset.MalformedInputException;

public class Win_Lab extends Frame implements ActionListener{
     Button bex=new Button("Exit");
     Button sea=new Button("Search");
     TextArea txa = new TextArea();
  public  Win_Lab()
{
  super("my window");
  setLayout(null);
  setBackground(new Color(150,200,100));
  setSize(450,250);
  add(bex);
```


Библиотека БГУИР

```
  add(sea);
  add(txa);
  bex.setBounds(110,190,100,20);
  bex.addActionListener(this);
  sea.setBounds(110,165,100,20);
  sea.addActionListener(this);
  txa.setBounds(20,50,300,100);

  this.show();
  this.setLocationRelativeTo(null);
}

  public void actionPerformed(ActionEvent ae)
{
   if(ae.getSource()==bex)
   System.exit(0);
   else
    if (ae.getSource()==sea)
    {
         String [] keywords=txa.getText().split(",");
         for (int j=0;j<keywords.length;j++)
        {
            System.out.println(keywords[j]);

        }
        File f = new File("e:/work5/Win_lab/src/source_html");
        ArrayList<File> files =
 new ArrayList<File>(Arrays.asList(f.listFiles()));
        txa.setText("");
        for (File elem : files)
        {
          int zcoincidence = test_url(elem,keywords);
          txa.append("\n"+elem+"  :"+zcoincidence);
        }}}

  public static int test_url(File elem, String [] keywords)
  {
      int res=0;
       URL url = null;
       URLConnection con = null;
       int i;
         try
     {
          String ffele=""+elem;
          url = new URL("file:/"+ffele.trim());
          con = url.openConnection();
          File file = new File("e:/work5/rezult.html");
          BufferedInputStream bis = new BufferedInputStream(
                                 con.getInputStream());
          BufferedOutputStream bos = new BufferedOutputStream(
                       new FileOutputStream(file));
          String bhtml=""; //file content in byte array
```


Библиотека БГУИР

```
        while ((i = bis.read()) != -1) {
           bos.write(i);
           bhtml+=(char)i;
                                          }
           bos.flush();
           bis.close();
            String htmlcontent=
 (new String(bhtml)).toLowerCase(); //file content in string
            System.out.println("New      url      content      is:
"+htmlcontent);
        for (int j=0;j<keywords.length;j++)
        {

if(htmlcontent.indexOf(keywords[j].trim().toLowerCase())>=0)
            res++;
        }
      }
        catch (MalformedInputException malformedInputException)
         {
         System.out.println("error
"+malformedInputException.getMessage());
         return -1;
         }
         catch (IOException ioException)
         {
          System.out.println("error "+ioException.getMessage());
          return -1;
         }
                  catch(Exception e)
         {
            System.out.println("error "+e.getMessage());
            return -1;
         }
            return res;
  }

public static void main(String[] args)
{
 new Win_Lab();
 }
}
```

В конструкторе создается сама форма и ее визуальные компоненты (кнопки и текстовая область). Нас в первую очередь интересует кнопка поиска (`sea`). Ее обработчик такой:

```
    if (ae.getSource()==sea)
    {
         String [] keywords=txa.getText().split(",");
         for (int j=0;j<keywords.length;j++)
```


Библиотека БГУИР

```
        {
            System.out.println(keywords[j]);
        }
        File f = new File("e:/work5/Win_lab/src/source_html");
        ArrayList<File> files =
 new ArrayList<File>(Arrays.asList(f.listFiles()));
        txa.setText("");
        for (File elem : files)
        {
          int zcoincidence = test_url(elem,keywords);
          txa.append("\n"+elem+"  :"+zcoincidence);
        }
       }
```

Сначала формируем массив ключевых слов (которые должны разделяться запятыми):

```
String [] keywords=txa.getText().split(",");
```

Затем формируем список файлов в указываемом директории:

```
File f = new File("e:/work5/Win_lab/src/source_html");
      ArrayList<File> files =
new ArrayList<File>(Arrays.asList(f.listFiles()));
```

После этого для каждого файла в цикле получаем с помощью метода `test_url` число совпадений. Если представленную выше «заготовку» программы выполнить с ключевыми словами `students`, `like`, `travelling`, то получим следующий результат в окне программы (рисунок 100).

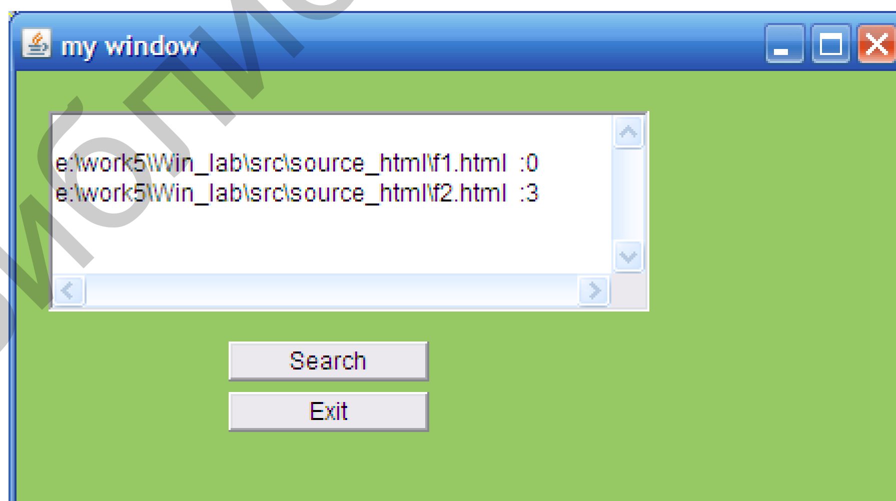


Рисунок 100 – Результат работы программы

Напротив имени каждого файла указано число совпадений с ключевыми словами.

Библиотека БГУИР

Задание

1 Подготовьте собственный набор `html`-документов (согласовать с преподавателем).

2 Завершите заготовку программы так, чтобы файл с наибольшим числом совпадений открывался браузером.

Контрольные вопросы

1 Что такое `URL`?
2 Как программно получить список всех файлов в папке?
3 Как программно запустить браузер и открыть `html`-файл?
5 Объясните использование потоковых классов в вашем приложении.

4.2 Работа в сети на основе сокетных соединений

Цель работы: познакомиться с техникой использования сокетных классов `Java`.

Краткое теоретическое содержание

В подразделе 2.1 представлены сведения по работе с сокетными классами на базе протоколов `TCP` и `UDP`. В разрабатываемом ниже приложении «клиент – сервер» используются два параллельно выполняющихся потока: **поток-сервер** и **поток-клиент**.

Для установления связи между клиентом и сервером нужно создать сокетное соединение как на серверной, так и на клиентской стороне. На стороне клиента это делается так:

```
class clientThread extends Thread{
  DataInputStream dis=null;
  Socket s=null;
 public clientThread()
   { try
    {
   s=new Socket("127.0.0.1",3001);
   .   .   .     и т. д.
```

Здесь объявлена переменная типа `Socket`, которая создается в команде

```
s=new Socket("127.0.0.1",3001);
```

Библиотека БГУИР

На стороне сервера указанные действия выполняются в главном методе потока `run`:

```
ServerSocket server;
String amountstring;
static  int amount=200;
public void run(){
      try       {
    server= new ServerSocket(2525);
    }
  catch(Exception e)
  { System.out.println("ERRSOCK+"+e);
  }
```

Подключение клиента к серверу:

```
s=server.accept();
```

В данной лабораторной работе сервер передает клиенту данные о банковском счете, представленные случайными целыми числами. На стороне сервера это делается таким образом:

```
class Account extends Thread{
     ServerSocket server;
     String amountstring;
     static  int amount=200;
     public void run(){
      try
         {
         server= new ServerSocket(3001); //Номер сокета
         }
       catch(Exception e)
       { System.out.println("Ошибка соединения+"+e);
       }
       while(true)
       { Socket s=null;
         try{
          s=server.accept(); //ожидание соединения с клиентом
            }
         catch(Exception e)
          {System.out.println("Ошибка"+e);}
          try
             {PrintStream ps=new PrintStream(s.getOutputStream());
//PrintStream предназначен для текстового вывода
             int amountcur=((int)(Math.random()*1000));
//отрицательный вклад –  снятие части денег со счета
              if (Math.random()>0.5)
```


Библиотека БГУИР

```
                amount-=amountcur;
                else
                amount+=amountcur;
                Integer x=new Integer(amount);
                amountstring=x.toString();
                ps.println("Account:"+amountstring);
           //передача строки клиенту
                ps.flush();
                s.close();//Сокетное соединение закрывается
}
catch(Exception e)
    {System.out.println("Ошибка  "+e);
     }
}
```

Поток вывода для сервера реализуется через переменную `ps`:

```
PrintStream
ps=new PrintStream(s.getOutputStream());
```

Объектная переменная `ps` предоставляет методы вывода, например

```
ps.println("Account:"+amountstring);
```

В команде

```
amount=((int)(Math.random()*1000));
```

на счет клиента добавляется случайная величина с помощью метода `random`, генерирующего случайные числа от `0` до `1`.

Изменение суммы в переменной `amount` выполняется путем внесения или снятия случайной величины со счета:

```
int amountcur=((int)(Math.random()*1000));
if (Math.random()>0.5)
amount-=amountcur;
else
amount+=amountcur;
```

Серверная часть реализована как приложение `Java` на основе формы с главным методом `main`:

```
public static void main(String args[])
    {
      serv f=new serv();
      f.resize(400,400);
      f.show();
      new Account().start();
  }
```

Базовым классом сервера является класс `serv`:

Библиотека БГУИР

```
 public class serv extends Frame{
      public boolean handleEvent(Event evt)//Используется обработчик
//событий ранних версий Java
    {
       if (evt.id==Event.WINDOW_DESTROY)//Закрыть приложение
         {System.exit(0);}
        return super.handleEvent(evt);
}
    public boolean mouseDown(Event evt,int x,int y)//Обработчик
//события от мыши
{
     new clientThread().start();//Запуск потока клиента
     return(true);
}
      public static void main(String args[])
    {
      serv f=new serv();
      f.resize(400,400);
      f.show();
      new Account().start();
   }
    }
```

Главный метод запускает поток-сервер

```
new Account().start();
```

Базовый класс приложения реализует обработку события закрытия окна:

```
public boolean handleEvent(Event evt)
    {
       if (evt.id==Event.WINDOW_DESTROY)
         {System.exit(0);}
        return super.handleEvent(evt);
     }
```

Тип события проверяется командой

```
if (evt.id==Event.WINDOW_DESTROY)
```

Второй метод

```
    public boolean mouseDown(Event evt,int x,int y)
{
     new clientThread().start();
     return(true);
}
```


Библиотека БГУИР

обрабатывает щелчок мышью в окне сервера (по щелчку в окне запускается клиент):

```
new clientThread().start();
```

И сервер, и клиент реализованы как отдельные потоки. Поток клиента такой:

```
class clientThread extends Thread{
 DataInputStream dis=null;
 Socket s=null;
 public clientThread()
   { try{
   s=new Socket("127.0.0.1",2525);
   dis= new DataInputStream(s.getInputStream());
    }
   catch(Exception e)
  {  System.out.println("Ошибка:  "+e);
  }}

 public void run()
{ while (true)
  {
     try
  {sleep(100);
   }
   catch(Exception er)
    {System.out.println("Ошибка  "+er);
     }
try{
    String msg=dis.readLine();
    if(msg==null)
        break;
        System.out.println(msg);
}
   catch(Exception e)
    {System.out.println("ERRORR+"+e);
    }
}
  }
    }
```

Клиент инициализируется таким образом:

```
 DataInputStream dis=null;
 Socket s=null;
```

```
 public clientThread()
   { try{
   s=new Socket("127.0.0.1",2525);
   dis= new DataInputStream(s.getInputStream());
}
```

Клиент пытается прочитать данные из сокета

`Socket("127.0.0.1",3001);`

Именно туда пишет данные сервер. Заметим, что клиент должен указать сетевой адрес компьютера, где расположен сервер (этот адрес называется `IP`-адресом). Клиент использует объектную переменную `dis` для чтения данных из сокета в методе `run`:

```
public void run()
{ while (true)
  {
     try
  {sleep(100); //клиент выполняет попытку чтения из сокета каждые
//100 миллисекунд
    }
   catch(Exception er)
    {System.out.println("Ошибка "+er);
     }
try{
    String msg=dis.readLine();// Клиент пытается прочитать строку
//из сокета
    if(msg==null)
         break;
         System.out.println(msg);   //прочитанная строка выводится
на //консоль
     }
```

Задание

1 Завершите приложение работоспособной программой, в которой используются описанные поток-сервер и поток-клиент.

2 Создайте свой вариант программы на основании следующих данных.

2.1 Напишите приложение для двух клиентов, которые работают с одним счетом. Каждому клиенту соответствует свой поток.

2.2 Создайте приложение «клиент – сервер» с некоторым визуальным интерфейсом, который выполняет запуск клиента по нажатии кнопки и отражает состояние счета в текстовом поле.

Библиотека БГУИР

Контрольные вопросы

1 Объясните назначение протоколов TCP, UDP. В чем их отличие?
2 С помощью каких объектов и каких методов данные отсылаются по сети?
3 Как реализовать пересылку данных в обратном направлении (от клиента к серверу)?
4 Как определить сетевой адрес вашего компьютера.

4.3 Работа с сервлетами

Цель работы: познакомиться с техникой использования сервлетов – классов Java на стороне сервера.

Краткое теоретическое содержание

В подразделе 2.5 представлены сведения по работе с сервлетами. Нас интересует поиск различных информационных объектов на стороне сервера. Клиентская часть обычно представляется сайтом (документом Html). В качестве примера возьмем следующий сайт:

```
<!DOCTYPE html>
<html>
      <head>
        <title></title>
        <meta    http-equiv="Content-Type"    content="text/html;
charset=UTF-8">
    </head>
    <body bgcolor="#aaccff">
        <Font color="green" size="10">
         Форма для работы со словарем
        </Font>
        <br>
        <br>
        <form name="frm" method="Get" action="MyServlet">

            <Font    color="blue"    size="6">    Введите    русское
слово:</Font><Input type="Text" name="txt" value=""/>
            <br>
            <br>
 <Font  color="blue"  size="6">Перевод:  </Font><input  type="text"
name ="trans" value="     "/><br>
        <h4>Кликни   здесь   для   получения   перевода   :<Input
type="submit" value="Перевести"/>
        </h4>
            </form>
```


Библиотека БГУИР

```
</body>
```

Отметим, что воспроизведение русского текста обеспечивается строкой `<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">`.

В тексте сайта выполняется вызов сервлета с именем `MyServlet`. Это имя указывается в атрибуте `action="MyServlet"`. Вызов сервлета выполняется по нажатию кнопки с типом `type="submit"`. При открытии сайт выглядит следующим образом (рисунок 101).

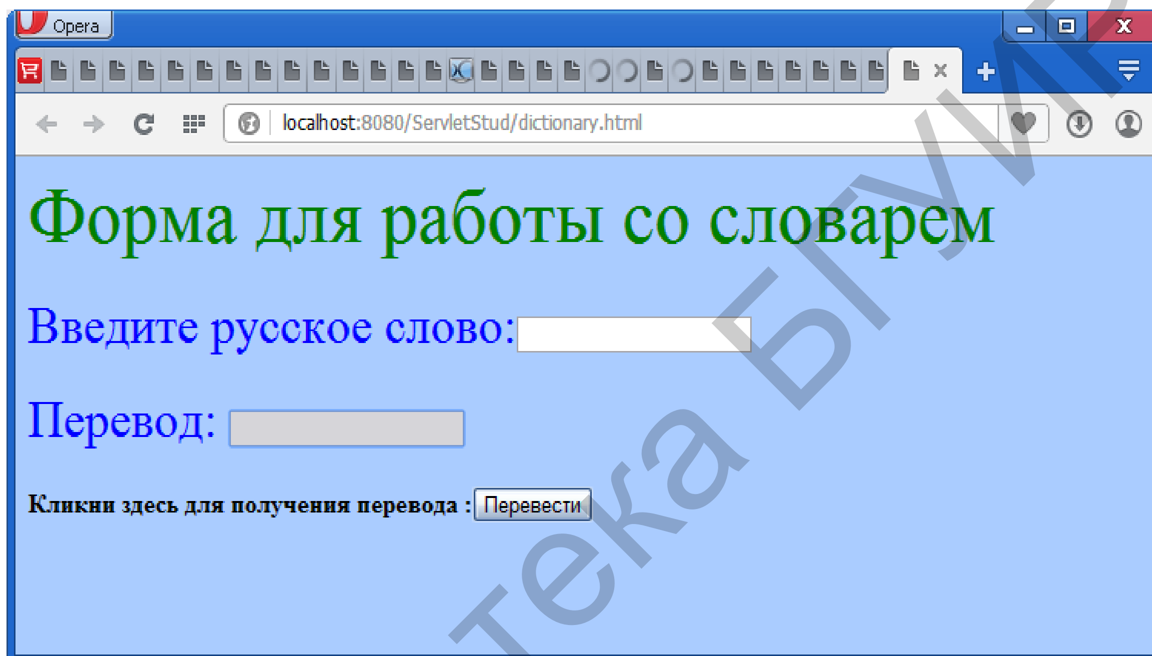


Рисунок 101 – Страница сайта

Нужно создать сервлет, обрабатывающий данную форму. Для построения приложения подобного типа следует использовать шаблон `Java Web`. Структура такого проекта состоит из следующих узлов (пример на рисунке 102).

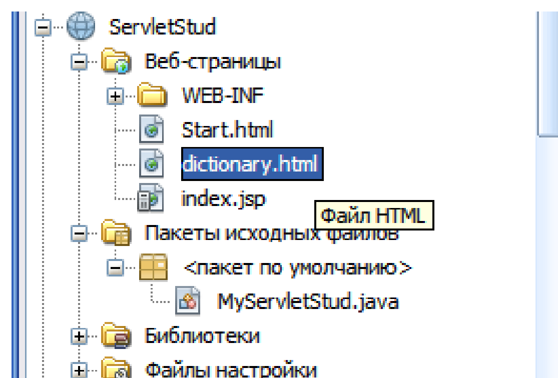


Рисунок 102 – Структура проекта для работы со словарем

Библиотека БГУИР

В узле WEB-INF помещают файлы jsp и html. В узле Пакеты исходных файлов помещают сервлеты и классы Java. В узел Библиотеки добавляют jar-файлы и библиотеки с требуемыми классами Java (если необходимо). Нам также понадобится изменить конфигурационный файл в узле Файлы и настройки.

Будем считать, что документ dictionary.html (текст которого приведен ранее) создан. Создаем теперь класс сервлета. Для этого открываем контекстное меню щелчком правой кнопки мыши на узле <пакет по умолчанию>, выбираем пункт Создать и опцию Сервлет. Указваем имя класса сервлета.

Класс сервлета содержит два важнейших метода – doGet и doPost. Оба предназначены для обработки данных от клиентской формы. Эти методы вызываются в зависимости от атрибута method тега form документа html:

```
<form name="frm" method="Get" action="MyServlet">
```

В методе Get данные клиентской формы передаются одним сообщением вместе со служебной информацией (заголовком). В методе Post данные разбиваются на несколько пакетов. Оба этих метода «перенаправлены» на один метод processRequest. Именно в последнем методе мы и «сосредоточим» всю необходимую обработку:

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException
{
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        String rus_word=""+request.getParameter("txt");

...........
```

Здесь приведена стартовая часть метода processRequest. У метода есть два встроенных объекта – request,response, с помощью которых можно получить значения элементов формы клиентского сайта и направить их обратно в клиентский сайт. Видим, как читается значение параметра

```
String rus_word=""+request.getParameter("txt");
```

Здесь txt – это имя элемента клиентской формы:

```
<Input type="Text" name="txt" value=""/>
```

Отправка значений обратно клиенту поясняется следующим фрагментом:

```
PrintWriter out = response.getWriter();
```

Библиотека БГУИР

```
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet MyServletStud</title>");
            out.println("</head>");
            out.println("<body bgcolor='#aaccff'");
            out.println("<form>");
            out.println("<h2> Привет клиенту!!!</h2><br><br>");
            out.println("</form>");
            out.println("</body>");
            out.println("</html>");
  .........
```

Создаем потоковый объект `PrintWriter out` и используем его метод `println`.

Теперь скорректируем конфигурационный файл `web.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <servlet>
        <servlet-name>MyServlet</servlet-name>
        <servlet-class>MyServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>MyServlet</servlet-name>
        <url-pattern>/MyServlet</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>
            Dictionary.html
         </welcome-file>
     </welcome-file-list>
     <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
</web-app>
```

Чтобы этот файл появился в проекте, нужно открыть контекстное меню на имени проекта и выполнить последовательно пункты `Очистить и Построить` и `Развернуть`. В документе прописан один или несколько классов сервлетов, а в тегах `<welcome-file>` указан стартовый сайт приложения (с которого данное приложение запускается).

Библиотека БГУИР

Задание

1 Завершите приложение работоспособной программой, в которой используется база данных, содержащая переводы слов.
2 Добавьте в программу обратный перевод

Контрольные вопросы

1 Что такое сервлет, какие основные объекты и методы он предоставляет?
2 Как сервлет определяется в клиентском сайте?
3 Для чего служит конфигурационный файл web.xml? Поясните его структуру.
4 Как вернуть результаты, полученные в сервлете, обратно на сторону клиента?

4.4 Технология JSF

Цель работы: познакомиться с технологией JSF (Java Server Faces).

Краткое теоретическое содержание

В пункте 2.3.1 представлены сведения по работе с JSF. Создаем web-приложение на базе JSF. Назовем его JSF_LAB. Шаги по созданию этого приложения мы опускаем (см. лекционную часть данного учебно-методического пособия). Добавляем в web-приложение managed bean (управляемый компонент). С этой целью щелкаем правой кнопкой мыши на узле Пакеты исходных файлов дерева проекта и выбираем пункты Создать и Управляемый компонент JSF. Изменяем его текст следующим образом:

```
package com;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
@ManagedBean
@RequestScoped
public class mancomp {
    /**
     * Creates a new instance of mancomp
     */
       private String input;
       private String output;

    public mancomp() {
```


Библиотека БГУИР

```
    };

        public String getAnswer(String x) {
        return "Hello from JSF "+x;
                                      }
     public void submit() {
    // handle form submission
    output = "You Are Welcome!!!"+input;
  }

  public String getInput() {
    return input;
  }

  public void setInput(String input) {
    this.input = input;
  }

  public String getOutput() {
    return output;
  }

  public void setOutput(String output) {
    this.output = output;
  }
}
```

Видим, что бин дает доступ к скрытым членам с именами `input`, `output`.

В файле `index.xhtml` пишем

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        <h2> Current Lesson on JFACE</h2>
<h:form>
     <h:graphicImage value="my.png"/>
     <br>
    <h:inputText value="#{mancomp.input}" />
     </br>
     <br>
    <h:commandButton value="Submit" action="#{mancomp.submit}">
    </h:commandButton>
     </br>
     <br> <h:outputText value="#{mancomp.output}" />
```


Библиотека БГУИР

```
</br> </h:form>
</h:body>
</html>
```

Теперь у нас есть:

- картинка:

```
<h:graphicImage value="my.png"/>
```

- входное текстовое поле:

```
<h:inputText value="#{mancomp.input}" />
```

- выходное текстовое поле:

```
<h:outputText value="#{mancomp.output}" />
```

- кнопка:

```
<h:commandButton value="Submit"
action="#{mancomp.submit}">
</h:commandButton>.
```

Запустим проект на выполнение. Результат показан на рисунке 103.

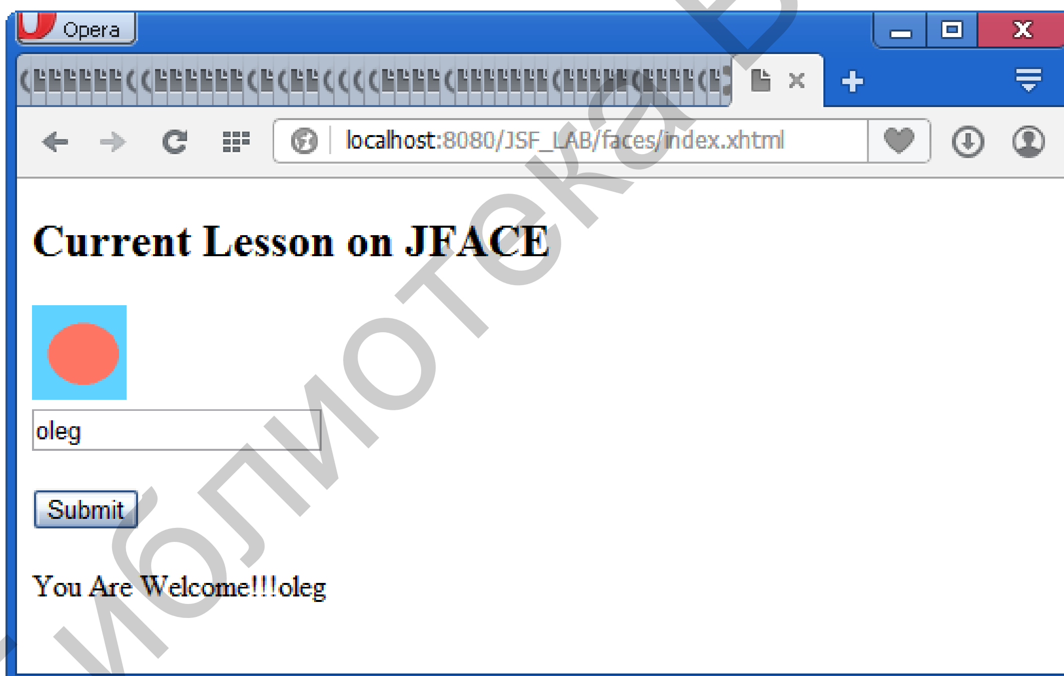


Рисунок 103 – Скриншот с окном приложения JSF

Обратимся теперь к работе с таблицей данных. Мы хотим получить приложение, отображающее таблицу с данными, как показано на рисунке 105. Структура нашего проекта должна иметь вид в соответствии с рисунком 104.

Библиотека БГУИР

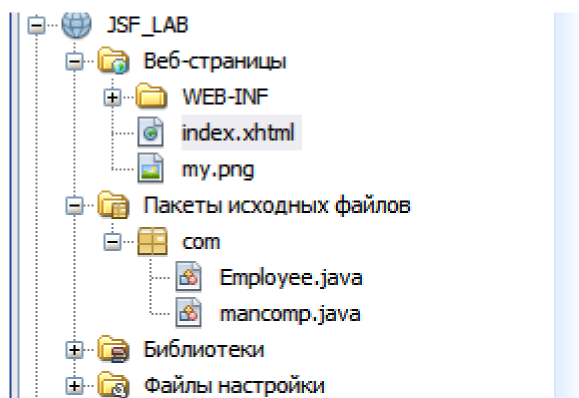


Рисунок 104 – Структура проекта на платформе `JFace`

Нам понадобится персистентный класс `Employee.java` для хранения структуры базы данных (см. рисунок 105).

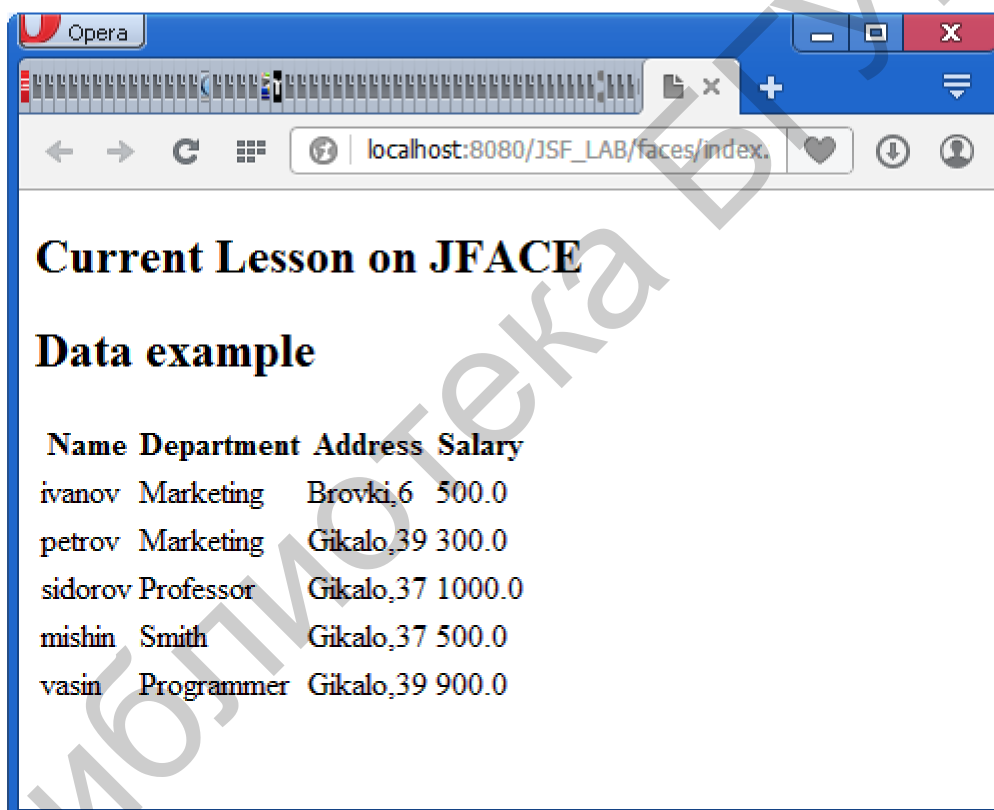


Рисунок 105 – Вывод информации из персистентного класса

Текст класса `Employee.java` такой:

```
package com;

public class Employee {
   private String name;
   private String department;
   private String address;
   private double wage;
```

Библиотека БГУИР

```
   private boolean canEdit;

   public   Employee   (String   name,   String   department,String
address,double wage){
       this.name = name;
       this.department = department;
       this.address = address;
       this.wage = wage;
       canEdit = false;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getDepartment() {
      return department;
   }

   public void setDepartment(String department) {
      this.department = department;
   }

   public String getAddress() {
      return address;
   }

   public void setAge(String address) {
      this.address = address;
   }

   public double getWage() {
      return wage;
   }

   public void setWage(double wage) {
      this.wage = wage;
   }

   public boolean isCanEdit() {
      return canEdit;
   }

   public void setCanEdit(boolean canEdit) {
      this.canEdit = canEdit;
   }
}
```

Библиотека БГУИР

Теперь перепишем боб следующим образом:

```
package com;

import javax.ejb.Stateless;
import java.util.*;
import java.io.Serializable;

@Stateless
public class EJB_LAB implements EJB_LABRemote, Serializable {

   private static final long serialVersionUID = 1L;

   private String fio;
   private String department;
   private String group;


   private static final ArrayList<Employee> employees
      = new ArrayList<Employee>(Arrays.asList(
      new Employee("ivanov", "Marketing", "10"),
      new Employee("petrov", "Marketing", "10"),
      new Employee("sidorov", "Professor", "12"),
      new Employee("mishin", "Smith", "12"),
      new Employee("vasin", "Programmer", "14")
   ));


    @Override
    public String getEmployeeInfo(String fio) {
        return null;
    }

   public String addEmployee() {
      Employee std =
  new Employee(fio,department,group);
      employees.add(std);
      return null;
   }

   public String deleteEmployee(Student em) {
      employees.remove(em);
      return null;
   }

     public String getFio() {
      return fio;
   }

   public void setName(String fio) {
      this.fio = fio;
```

Библиотека БГУИР

```
    }

    public String getDepartment() {
       return department;
    }

    public void setDepartment(String department) {
       this.department = department;
    }

    public String getGroup() {
       return group;
    }

    public void setGroup(String group) {
       this.group = group;
    }
}
```

В этом классе описаны поля таблицы

```
    private String fio;
    private String department;
    private String group;
```

и методы доступа к ним (set/get).

Смысл помещенного выше текста легко понять из описания управляемого бина (managed bean) в лекционном курсе. Данный бин содержит все необходимые методы для работы с таблицей из сайта. Текст сайта теперь имеет такой вид:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      >
    <h:head>
        <title>Facelet Title</title>
     </h:head>
     <h:body>
         <h2> Current Lesson on JFACE</h2>

  <h2>Data example</h2>
    <h:form>
       <h:dataTable value="#{mancomp.employees}" var="employee">

          <h:column>
```

```
            <f:facet name="header">Name</f:facet>
            #{employee.name}
         </h:column>
         <h:column>
            <f:facet name="header">Department</f:facet>
            #{employee.department}
         </h:column>
         <h:column>
            <f:facet name="header">Address</f:facet>
            #{employee.address}
         </h:column>
         <h:column>
            <f:facet name="header">Wage</f:facet>
            #{employee.wage}
         </h:column>
      </h:dataTable>
    </h:form>
    </h:body>
</html>
```

Таблица вставляется таким образом:

```
<h:dataTable value="#{mancomp.employees}" var="employee">
```

Данные в таблицу поставляются из коллекции

```
ArrayList<Employee> employees.
```

Каждый столбец таблицы представлен заголовком, например:

```
<f:facet name="header">Address</f:facet>
```

и содержимым:

```
<h:column>
   #{employee.address}
</h:column>.
```

Задание

Завершите приложение работоспособной программой, в которую следует добавить текстовые поля и кнопки для ввода новых значений в таблицу `employee` и удаления записей из таблицы. Если работа выполнена досрочно, то реализуйте дополнительно поиск информации по фамилии работника.

Контрольные вопросы

1 Что такое управляемый компонент (бин)? Как он связывается с главным сайтом (`jsp`)?
2 Объясните структуру сайта `index.jsp` и назначение использованных в нем элементов.

Библиотека БГУИР

3 Каким образом выполняется привязка полей класса к столбцам таблицы?

4.5 Создание web-сервисов

Цель работы: познакомиться с технологией web-сервисов.

Краткое теоретическое содержание

В подразделе 3.2 представлены сведения по работе с web-сервисами. Создадим web-сервис для скачивания файлов (file-upload). Сначала создаем web-проект (имя WebSErviceLab), структура которого показана на рисунке 106.

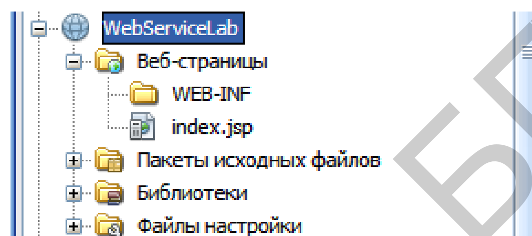


Рисунок 106 – Структура проекта web-сервиса

Данный проект пуст. В него следует добавить web-службу. Для этого активизируем мастера через контекстное меню на имени проекта и выберем пункт Создать – Другое - web-службы (рисунок 107).

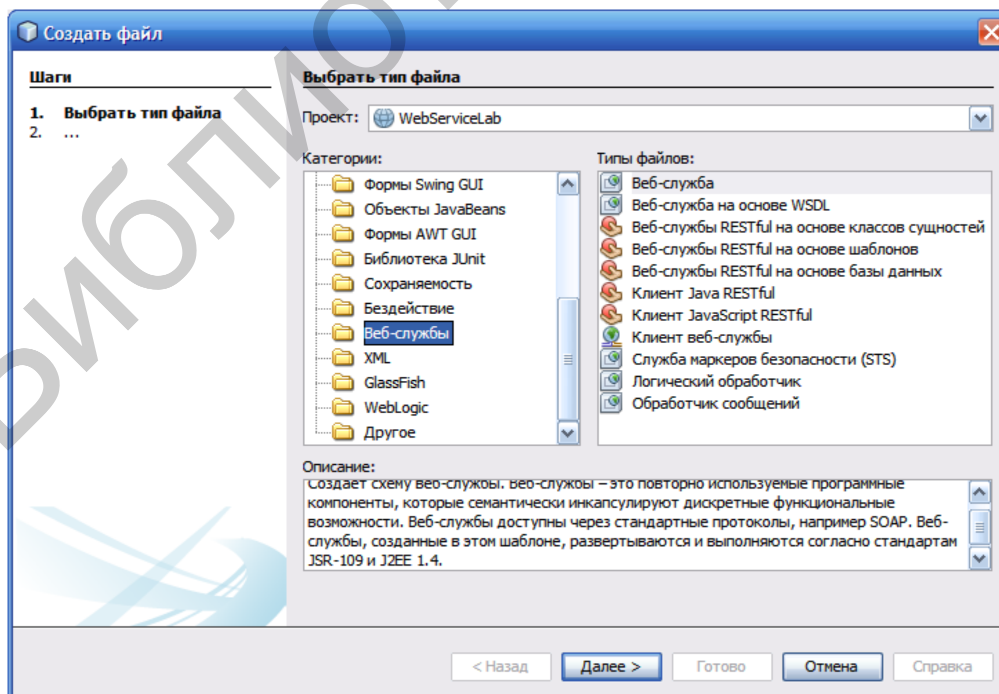


Рисунок 107 – Добавление web-службы в дерево проекта с помощью мастера

Библиотека БГУИР

Выбираем тип файла web-служба и нажимаем Далее. Вводим имя службы и имя пакета (рисунок 108).

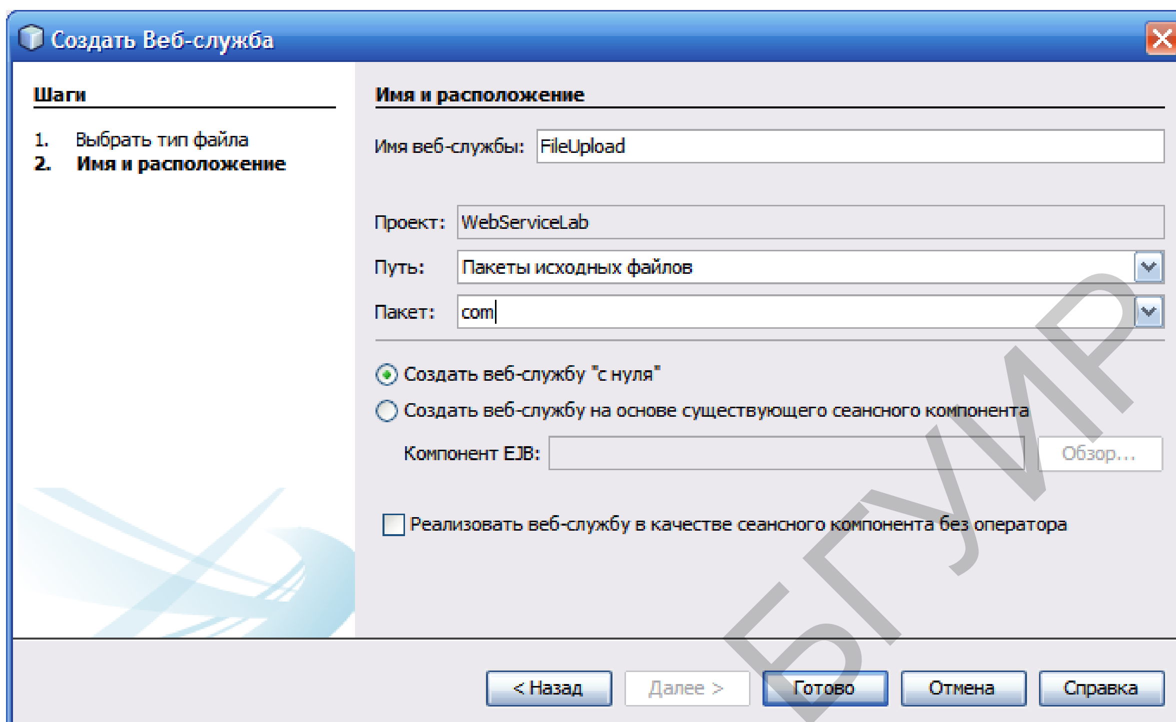


Рисунок 108 – Задание имени службы и имени пакета

Нажимаем Готово. Система создает следующую заготовку сервиса:

```
package com;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;


@WebService(serviceName = "FileUpload")
public class FileUpload {


    @WebMethod(operationName = "hello")
    public String hello(@WebParam(name = "name") String txt) {
        return "Hello " + txt + " !";
    }
}
```

Для начала добавим в службу свой собственный метод. Щелкаем правой кнопкой мыши в окне кода и выбираем пункт Вставка кода из контекстного меню (рисунок 109).

Библиотека БГУИР

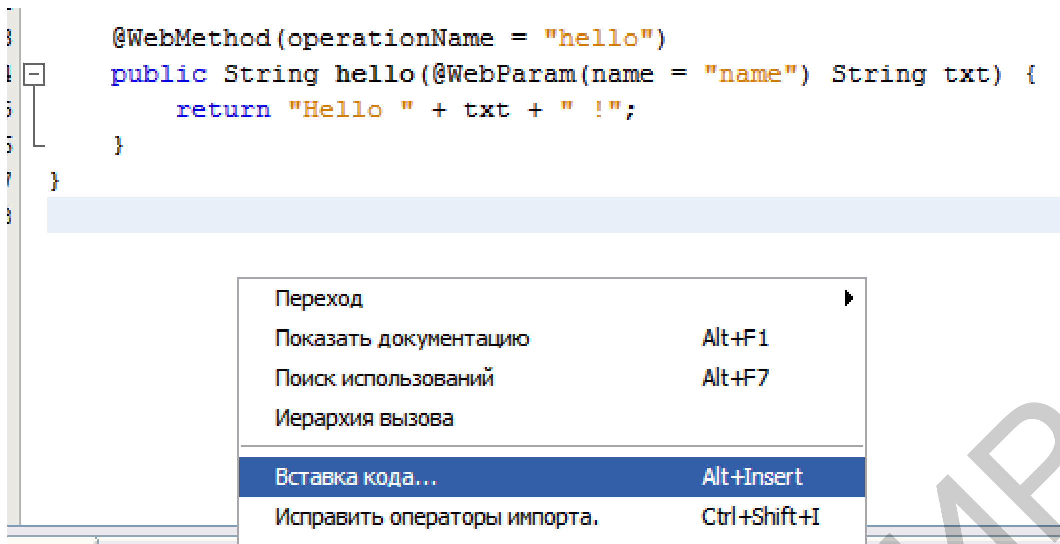


Рисунок 109 – Добавление метода в web-службу

На экране появится следующее окно (рисунок 110).

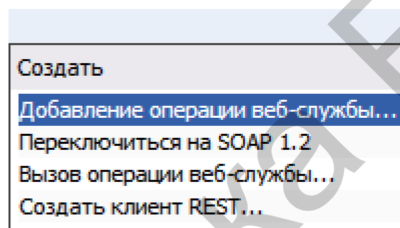


Рисунок 110 – Контекстное меню для добавления метода

Выбираем пункт `Добавление операции web-службы` (рисунок 111).

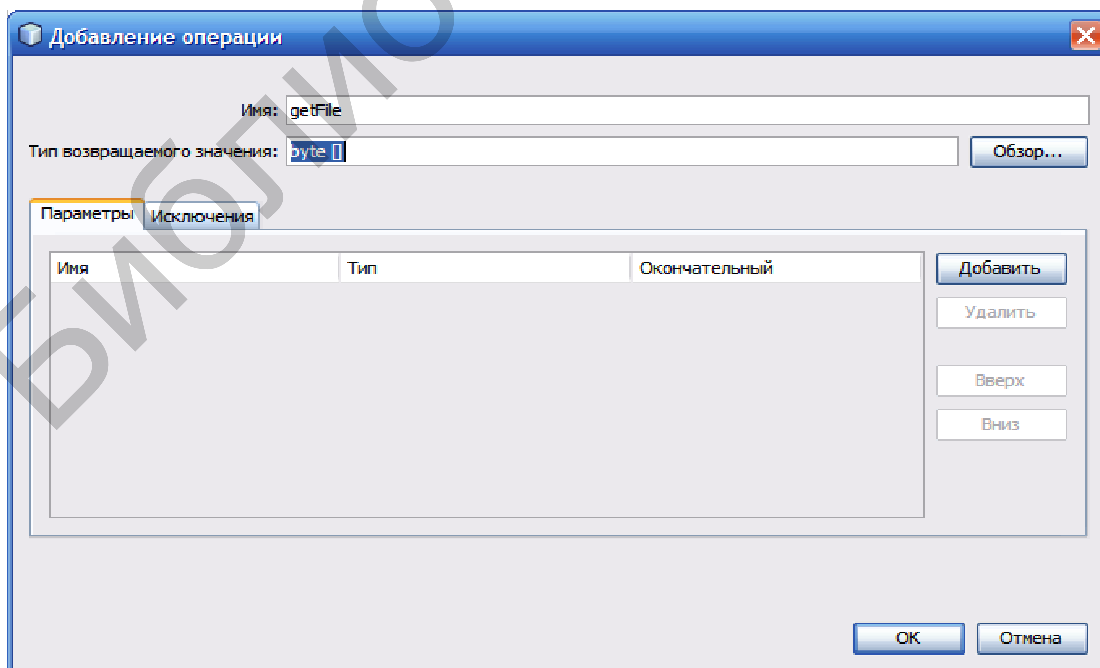


Рисунок 111– Определение имени и параметров метода

Библиотека БГУИР

Задаем имя метода `getFile` и тип возвращаемого значения – массив байтов. В результате код в окне редактора принял такой вид:

```
package com;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;


@WebService(serviceName = "FileUpload")
public class FileUpload {

    @WebMethod(operationName = "hello")
    public String hello(@WebParam(name = "name") String txt) {
        return "Hello " + txt + " !";
    }

       @WebMethod(operationName = "getFile")
    public byte [] getFile() {
        //TODO write your implementation code here:
        return null;
    }
}
```

Наша задача – реализовать метод `getFile`. Мы намереваемся пока просто передать текстовый файл, конверсированный в массив байтов, клиенту. Причем отсылаем массив байтов, в который этот файл преобразуем. Текстовый файл заранее определен. Код нашего метода примет такой вид:

```
package com;
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import java.io.*;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.nio.file.Path;


@WebService(serviceName = "FileUpload")
public class FileUpload {

    @WebMethod(operationName = "hello")
    public String hello(@WebParam(name = "name") String txt) {
        return "Hello " + txt + " !";
    }

       @WebMethod(operationName = "getFile")
  public byte [] getFile() {
```


Библиотека БГУИР

```
        byte [] buf=null;
        java.awt.FileDialog fd=null;
        java.awt.Dialog di=null;

        try
        {
          String s1="e:/work5/hib1.txt";
          Path path = Paths.get(s1);
          buf = Files.readAllBytes(path);
        }

        catch(Exception ex)
        {
        }
        return buf;
    }}
```

Выполним сервис и разместим его (`deploy`), используя опции меню. Сначала из контекстного меню выберем `Очистить и построить`, затем – `Развернуть`.

Серверная часть готова.

Создаем клиент как обычное `Java`-приложение (`Java Application` с именем `WebServiceClient`). Вставляем в проект `web`-клиент (приводимая далее последовательность скриншотов показывает, как это делается – рисунки 112–114).

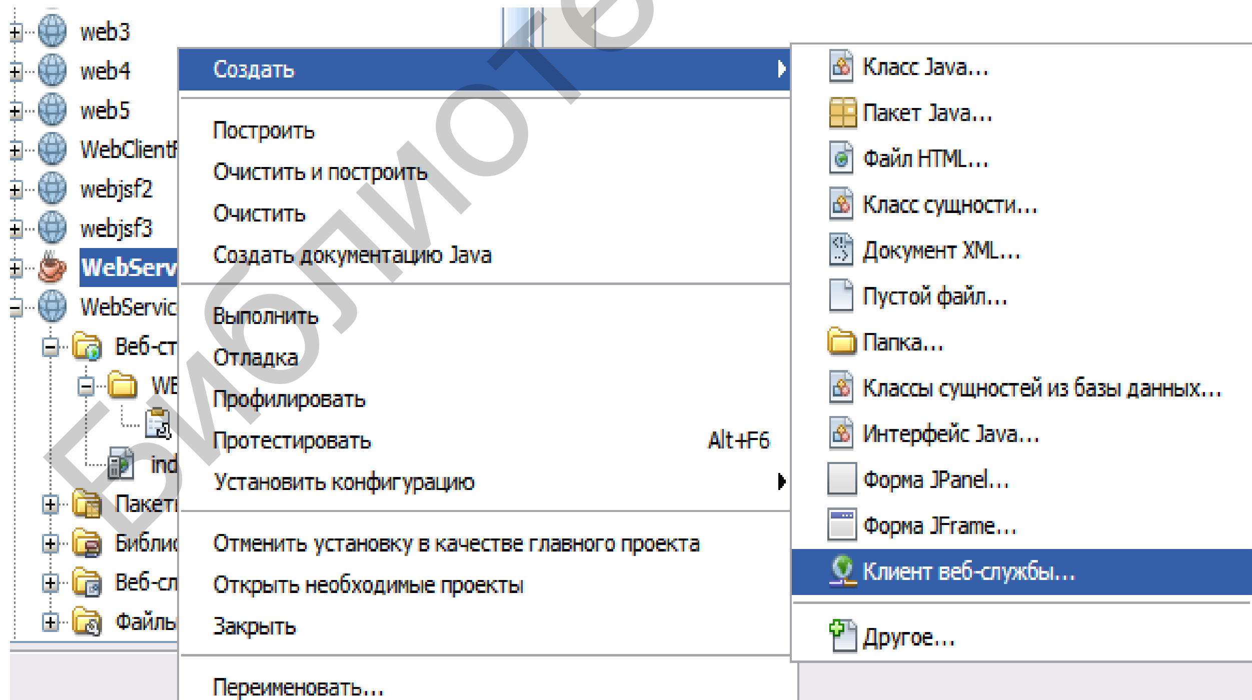


Рисунок 112 – Создание клиента `web`-службы

Библиотека БГУИР

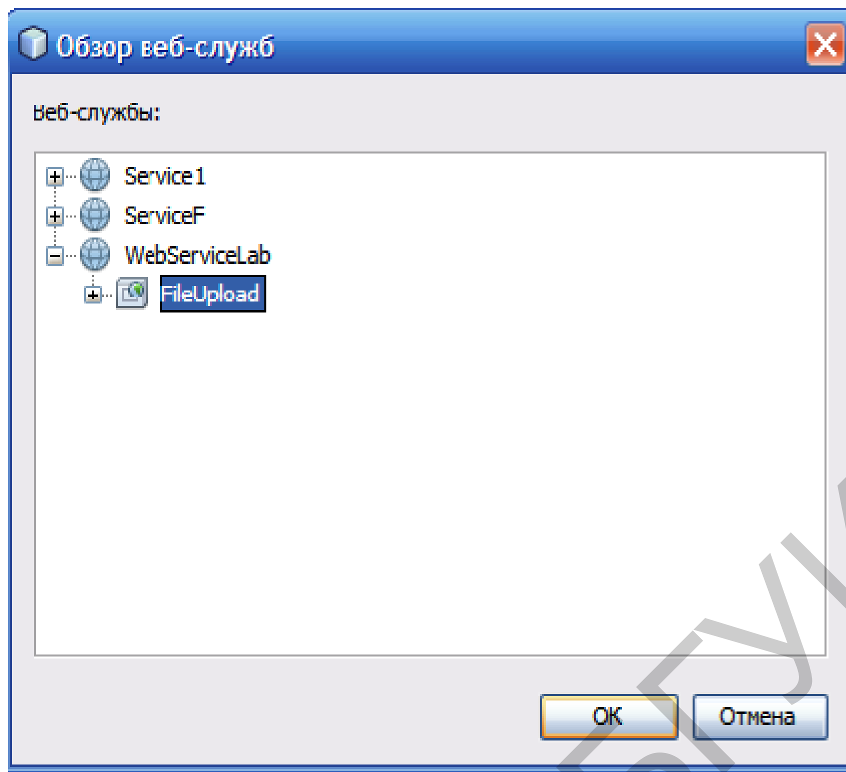


Рисунок 113 – Добавление класса web-сервиса

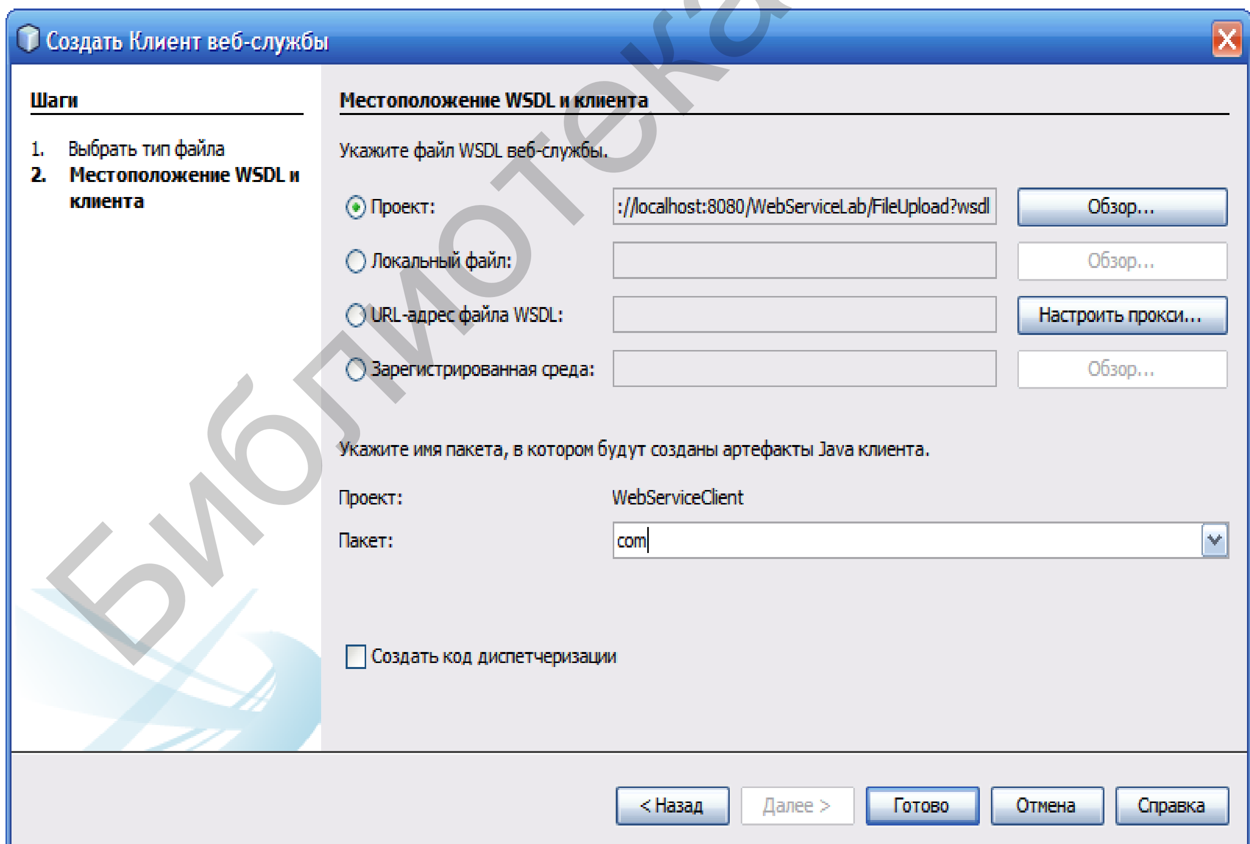


Рисунок 114 – Ссылка на проект web-сервиса

Библиотека БГУИР

Структура проекта (с серверной и клиентской частями) показана на рисунке 115.

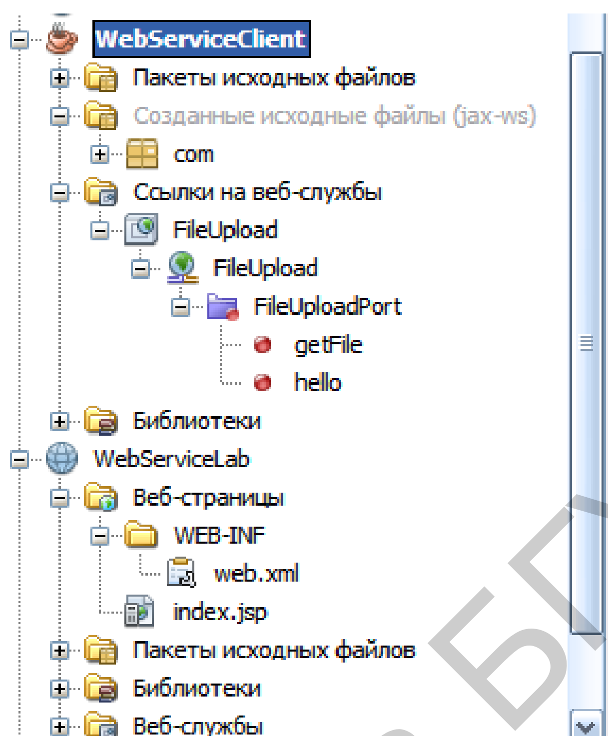


Рисунок 115 – Структура проекта с серверной и клиентской частями

Программируем клиентскую часть таким образом:

```
package webserviceclient;
import com.FileUpload;
import com.FileUpload_Service;

public class WebServiceClient {

    public static void main(String[] args) {
               FileUpload_Service service =
 new FileUpload_Service();

System.out.println(service.getFileUploadPort().hello
("OLEG GERMAN"));
              try
              {
               byte [] fromserv
 = service.getFileUploadPort().getFile();
               if((fromserv==null)||(fromserv.length<=0))
               {
                   System.out.println("File was not sent");
               }
               else
               {
```

Библиотека БГУИР

```
                    String str = new String(fromserv);
                    System.out.println(str);

                 }

                }
                catch(Exception ex)
                {
                    System.out.println
("Cannot get file from service:"+ex.getMessage());
        }
    }
}
```

Файл считывается в следующем фрагменте:

```
   byte [] fromserv
 = service.getFileUploadPort().getFile();
```

Результат работы показан ниже:

```
Hello OLEG GERMAN !
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE   hibernate-mapping   PUBLIC   "-//Hibernate/Hibernate
Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="hiberjava">
    <class name="stud" table="mytable2">
       <id name="id" column="p_key">
            <generator class="native"/>
        </id>

        <property name="name" column="fio"/>
        <property name="groupp" column="groupp"/>
    </class>
</hibernate-mapping>
ПОСТРОЕНИЕ УСПЕШНО ЗАВЕРШЕНО (общее время: 14 секунд)
```

Задание

Усложните приведенное приложение. Клиенту web-сервис должен передать список всех текстовых файлов (для простоты – txt), находящихся в электронной библиотеке. Этот список следует раскрыть в окне клиента. Клиент должен произвести выбор книги, а затем повторно обратиться к сервису за самой книгой. Ясно, что приложение клиента следует выполнить как оконное.

Библиотека БГУИР

Контрольные вопросы

1 Что такое управляемый web-сервис, как он связывается с клиентом?
2 Расскажите, как создать web-сервис.
3 Расскажите, как создать службу клиента.

4.6 Создание EJB-компонентов

Цель работы: познакомиться с технологией EJB.

Краткое теоретическое содержание

В подразделе 3.3 представлены сведения по работе с сеансовыми компонентами EJB. В данной работе мы должны развить навыки по использованию компонентной технологии.

Создадим сначала библиотечный класс с именем EJB_LAB_LIB (пока интерфейс пуст и сама библиотека пуста – будет использоваться как контейнер боба). Следующим шагом является создание модуля (класса) боба. Для этого создаем новый проект и выбираем категорию проекта Java EE (Модуль EJB). Введем имя проекта EJB_LAB (рисунок 116).

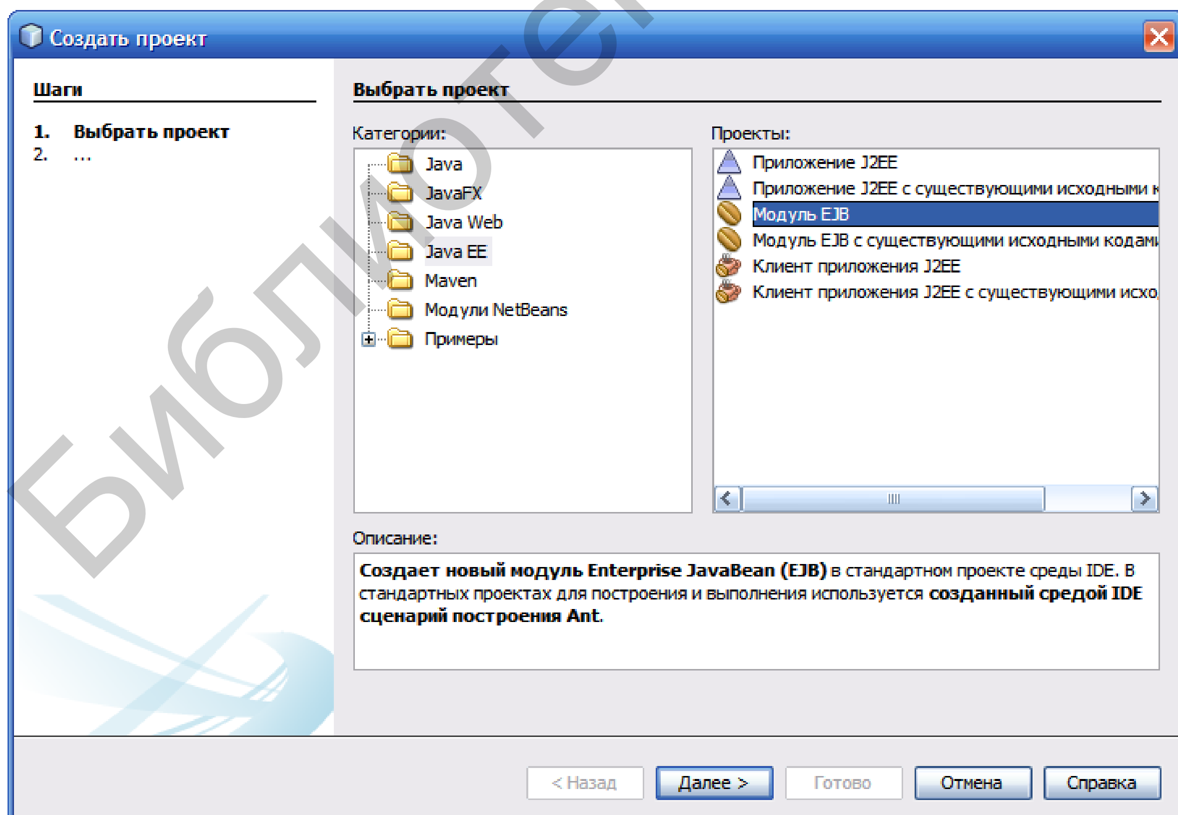


Рисунок 116 – Создание модуля EJB

Библиотека БГУИР

Создаем сеансовый компонент (рисунок 117):

– имя пакета – `com` (имя пакета произвольное);

– тип сеанса – без сохранения состояния (`Stateless`);

– тип интерфейса – удаленный интерфейс в проекте (в качестве имени проекта выбираем из списка ранее созданный пустой библиотечный класс).

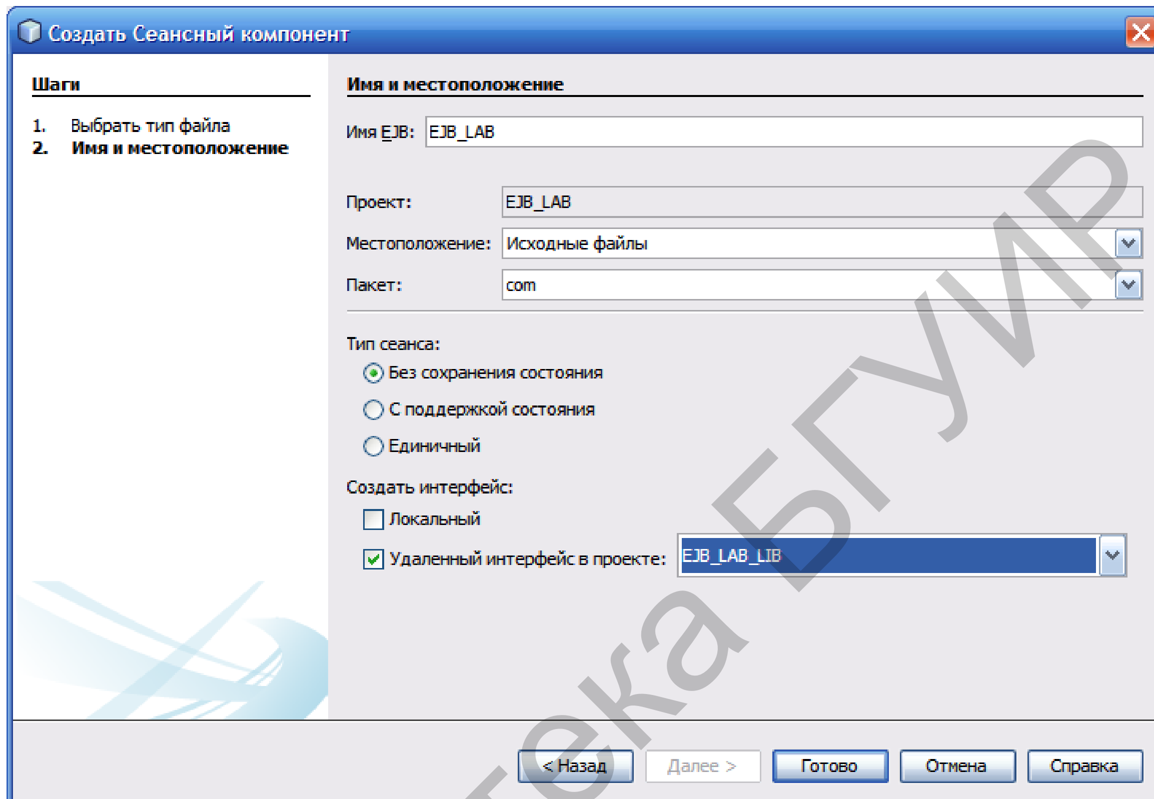


Рисунок 117 – Создание сеансового компонента

Нажимаем `Готово`.

Создается следующая заготовка кода сеансового компонента:

```
package com;

import javax.ejb.Stateless;
@Stateless
public class EJB_LAB implements EJB_LABRemote
 {
 }
```

В настоящий момент мы имеем класс сеансового компонента и пустой интерфейс в библиотечном архиве. Нам нужно добавить методы в компонент – они называются бизнес-методами. Для добавления бизнес-метода активизируем контекстное меню щелчком правой кнопки мыши в окне редактора кода компонента. Выбираем пункт `Вставка кода`. Затем выбираем пункт `Бизнес-метод`. Мы намереваемся работать с

Библиотека БГУИР

персистентным классом. Пока ограничимся «заготовкой» бизнес-метода, цель которого – получить информацию о студенте (рисунок 118).

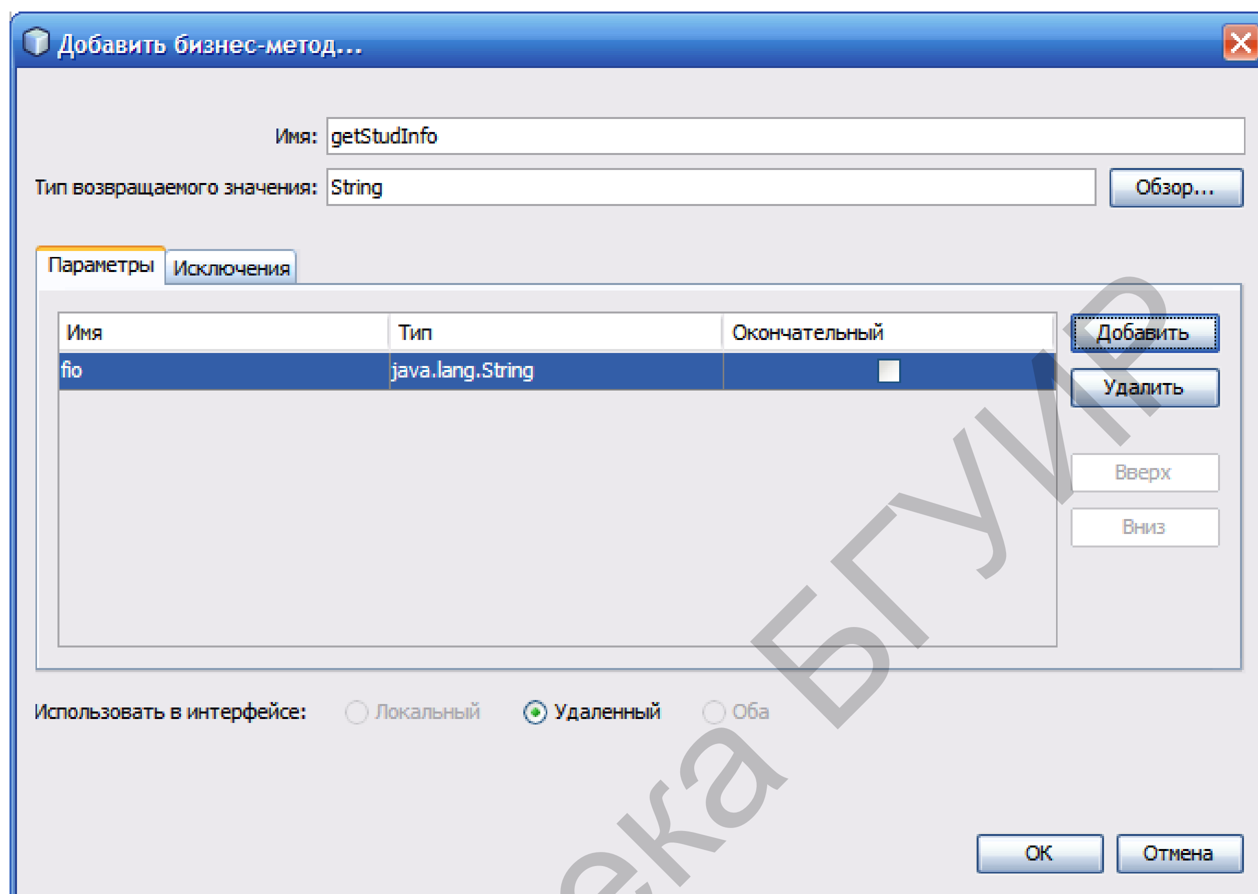


Рисунок 118 – Задание параметров бизнес-метода для получения информации о студенте

```
package com;
import javax.ejb.Stateless;
@Stateless
public class EJB_LAB implements EJB_LABRemote {

    @Override
    public String getStudInfo(String fio) {
        return null;
    }
}
```

Система автоматически добавила в интерфейс библиотеки наш бизнес-метод:

```
package com;

import javax.ejb.Remote;

@Remote
public interface EJB_LABRemote {
```


Библиотека БГУИР

```
    String getStudInfo(String fio);

}
```

Создаем персистентный класс

```
package com;

public class Student {
   private String fio;
   private String department;
   private String group;


   public Student (String fio, String department,String group){
      this.fio = fio;
      this.department = department;
      this.group = group;

   }

   public String getFio() {
      return fio;
   }

   public void setFio(String name) {
      this.fio = fio;
   }

   public String getDepartment() {
      return department;
   }

   public void setDepartment(String department) {
      this.department = department;
   }

   public String getGroup() {
      return group;
   }

   public void setGroup(String group) {
      this.group = group;
   }
   }
```

Изменяем (расширяем) класс боба следующим образом:

```
package com;

import javax.ejb.Stateless;
```


Библиотека БГУИР

```
import java.util.*;
import java.io.Serializable;

@Stateless
public class EJB_LAB implements EJB_LABRemote, Serializable {

   private static final long serialVersionUID = 1L;

   private String fio;
   private String department;
   private String group;


   private static final ArrayList<Student> students
      = new ArrayList<Student>(Arrays.asList(
      new Student("ivanov", "Marketing", "10"),
      new Student("petrov", "Marketing", "10"),
      new Student("sidorov", "Professor", "12"),
      new Student("mishin", "Smith", "12"),
      new Student("vasin", "Programmer", "14")
   ));


    @Override
    public String getStudInfo(String fio) {
     String rez="???";
         for (Iterator<Student> it =
             students.iterator(); it.hasNext();) {
            Student st=it.next();
            String name=st.getFio();
            if(name.equals(fio))
            {
             rez=st.getDepartment()+"  "+st.getGroup();
             break;
            }
        }

        return rez;    }

   public String addStudent() {
      Student std =
  new Student(fio,department,group);
      students.add(std);
      return null;
   }

   public String deleteStud(Student em) {
      students.remove(em);
      return null;
   }
```


Библиотека БГУИР

```
    public String getFio() {
        return fio;
    }

    public void setName(String fio) {
        this.fio = fio;
    }

    public String getDepartment() {
        return department;
    }

    public void setDepartment(String department) {
        this.department = department;
    }

    public String getGroup() {
        return group;
    }

    public void setGroup(String group) {
        this.group = group;
    }
}
```

Теперь корпоративное приложение можно построить и запустить. При запуске приложения среда IDE развернет архив EAR на сервере.

Щелкните правой кнопкой мыши на корпоративном приложении EJB_LAB и выберите Deploy (Развернуть). Структура нашего проекта представлена на рисунке 119.

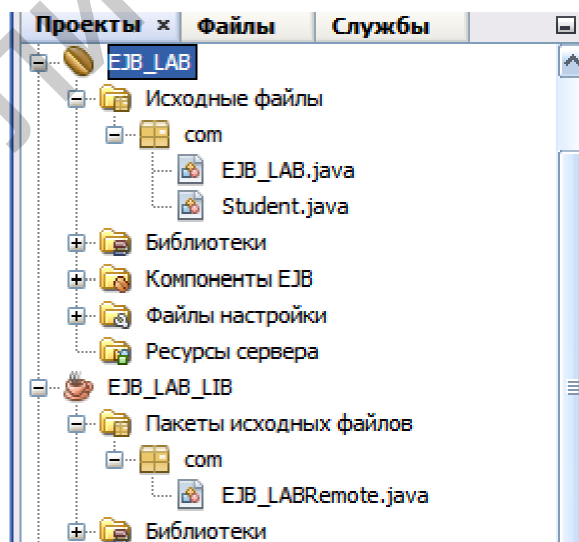


Рисунок 119 – Структура проекта на основе компонента EJB

Библиотека БГУИР

После выбора Развернуть среда IDE собирает корпоративное приложение и разворачивает архив EAR на сервере.

Создаем клиентское приложение. Выбираем Файл > Создать проект и затем Клиент корпоративного приложения в категории Java EE. Вводим имя клиентского приложения EJB_LAB_CLIENT в поле Project Name (Имя проекта). Выбираем GlassFish Server в качестве сервера.

Теперь следует добавить библиотеку классов, содержащую удаленный интерфейс. Выберите узел Библиотеки, откройте пункт Добавить проект в контекстном меню (рисунок 120).

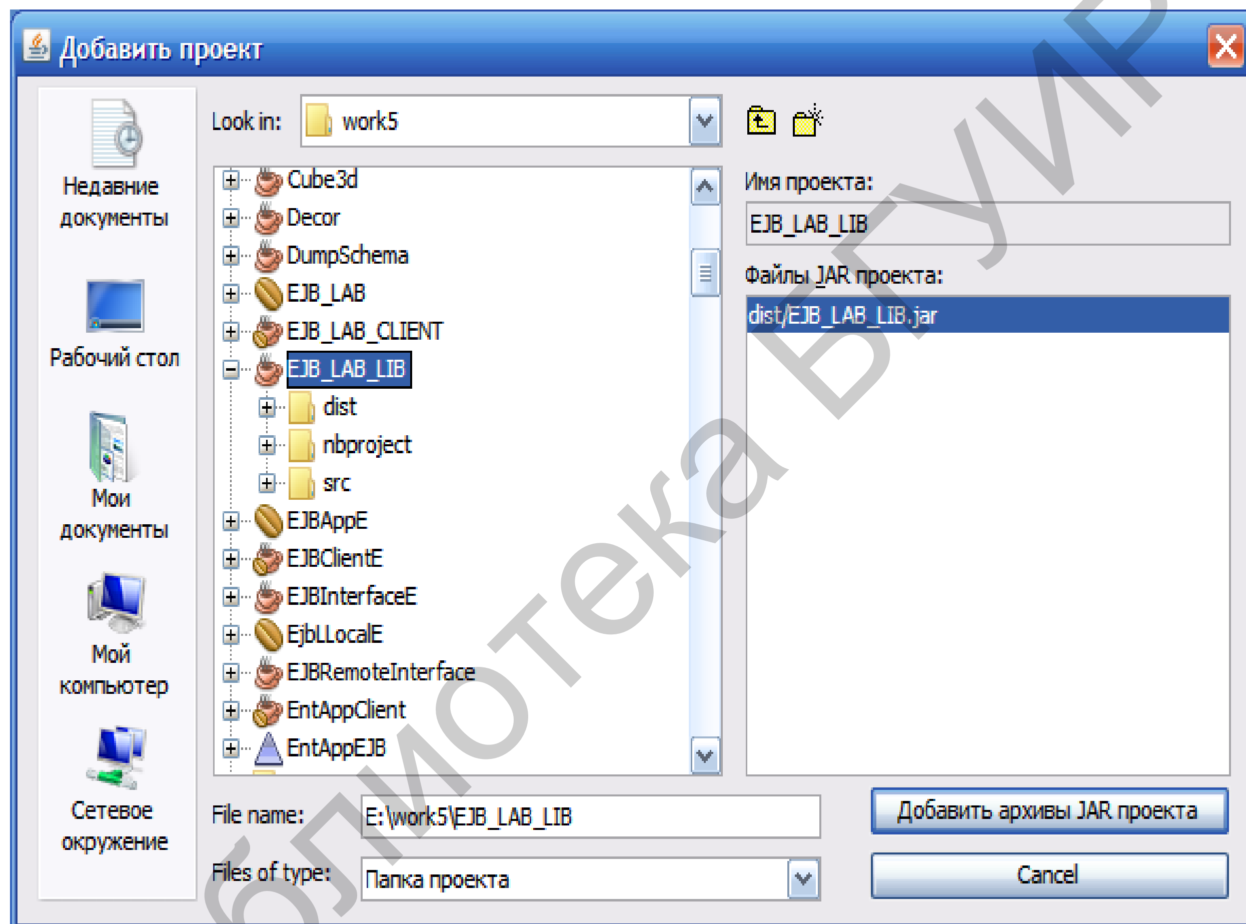


Рисунок 120 – Добавление библиотеки классов

Библиотека БГУИР

Нажмите кнопку Добавить архивы JAR проекта. Откройте исходный файл Main.java клиентского приложения в редакторе. Добавьте в него ссылку на сеансовый компонент следующим образом.

Щелкните правой кнопкой мыши на исходном коде и выберите Insert Code (Вставить код) (Alt-Insert), затем выберите Call Enterprise Bean (Вызов компонента EJB), чтобы открыть диалоговое окно вызова компонента корпоративного уровня (рисунки 121, 122).

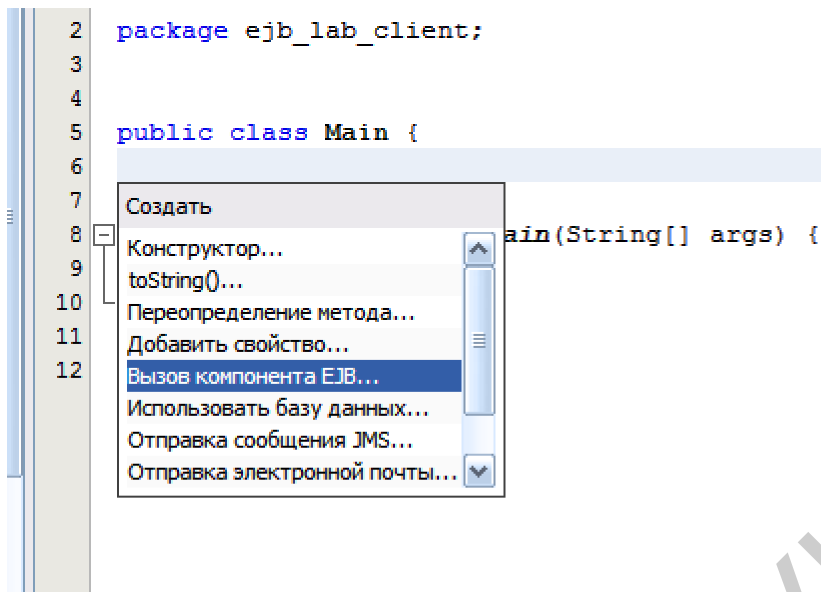


Рисунок 121 – Вызов компонента корпоративного уровня

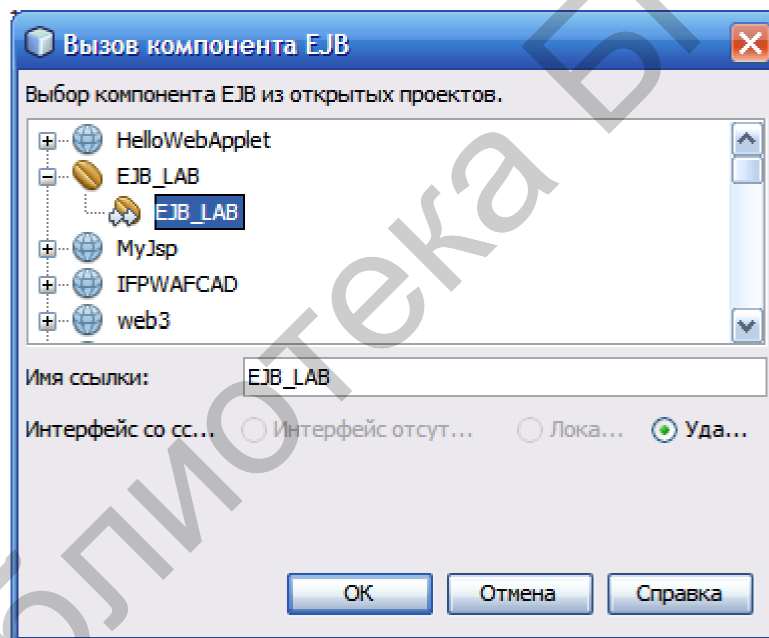


Рисунок 122 – Вызов компонента корпоративного уровня (продолжение)

Выберите узел проекта `EJB_LAB` (см. рисунок 122), затем сам компонент (с тем же именем в данном случае).

Нажмите `OK`. Клиентское приложение запишем таким образом:

```
package ejb_lab_client;

import com.EJB_LABRemote;
import javax.ejb.EJB;

public class Main
{
    @EJB
```

Библиотека БГУИР

```
    private static EJB_LABRemote eJB_LAB;
    public static void main(String[] args) {
     System.out.println("Информация о студенте Иванове: " + 
eJB_LAB.getStudInfo("ivanov"));
    }
}
```

В окне вывода результатов получим

```
Copying 1 file to E:\work5\EJB_LAB_CLIENT\dist
Copying 2 files to 
E:\work5\EJB_LAB_CLIENT\dist\EJB_LAB_CLIENTClient
Warning: E:\work5\EJB_LAB_CLIENT\dist\gfdeploy\EJB_LAB_CLIENT 
does not exist.
Информация о студенте Иванове: Marketing  10
run-single:
ПОСТРОЕНИЕ УСПЕШНО ЗАВЕРШЕНО (общее время: 1 минута 9 секунд)
```

Заметим, что клиент не видит методы компонента

```
public String addStudent(),
public String deleteStud(Student em),
```

поскольку они не объявлены в интерфейсе компонента и являются локальными методами компонента. С учетом этого факта сформулируем задание.

Задание

Усложните приведенное приложение, включив в интерфейс методы

```
public String addStudent(),
public String deleteStud(Student em),
```

а также методы для вывода списка всех студентов, подсчета числа студентов в указанной группе, вывода списка студентов, обучающихся на указанном факультете (`department`).

Контрольные вопросы

1 Как вы понимаете назначение и принципы работы сеансового компонента?

2 Как реализуется удаленный сеансовый компонент?

3 Как добавить в компонент новый бизнес-метод?

4 Как осуществляется доступ к методам компонента из клиента?

Библиотека БГУИР

4.7 Технология `Model-View-Controller` (`Spring Java`)

Цель работы: познакомиться с технологией Spring.

Краткое теоретическое содержание

В пункте 3.5.2 представлены сведения по работе с Java Spring. Технология Java Spring базируется на триаде <Model – View – Controller>. Model представляет данные, например, на основе персистентного класса(ов). View представляет отображаемый документ (например html/jsp), через который осуществляется ввод-вывод данных. Controller содержит методы, осуществляющие стыковку между Model и View. Приложение Spring создаем как web-приложение. В процессе выполнения мастера по созданию приложений указываем платформу Spring Web MVC. Создадим заготовку приложения с именем SpringLab. В настройках мастера изменим имя обработчика (mydispatcher) (рисунок 123).

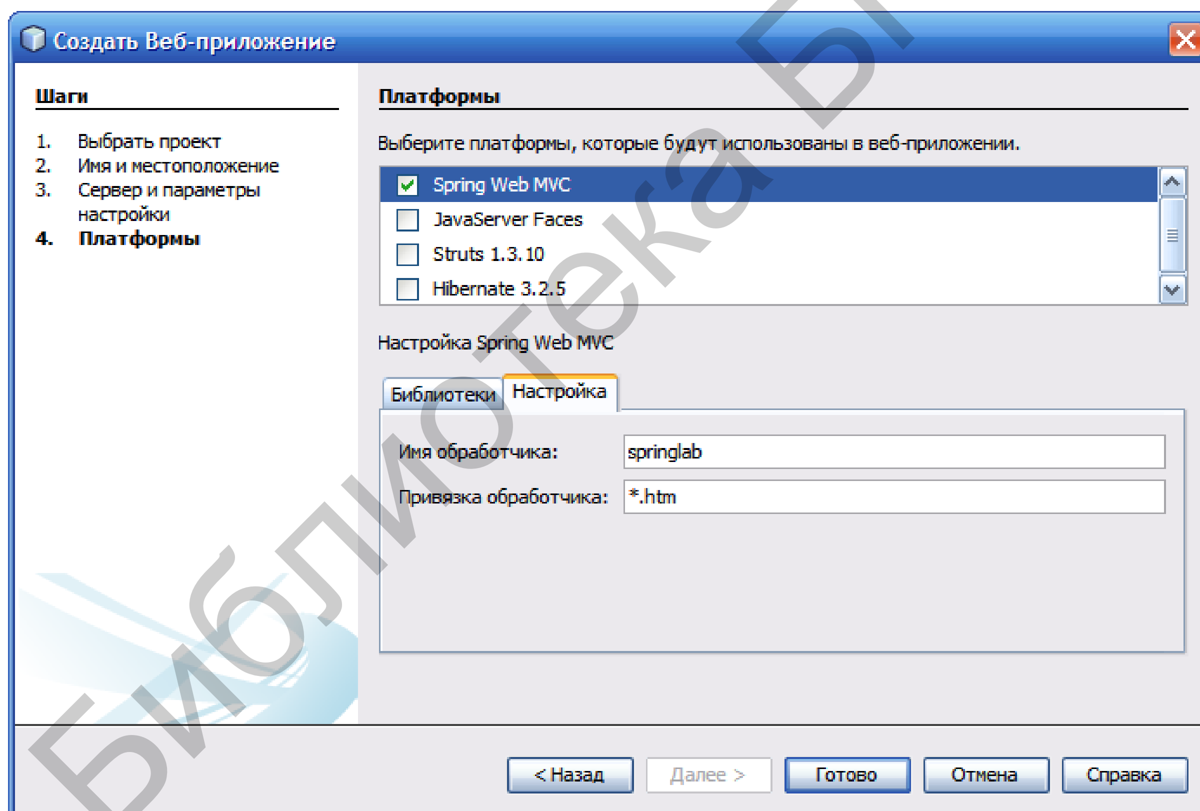


Рисунок 123 – Задание платформы Spring MVC для проекта SpringLab

Структура проекта будет иметь такой вид, как показано на рисунке 124.

Библиотека БГУИР

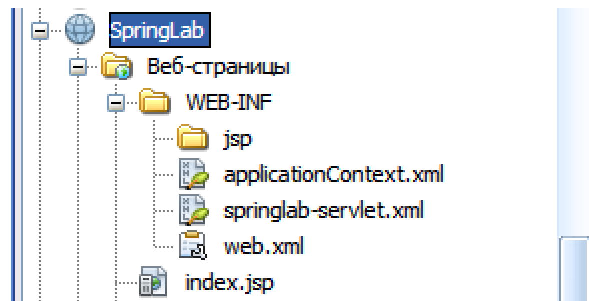


Рисунок 124 – Элементы дерева проекта на платформе Spring MVC

Сначала отредактируем конфигурационный файл web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <context-param>
     <param-name>contextConfigLocation</param-name>
     <param-value>/WEB-INF/applicationContext.xml</param-value>
    </context-param>
    <listener>
        <listener-
class>org.springframework.web.context.ContextLoaderListener</lis
tener-class>
    </listener>
    <servlet>
        <servlet-name>springlab</servlet-name>
        <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet
-class>
        <load-on-startup>2</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>springlab</servlet-name>
        <url-pattern>*.htm</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

Наша правка коснулась только одной строки:

```
<welcome-file>index.jsp</welcome-file>.
```

В этой строке указывается, что начальным файлом проекта является `index.jsp` (при запуске этот файл открывается первым). Далее изменяем содержимое файла `index.jsp`, просто переадресовав его на документ `main.html`, который, разумеется, нужно будет построить и добавить в проект:

```
<%@ page session="false"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"
%>

<html>
  <head><title>Spring LAB</title></head>
 <c:redirect url="/my.htm"/>
</html>
```

Создаем далее файл `main.jsp` и добавляем его в проект (рисунок 125).

```
<%@ page session="false"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"
%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
     <h1>Car List</h1>
     <c:forEach items="${carList}" var="car">
          ${car.model}: ${car.price}
          <br />
     </c:forEach>
    </body>
</html>
```

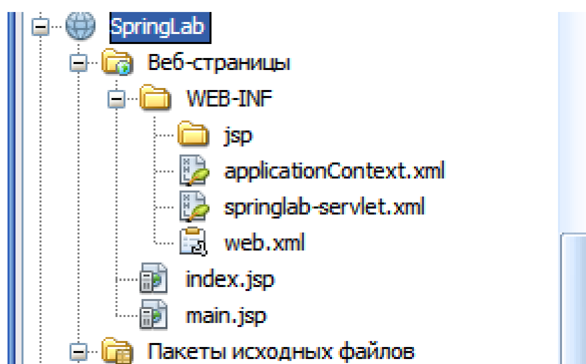


Рисунок 125 – Добавление файла `main.jsp` в дерево проекта

Библиотека БГУИР

Важно отметить, что мы вынесли файлы `index.jsp`, `main.jsp` из папки `jsp` (см. рисунок 125).

Именно на файл `main.jsp` и производится переадресация при запуске приложения (хотя расширение у него `jsp`, а не `html` – это один и тот же файл). Здесь выполняется отображение записей по автомобилям в цикле:

```
c:forEach items="${carList}" var="car".
```

Производится обращение к коллекции `carList`, с которой работает контроллер. Каждый экземпляр этой коллекции представлен переменной `car` с полями `model` и `price`.

Создаем контроллер. Это обычный класс `java-сервлета` (по сути). Размещаем его в узле `SpringLab.contr Пакета исходных файлов` (рисунок 126).

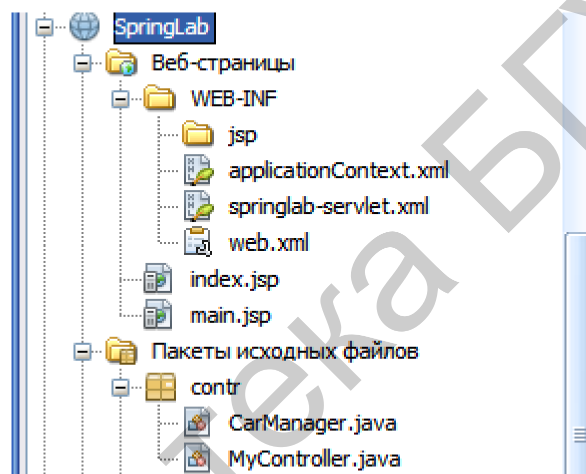


Рисунок 126 – Расширение проекта

Итак, наше приложение последовательно усложняется. У нас есть документ `view` – это `main.jsp` (`main.html`). У нас появился контроллер `MyController.java` (файл `CarManager.java` пока во внимание не берем). В конфигурационном файле `springlab-servlet.xml` мы должны эту связь прописать. Вот текст этого конфигурационного файла:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-
2.5.xsd">
  <!-- the application context definition for the springapp
DispatcherServlet -->
  <bean name="/main.htm" class="contr.MyController"/>
</beans>
```


Библиотека БГУИР

Именно строка

```
<bean name="/main.htm" class="contr.MyController"/>
```

и реализует требуемую связь `view-controller`. Мы должны записать текст конфигурационного файла так, как указано здесь. Прежде чем писать контроллер, нужно создать персистентный класс (`model`). Добавляем его в проект, создав предварительно пакет `model`:

```
package model;

public class Car {
     private Long id;
     private String model;
     private int price;

     public Long getId() {
          return id;
     }
     public void setId(Long id) {
          this.id = id;
     }

     public String getModel() {
          return model;
     }
     public void setModel(String model) {
          this.model = model;
     }
     public int getPrice() {
          return price;
     }
     public void setPrice(int price) {
          this.price = price;
     }
}
```

Теперь нам потребуется класс для доступа к объектам персистентного класса `Car`. Создаем новый класс `CarManager` в пакете `contr`:

```
package contr;
import java.util.LinkedList;
import java.util.List;
import model.Car;

public class CarManager {
     private static List<Car> carList;
     static {
               Car car1 = new Car();
          car1.setId((long)1);
          car1.setModel("SL 500");
```

Библиотека БГУИР

```
            car1.setPrice(20000);
            Car car2 = new Car();
            car2.setId((long)2);
            car2.setModel("607");
            car2.setPrice(35000);
                  Car car3 = new Car();
            car3.setId((long)3);
            car3.setModel("KIA");
            car3.setPrice(15000);
                  Car car4 = new Car();
            car4.setId((long)4);
            car4.setModel("Toyota");
            car4.setPrice(25000);

            carList = new LinkedList<Car>();
            carList.add(car1);
            carList.add(car2);
            carList.add(car3);
            carList.add(car4);
      }
      public List<Car> getCarList() {
            return carList;
      }
}
```

Дерево проекта выглядит в данный момент так, как показано на рисунке 127.

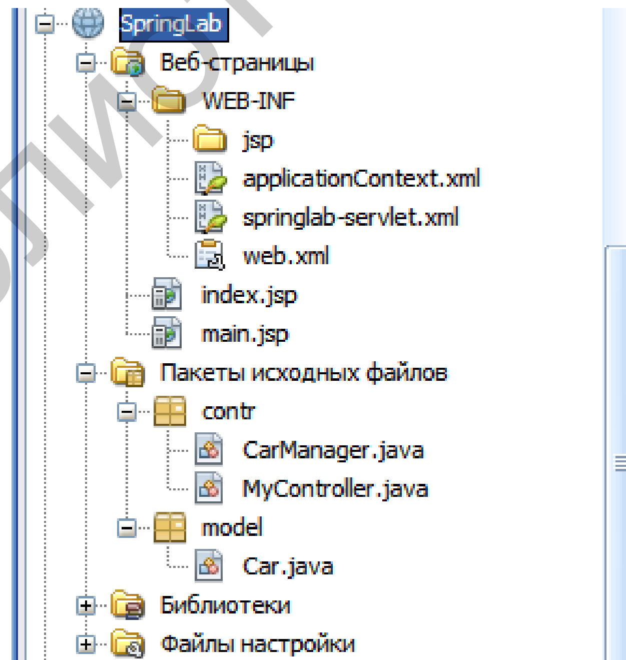


Рисунок 127 – Текущее состояние дерева проекта

Библиотека БГУИР

Наконец, пишем сам контроллер:

```
package contr;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;

public class MyController implements Controller{


     public ModelAndView handleRequest(HttpServletRequest arg0,
               HttpServletResponse arg1) throws Exception {

          CarManager carManager = new CarManager();

          ModelAndView modelAndView = new
                         ModelAndView("carList");
          modelAndView.addObject("carList",
                         carManager.getCarList());

          return modelAndView;
     }
}
```

Метод `handleRequest` является ключевым.

Конфигурационный файл `applicationContext.xml` должен иметь такой вид:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
       http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
       http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.0.xsd">

    <!--bean id="propertyConfigurer"

class="org.springframework.beans.factory.config.PropertyPlacehol
derConfigurer"
```


Библиотека БГУИР

```
        p:location="/WEB-INF/jdbc.properties" />

    <bean id="dataSource"

class="org.springframework.jdbc.datasource.DriverManagerDataSour
ce"
          p:driverClassName="${jdbc.driverClassName}"
          p:url="${jdbc.url}"
          p:username="${jdbc.username}"
          p:password="${jdbc.password}" /-->

    <!-- ADD PERSISTENCE SUPPORT HERE (jpa, hibernate, etc) -->
</beans>
```

Итак, проект SpringLab создан. Выполним его (`Очистить и построить`, затем `Развернуть`, затем `Выполнить`). Результат показан на рисунке 128.

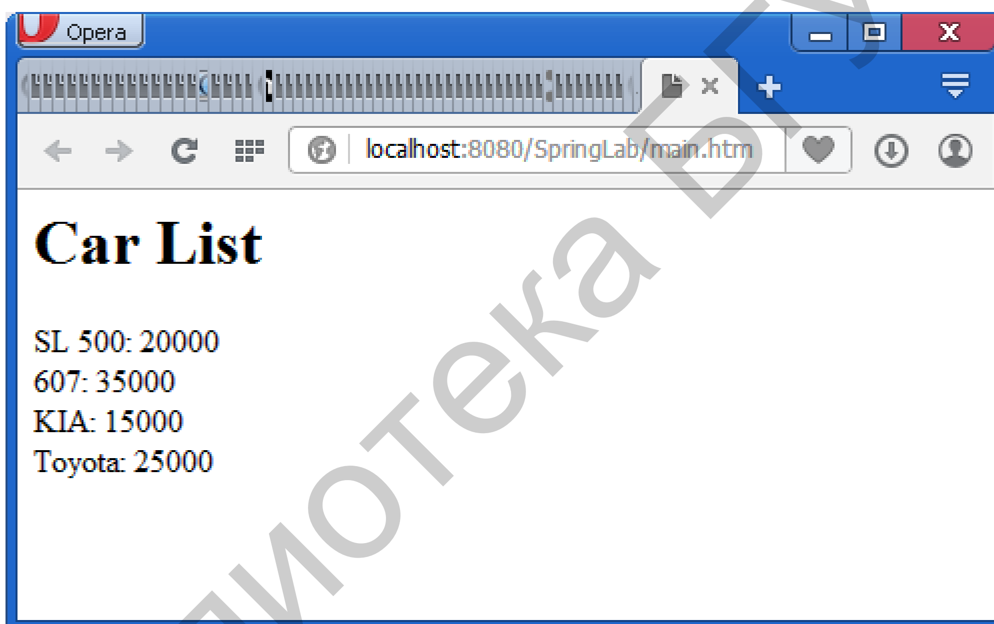


Рисунок 128 – Окно приложения

Очевидным образом данный учебный пример можно развивать в части усложнения обработки, выполняемой контроллером, добавления новых представлений и новых контроллеров. Итак, вызовем новый контроллер. Сначала в документе `view` (`main.jsp`) укажем кнопку с гиперссылкой:

```
<%@ page session="false"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@  taglib  prefix="fmt"  uri="http://java.sun.com/jsp/jstl/fmt"
%>
<html>
    <head>
        <meta    http-equiv="Content-Type"    content="text/html;
charset=UTF-8">
```


Библиотека БГУИР

```
        <title>JSP Page</title>
    </head>
    <body>
        <form:form  commandName="userForm">

<a href="hello.htm"> Say some words</a>
<input type="button"
onclick="location.href='hello.htm'"
value="TryAnotherController" >


  <h2><B><center>Car List</center></B></h2>
        <br/>
     <c:forEach items="${carList}" var="car">
            ${car.model}: ${car.price}
          <br />
     </c:forEach>
  </form:form>
    </body>
</html>
```

Затем создадим новый класс контроллера `AnotherController.java`:

```
package contr;

import java.util.Date;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;

public class AnotherController implements Controller{

    @RequestMapping(value     =     "/hello.htm",     method     =
RequestMethod.GET)
     public ModelAndView handleRequest(HttpServletRequest arg0,
               HttpServletResponse arg1) throws Exception {

           String now = (new Date()).toString();
                 return   new   ModelAndView("hello.jsp",   "now",
now);

     }
}
```

Необходимо отследить добавление всех указанных импортов. Теперь в конфигурационном файле `springlab-servlet.xml` пропишем связь между `view` и `controller`:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-
2.5.xsd">
  <!-- the application context definition for the springapp
DispatcherServlet -->
   <bean name="/main.htm" class="contr.MyController"/>
   <bean name="/hello.htm" class="contr.AnotherController"/>
</beans>
```

Запустим приложение на выполнение. Нажмем кнопку или гиперссылку. Получим скриншот, показанный на рисунке 129.

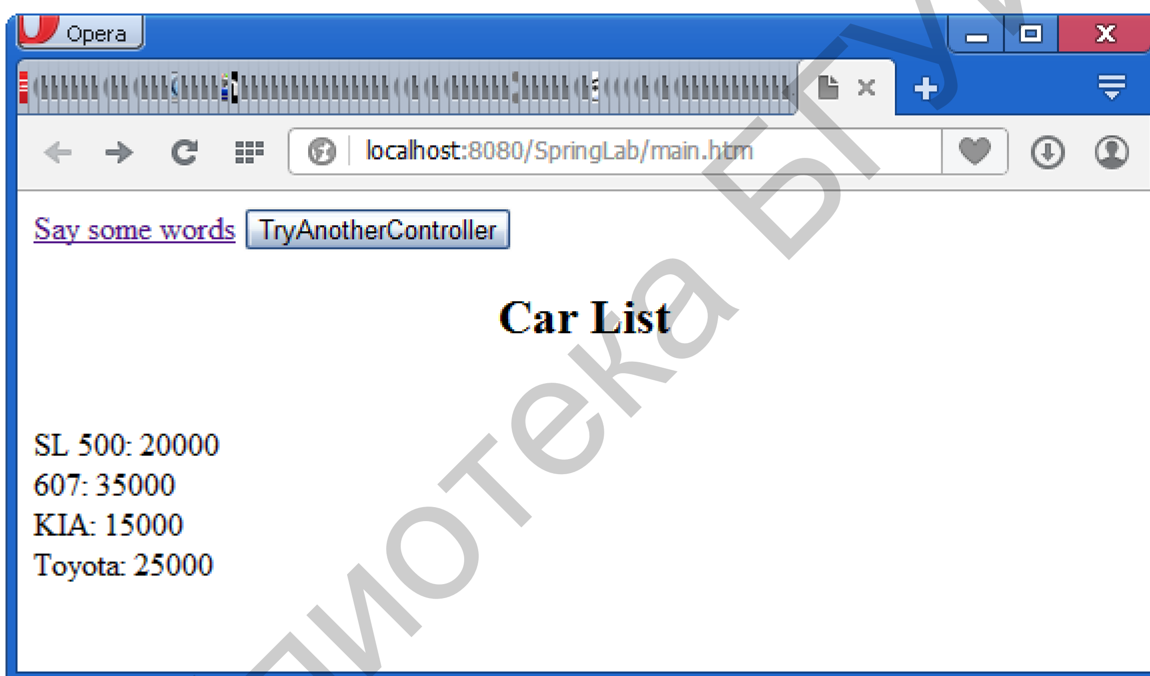


Рисунок 129 – Добавление элементов интерфейса

Остается последнее – указать, как передавать данные в контроллер. Разместим в документе поле ввода (выделено цветом):

```
    <body>
        <form:form  commandName="userForm">
<a href="hello.htm"> Say some words</a>
<input    type="button"       onclick="location.href='hello.htm'"
value="TryAnotherController" >
  <br/>
  <flow>
    <input name="carmodel" type="string"
           required="true" />
</flow>
```


Библиотека БГУИР

```
            <h2><B><center>Car List</center></B></h2>
        <br/>
     <c:forEach items="${carList}" var="car">
            ${car.model}: ${car.price}
          <br />
     </c:forEach>
  </form:form>
```

При запуске приложения окно имеет вид, показанный на рисунке 130.

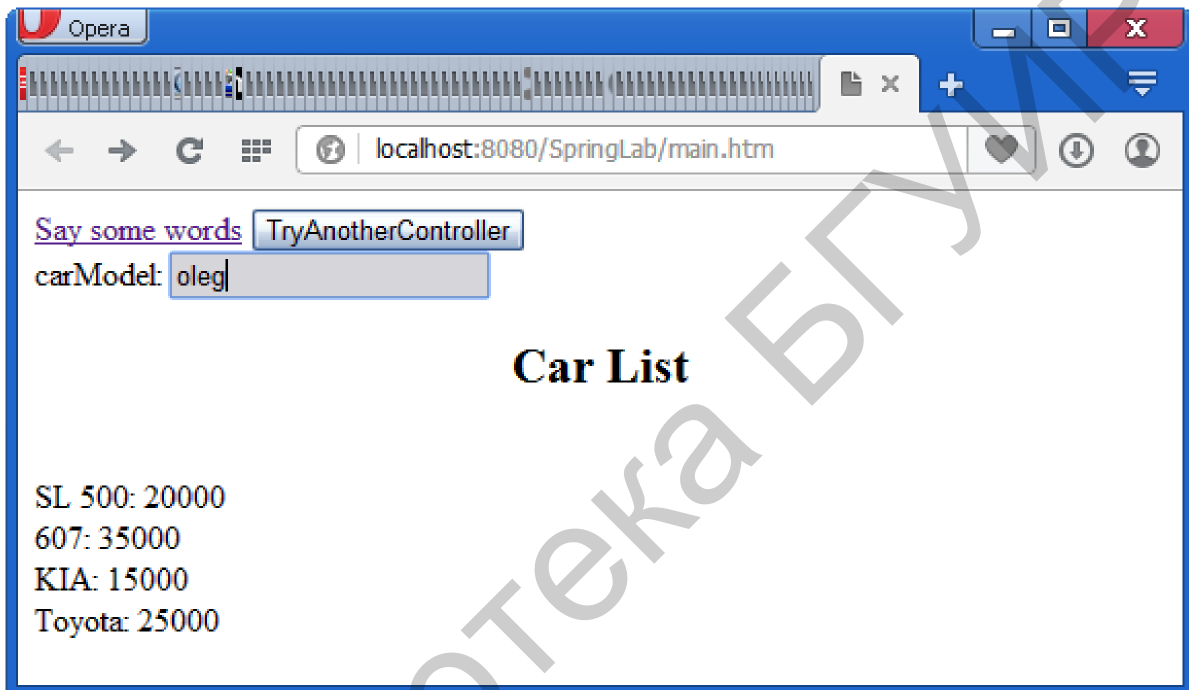


Рисунок 130 – Добавление поля ввода

У поля ввода есть параметр `name`. По значению этого параметра мы можем получить к нему доступ. Для этого несколько изменим текст файла `main.jsp`:

```
<%@ page session="false"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@  taglib  prefix="fmt"  uri="http://java.sun.com/jsp/jstl/fmt"
%>
<html>
    <head>
        <meta    http-equiv="Content-Type"    content="text/html;
charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body bgcolor="#aaffee">
        <form  method="POST" action="hello.htm">
```

Библиотека БГУИР

```
<a href="hello.htm"> Say some words</a>
<input    type="button"       onclick="location.href='hello.htm'"
value="TryAnotherController_A" >
  <br/>
  <br/>

 <flow>

      input     model:     <input     name="model"     Id="model"
type="String"/>

  </flow>
<br/>
 <input type="Submit" value="TryAnotherController_B" >
<br/>
              <h2><B><center>Car List</center></B></h2>
        <br/>
     <c:forEach items="${carList}" var="car">
           ${car.model}: ${car.price}
          <br />
     </c:forEach>
  </form:form>
    </body>
</html>
```

Теперь после запуска приложения открывается окно, показанное на рисунке 131.

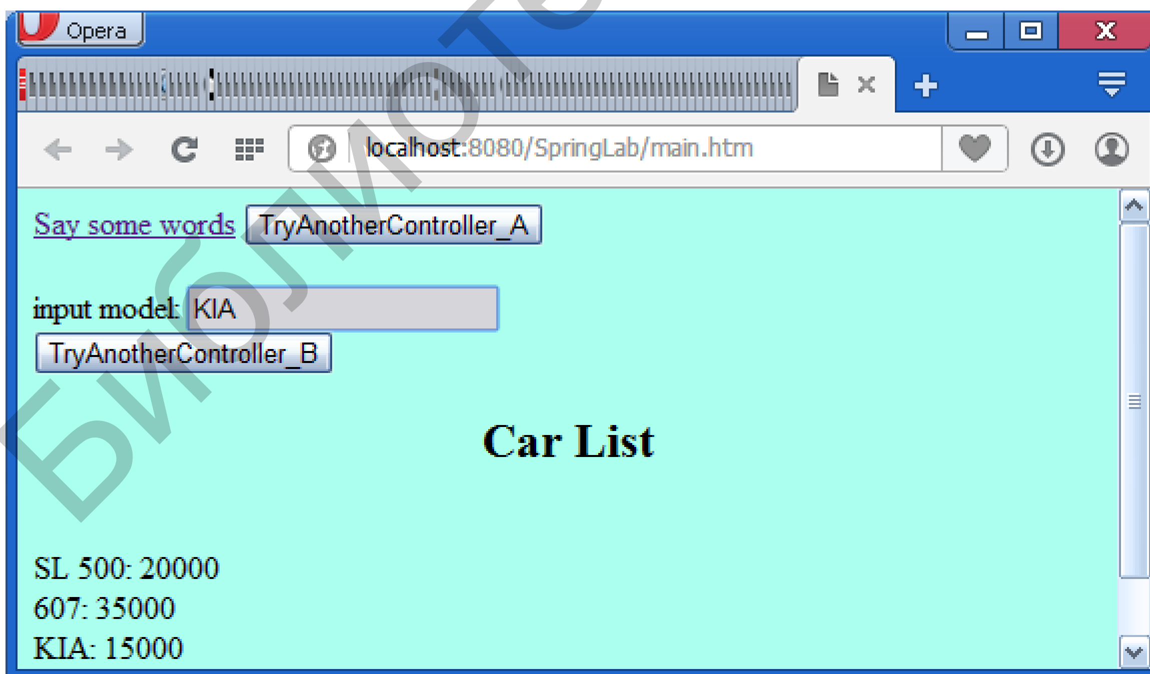


Рисунок 131 – Добавление еще одного контроллера

Библиотека БГУИР

Отличие кнопки `TryAnotherContrioller_B` от `TryAnotherContrioller_A` состоит в том, что первая обеспечивает передачу в контроллер значений элементов формы, в то время как вторая кнопка этого не делает. Набрав текст `KIA` в окне ввода, нажмем кнопку `TryAnotherContrioller_B` и получим окно, как на рисунке 132.

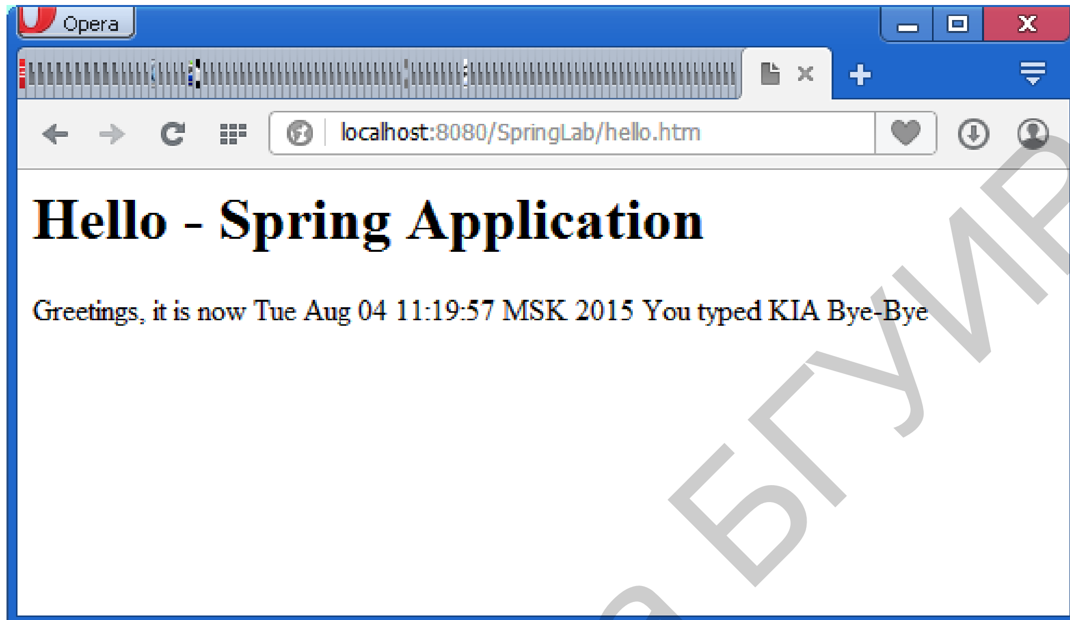


Рисунок 132 – Реакция второго контроллера

Текст контроллера мы изменили таким образом:

```
package contr;

import java.util.Date;
import java.util.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;
import model.Car;

public class AnotherController implements Controller{

    @RequestMapping(value     =     "/hello.htm",     method     =
RequestMethod.GET)

     public ModelAndView handleRequest(HttpServletRequest arg0,
               HttpServletResponse arg1) throws Exception {

        String parname="-";
     Enumeration pars=arg0.getAttributeNames();
```


Библиотека БГУИР

```
        while(pars.hasMoreElements())
        {
            parname=parname+(String) pars.nextElement()+"\n";
        }
        String now = (new Date()).toString()+ "    You typed "+arg0.getParameter("model");// arg0.getParameter("carmodel");
       return new ModelAndView("hello.jsp", "now", now);
    }}
```

Задание

Усложните приведенное приложение, включив в интерфейс методы для поиска цены автомобиля по его модели, добавления автомобиля в персистентный класс, поиска автомобилей, цена на которые не превосходит введенную в форме клиента.

Контрольные вопросы

1 Как вы понимаете назначение и принципы работы контроллера?
2 Как вы понимаете назначение и принципы работы представления?
3 Как связать представление (`view`) с контроллером (`controller`)?
4 Как связать персистентный класс с контроллером?

4.8 Технология `HIBERNATE`

Цель работы: познакомиться с технологией `Hibernate`.

Краткое теоретическое содержание

В пункте 3.5.1 представлены сведения по работе с `Hibernate`. Технология `Hibernate` предназначена для работы с базой данных как с классом (такой класс называется персистентным). Доступ к полям выполняется с помощью методов `set` и `get`. Создадим проект на базе обычного приложения `Java` и реализуем в нем визуальный интерфейс для работы с базой данных. Назовем проект `HiberLabVis`. Подключим к нему библиотеку `Hibernate JPA`. Кроме того, подключаем драйвер `derbyClient.jar` из папки `lib` инсталляции `GlassFish`. Основной класс размещается в пакете `hiberlabvis`. В пакете `perst` размещаем персистентный класс.

Библиотека БГУИР

Структура дерева проекта (законченный вариант) имеет вид, представленный на рисунке 133.

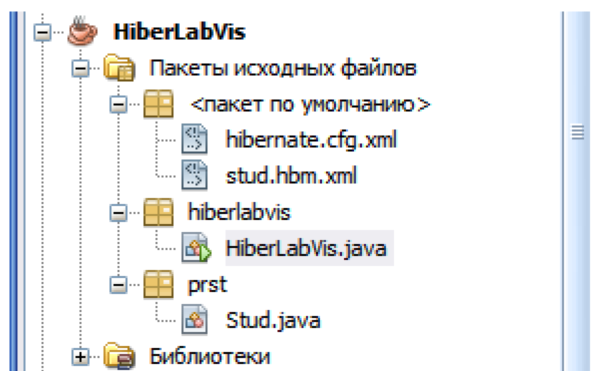


Рисунок 133 – Структура проекта на основе `Hibernate`

Вот текст персистентного класса `Stud.java`

```
package prst;

import org.hibernate.annotations.GenericGenerator;
import java.util.*;
import java.io.*;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import org.hibernate.Session;

 class stud implements Serializable{

   private String fio;
   private int age;

   public stud() {}

   public stud(String name, int g) {
       this.fio = name;
       this.age = g;
   }
    public void setFio(String name) {
        this.fio = name;
    }
```

Библиотека БГУИР

```
     public int getAge() {
        return age;
    }

    public String getFio() {
        return fio;
    }
    public void setAge(int g) {
        this.age = g;
    }
}
```

Сформируем теперь основной класс приложения `HiberLabVis.java`:

```
package hiberlabvis;
import org.hibernate.annotations.GenericGenerator;
import java.util.*;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import org.hibernate.Session;
import prst.*;
import java.awt.*;
import java.awt.event.*;

public class HiberLabVis extends Frame implements ActionListener
{
      private static SessionFactory sessionFactory;
      static Session session = null;
      static  Connection connection = null;
      static  Statement statement = null;
      static  ResultSet rs = null;
      private static String dbURL =

"jdbc:derby://localhost:1527/MeineBase;user=oleg;password=german
";
      private static Connection conn = null;
      static boolean bcon=false;
            Button bex=new Button("Выход");
            Button connect=new Button("Соединить");
            Label lfio=new Label("Имя");
            Label lage=new Label("Возраст");
```


Библиотека БГУИР

```
     static TextField tfio = new TextField();
     static TextField tage = new TextField();
            Label lstat=new Label("Статус");
     static TextField tstat = new TextField();

  public  HiberLabVis()
{
  super("my window");
  setLayout(null);
  setBackground(new Color(150,200,100));
  setSize(350,450);
  add(bex);
  bex.setBounds(120,250,100,20);
  bex.addActionListener(this);
  add(connect);
  connect.setBounds(120,280,100,20);
  connect.addActionListener(this);

  add(lfio);
  lfio.setBounds(30,30,80,20);
  add(lage);
  lage.setBounds(30,60,80,20);
  add(tfio);
  tfio.setBounds(120,30,100,20);
  add(tage);
  tage.setBounds(120,60,100,20);
  add(lstat);
  lstat.setBounds(30,420,100,20);
  add(tstat);
  tstat.setBounds(120,420,140,20);
  tstat.setText("База отсоединена");

  this.show();
  this.setLocationRelativeTo(null);
}

  public void actionPerformed(ActionEvent ae)
{
   if(ae.getSource()==bex)
   System.exit(0);
   else
    if(ae.getSource()==connect)
   ConnectBase();

   }

     public static void ConnectBase() {
             if(bcon==false)
         {
        tstat.setText("Строим Factory");
        sessionFactory =
 new Configuration().configure().buildSessionFactory();
```


Библиотека БГУИР

```
        tstat.setText("Session Factory построена");
        tstat.setText("Соединяемся с базой");
        try
        {

Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance(
);
            conn = DriverManager.getConnection(dbURL);
        }
        catch (Exception except)
        {

tstat.setText("Ошибка соединения:"+except.getMessage());
        }

           tstat.setText("Соединение    установлено    с    Derby
database");
           bcon=true;
           session = sessionFactory.openSession();
           session.beginTransaction();
           List<Stud>            result=session.createQuery("from
Stud").list();
            for(Stud x:result)
            {
             tfio.setText(""+x.getFio());
             tage.setText(""+x.getAge());
             break;
            }
             session.getTransaction().commit();
             try
             {
             //conn.close();
             }

        catch(Exception ex)
        {
         tstat.setText("Соединение не было установлено!!!");
         bcon=false;
         }
         }

         else
         {
          tstat.setText("База уже подсоединена");

         }

     }

     public static void main(String[] args) {
        new HiberLabVis();
```

```
    }
  }
```

Окно приложения в действии показано на рисунке 134.

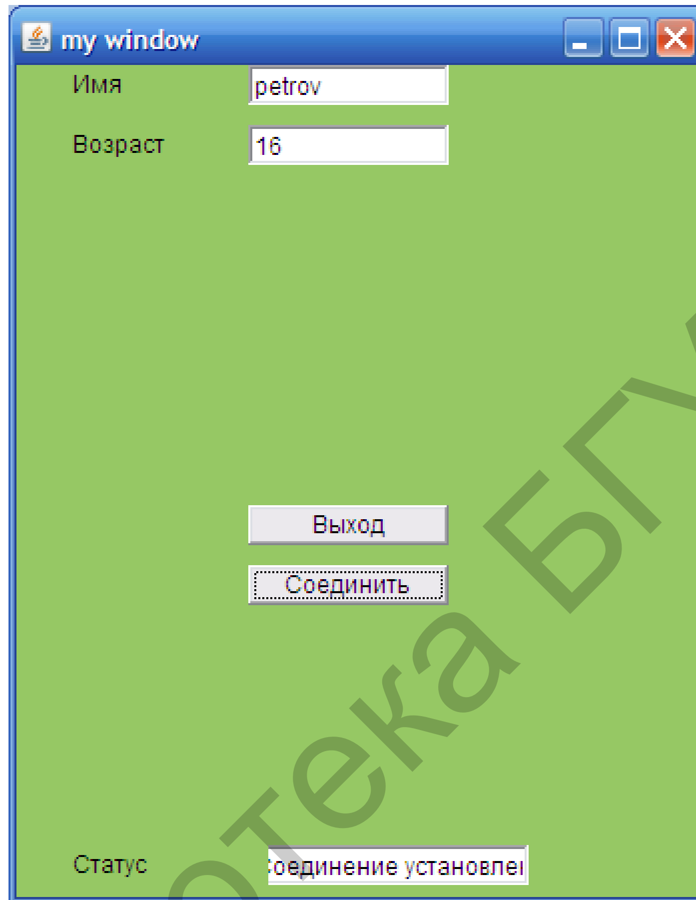


Рисунок 134 – Окно приложения

В классе объявлены текстовые метки, поля и кнопки:

```
Button bex=new Button("Выход");
            Button connect=new Button("Соединить");
            Label lfio=new Label("Имя");
            Label lage=new Label("Возраст");
     static TextField tfio = new TextField();
     static TextField tage = new TextField();
            Label lstat=new Label("Статус");
     static TextField tstat = new TextField();
```

Все эти элементы добавляются в конструкторе:

```
add(lfio);
lfio.setBounds(30,30,80,20);
add(lage);
```


Библиотека БГУИР

```
  lage.setBounds(30,60,80,20);
  add(tfio);
  tfio.setBounds(120,30,100,20);
  add(tage);
  tage.setBounds(120,60,100,20);
  add(lstat);
  lstat.setBounds(30,420,100,20);
  add(tstat);
  tstat.setBounds(120,420,140,20);
  tstat.setText("База отсоединена");
```

Для кнопок добавляются прослушиватели событий, например:

```
bex.addActionListener(this);
```

Обработчик событий реализован следующим образом:

```
  public void actionPerformed(ActionEvent ae)
{
   if(ae.getSource()==bex)
   System.exit(0);
   else
    if(ae.getSource()==connect)
   ConnectBase();

   }
```

Метод `actionPerformed` является стандартным методом обработки событий от кнопок и меню.

Поля класса `Stud` должны быть сопоставлены с колонками таблицы в базе данных. Для этого мы используем специальный конфигурационный `map`-файл с именем `stud.hbm.xml`. Вот его вид:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE   hibernate-mapping   PUBLIC   "-//Hibernate/Hibernate
Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="HiberLabVis">
    <class name="prst.Stud" table="stud">
       <id name="fio" column="fio">
            <generator class="native"/>
        </id>

        <property name="age" column="age"/>
    </class>
</hibernate-mapping>
```


Библиотека БГУИР

Заметим, что один столбец должен быть ключевым (в нашем примере – это столбец `fio`). Ключевой столбец объявляется в теге `<id>`. Также нужно определить второй конфигурационный файл `hibernate.cfg.xml`. Он стандартный и имеет такой вид:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-
3.0.dtd">

<hibernate-configuration>

    <session-factory>

      <!-- Database connection settings -->
        <property
name="connection.driver_class">org.apache.derby.jdbc.ClientDrive
r</property>
        <property
name="connection.url">jdbc:derby://localhost:1527/MeineBase</pro
perty>
        <property name="connection.username">oleg</property>
        <property name="connection.password">german</property>

      <!-- SQL dialect -->
        <property
name="dialect">org.hibernate.dialect.DerbyDialect</property>

      <!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">1</property>

      <!-- Enable Hibernate's automatic session context
management -->
        <property
name="current_session_context_class">thread</property>

      <!-- Disable the second-level cache -->
        <property
name="cache.provider_class">org.hibernate.cache.NoCacheProvider<
/property>

      <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">true</property>


      <!-- Mapping files -->
        <mapping resource="stud.hbm.xml"/>

    </session-factory>
```


Библиотека БГУИР

```
</hibernate-configuration>
```

В этом файле объявлен класс драйвера

```
<property
name="connection.driver_class">org.apache.derby.jdbc.ClientDrive
r</property>
```

Объявлена строка соединения

```
        <property
name="connection.url">jdbc:derby://localhost:1527/MeineBase</pro
perty>,
```

а также имя пользователя и пароль.

Задание

Усложните приведенное приложение, доработав интерфейс с базой данных, добавив методы для поиска, редактирования (ввода новых) записей и удаления.

Контрольные вопросы

1 В чем специфика технологии Hibernate?
2 Какие конфигурационные файлы использует Hibernate и каковы их роли?
3 Как осуществляется привязка персистентного класса к таблице базы данных?
4 Как получить доступ к записям из персистентного класса?

Библиотека БГУИР

ПРИЛОЖЕНИЕ А
(обязательное)

Краткое введение в язык Java SDK

Имеется две основные среды программирования на языке Java – Java SDK (Standard development kit) и JEE (Java Enterprise Edition). Java SDK – это и есть «стандартный» Java. Java EE – это «корпоративный» Java, который используется на предприятиях для разработки распределенных и сетевых систем. Большинство примеров этого учебно-методического пособия относилось к Java EE. Однако знание Java SDK необходимо в качестве первой ступени владения языком. Для освоения используем среду NetBeans (свободно скачиваемую в Интернете).

А.1 Установка Java и NetBeans

Сначала устанавливаем Java. Скачиваем файл java_ee_sdk-6u4-jdk7-windows.exe (рисунок А.1) и запускаем его.

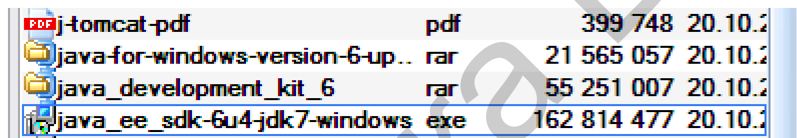


Рисунок А.1 – Инсталляция Java SDK

Затем скачиваем и устанавливаем среду выполнения java JRE (рисунок А.2).

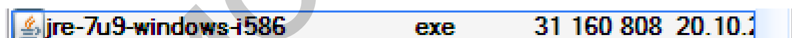


Рисунок А.2 – Инсталляция Java JRE

Затем скачиваем и устанавливаем NetBeans (рисунок А.3).

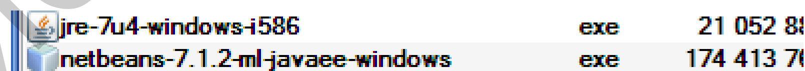


Рисунок А.3 – Инсталляция Java NetBeans

Запускаем NetBeans.

А.2 Объектные принципы Java

Консольные приложения

Java-приложение представляет собой один или несколько классов. В качестве первого примера рассмотрим следующий.

Библиотека БГУИР

Пример простого использования класса:

```
package object_programming_principlese;
import java.awt.*;
import java.util.Scanner;

 class SimpleArithmetic
 { int Add(int par1,int par2)
     { return par1+par2;}}

public class Object_Programming_PrinciplesE {
     public static void main(String[] args) {
        System.out.println("Input first number");
        Scanner input= new Scanner(System.in);
        int n1=input.nextInt();
        System.out.println("Input second number");
        int n2=input.nextInt();
        SimpleArithmetic sa=new SimpleArithmetic();
        System.out.printf("The sum of %d and %d is
%d",n1,n2,sa.Add(n1, n2)); }}
```

В приведенном `java`-приложении объявлено два класса:

`SimpleArithmetic` и `Object_Programming_PrinciplesE`,

объединенных в пакет `object_programming_principles`. Пакет объявляется как самая первая строка программы. Если пакет не объявлен, то он используется как анонимный (без имени). Следовательно, пакет является вариантом объединения классов. Классы в одном и том же пакете «видят» друг друга, т. е. могут использовать друг друга, если это позволяют их уровни доступа.

Уровень доступа `public` является предельно общим и делает класс доступным для использования в других классах. Уровень доступа `private`, напротив, делает класс недоступным для использования в других классах.

Уровень доступа `protected` делает класс доступным только в дочерних классах. Если не объявлять спецификатор доступа, то класс будет «видим» в пределах папки (каталога – folder). Класс, объявленный как `public`, должен содержать точку входа – метод `main`, с которого начинается его выполнение. Классы состоят из методов и переменных – членов класса. В данном примере класс `SimpleArithmetic` содержит метод `Add`, а класс `Object_Programming_PrinciplesE` – метод `main`. Метод должен возвращать значение определенного типа и, как правило, использует аргументы. Так, метод `int Add(int par1,int par2)` возвращает значение целого типа (`int`) и использует два параметра `par1`, `par2` также целого типа. Метод заканчивается командой `return`, которая и возвращает ответ. В методе может быть несколько команд `return`. Классы и методы должны записываться в фигурных скобках.

Библиотека БГУИР

Объекты создаются с помощью конструкторов классов – методов, имена которых совпадают с именами классов. В нашем примере объект класса `SimpleArithmetic` создавался в команде

```
SimpleArithmetic sa=new SimpleArithmetic();
```

При обращении к конструктору обязательно указывается ключевое слово `new`. Конструкторы в общем случае должны объявляться в классах, однако если конструктор не содержит параметров, то его можно не объявлять (действует объявление по умолчанию – by default). В рассматриваемом примере именно это имеет место.

Если хотим использовать классы без объектов, то нужно объявлять методы как `static`. Имеются стандартные классы `Java`, подключаемые с помощью библиотечных пакетов. Подключение их реализуется через команду `import`. В рассматриваемом примере использованы стандартные классы `System.out` и `Scanner`. Оба используются для консольного ввода-вывода. Они подключаются посредством команд

```
import java.awt.*;
import java.util.Scanner;
```

Объект класса `Scanner` используется для ввода значения с клавиатуры. Создаем этот объект таким образом:

```
Scanner input= new Scanner(System.in);
```

Читаем значения параметров с помощью этого объекта (с именем) так:

```
int n1=input.nextInt();
```

и

```
int n2=input.nextInt();
```

Класс используется для вывода результатов/текста на консоль, например:

```
System.out.println("Input second number");
```

Для вывода используется метод `println`.

Итак, рассматриваемое приложение начинается с метода `main` и приглашает ввести два числа. Затем создается объект класса `SimpleArithmetic`, вызывается метод `sa.Add(n1, n2)` (с указанием в качестве параметров двух введенных целых чисел `n1, n2`) этого объекта. Заметим, что обращение к методу `sa.Add` выполняется внутри другого метода:

```
System.out.printf("The sum of %d and %d is
                   %d",n1,n2,sa.Add(n1, n2));
```

Вообще, `System.out.printf` используется для форматированного вывода. Специальные обозначения `%d` (place holders) используются для подстановки вместо них параметров, перечисляемых после закрывающих двойных кавычек. Маленькая литера `d` указывает, что подставляется значение целого типа.

Библиотека БГУИР

A.3 Обработка исключений

Важную роль играет обработка исключений (ошибок). Как правило, программные ошибки вызывают завершение приложения и выдачу диагностической информации. Во многих случаях при возникновении ошибок следует продолжить программу, предусмотрев выполнение определенных действий. Для обработки исключительных ситуаций используют конструкции типа `try {…} catch(Exception e){…}`. Написанный здесь образец является предельно общим. Поясним его следующим примером, где выполняется попытка деления на 0:

```
package zerodivision;
import java.awt.*;
import java.util.Scanner;

public class ZeroDivision {

    public static void main(String[] args) {

        System.out.println("Input first number");
        Scanner input= new Scanner(System.in);
        int n1=input.nextInt();
        System.out.println("Input second number");
        int n2=input.nextInt();
     try
 {      double res=(double)(n1 / n2);
        System.out.printf("The   division   of   %d   by   %d   is
%f",n1,n2,res);
 }
 catch(Exception ex)
 {System.out.println("Division   by   zero   is   forbidden!!!
"+ex.getMessage()); }}}
```

Если при вводе второго числа ввести `0`, то возникнет исключение, которое будет перехвачено блоком `catch`. Программа при этом аварийно не завершается. Наряду с `try` и `catch` используют также конструкцию `finally`, которая выполняется всегда, есть ли ошибка или нет.

A.4 Потоки

Поток – это ветвь основной программы, которая выполняется параллельно ей. Если нужно объявить, что класс использует поток, то в объявлении класса указываем интерфейс `implements java.lang.Runnable`. В данном интерфейсе объявлен метод `run` – основной метод потока. Когда поток стартует, то он начинает выполнять метод `run`. После завершения метода `run` поток автоматически уничтожается. Следующий фрагмент кода

Библиотека БГУИР

содержит пример метода `run`, который мы используем в учебном приложении:

```
  public void run() {
         while ( fThread != null){
            try{
                Thread.sleep (300);
                ++bellTolls;
    javax.swing.ImageIcon img =
                             new
javax.swing.ImageIcon("E:\\work5\\Ex1bThread\\src\\Bella.gif");
                aLabel.setIcon(img);

            }
            catch  (InterruptedException e) { }
            anOther();
            if (this.isDone())  fThread = null;}
```

Здесь программное имя потокового объекта `fThread`. Происходит динамически повторяемое поочередное отображение двух картинок (`icon`) через паузу. Первая картинка создается в команде

```
     javax.swing.ImageIcon img =
                             new
javax.swing.ImageIcon("E:\\work5\\Ex1bThread\\src\\Bella.gif");
```

и отображается в поле label:

```
               aLabel.setIcon(img);
```

Метод `anOther()` создает и отображает вторую картинку и имеет следующий вид:

```
public void anOther(){
        try{
            Thread.sleep (300);
              javax.swing.ImageIcon img =
new
javax.swing.ImageIcon("E:\\work5\\Ex1bThread\\src\\Sun.gif");
                aLabel.setIcon(img);
                   }
        catch  (InterruptedException e) { }}
```

Через паузу (`Thread.sleep`) в 300 мс на ярлыке отображается другая картинка (`Sun.gif`) – рисунок А.4.

Библиотека БГУИР

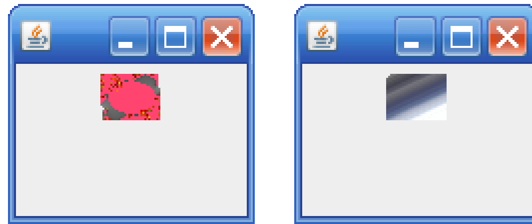


Рисунок А.4 – Смена картинок в потоках

Приводим полный текст программы:

```
package ex1bthread;
import java.awt.*;
import javax.swing.*;

public  class  Ex1bThread  extends  javax.swing.JFrame  implements
java.lang.Runnable{
    public Ex1bThread() {
        aPanel = new javax.swing.JPanel();
        aLabel = new javax.swing.JLabel();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLO
SE);
        setLocationRelativeTo(null);
        setPreferredSize(new java.awt.Dimension(50, 110));
        setBackground(Color.PINK);
        add(aPanel);
        aPanel.add(aLabel);
        pack();
        start();
    }

    public final void start() {
        if (fThread == null){
            fThread = new java.lang.Thread (this);
            fThread.start();
        }
    }

       public void run() {
        while ( fThread != null){

            try{
                Thread.sleep (300);
                ++bellTolls;

                javax.swing.ImageIcon img =
new
javax.swing.ImageIcon("E:\\work5\\Ex1bThread\\src\\Bellb.gif");
                aLabel.setIcon(img);
            }
```


Библиотека БГУИР

```
            catch  (InterruptedException e) { }
            anOther();
            if (this.isDone())  fThread = null;
        }
    }

    public void anOther(){

        try
       {
            Thread.sleep (300);
              javax.swing.ImageIcon           img           =new
javax.swing.ImageIcon("E:\\work5\\Ex1bThread\\src\\Sun.gif");
                aLabel.setIcon(img);

        }
        catch  (InterruptedException e) { }

     }

    public boolean isDone() {
        boolean temp=false;
        if(bellTolls==COUNTS) {
            temp=true;
            bellTolls=0;
        }
        return temp;
    }

    public static void main(String args[]) {
       Ex1bThread z=
        new Ex1bThread();
       z.setVisible(true);
       z.setBackground(Color.PINK);

    }
    private javax.swing.JLabel aLabel;
    private javax.swing.JPanel aPanel;
    Thread fThread;
    int bellTolls=0;
    final int COUNTS=50;

}
```

Конструктор основного класса `Ex1bThread` объявлен и реализован следующим образом:

```
    public Ex1bThread() {
        aPanel = new javax.swing.JPanel();
        aLabel = new javax.swing.JLabel();
```


Библиотека БГУИР

```
setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLO
SE);
        setLocationRelativeTo(null);//размещает окно по центру
                                    //экрана
        setPreferredSize(new java.awt.Dimension(50, 110));
        setBackground(Color.PINK);
        add(aPanel);
        aPanel.add(aLabel);
        pack();
        start();}
```

Создается панель (вид окна)

```
aPanel = new javax.swing.JPanel(),
```

на которую добавляем текстовый ярлык (`Label`):

```
aPanel.add(aLabel).
```

Вызываем метод `start` для запуска потока:

```
    public final void start() {
        if (fThread == null){
            fThread = new java.lang.Thread (this);
            fThread.start();//поток запускается командой start
        }
```

Смена картинок будет выполняться число раз, регулируемое переменной `bellToll` в методе `isDone()` (в приложении – 50 раз).

А.5 Работа с файлами

`Java` предоставляет средства для низкоуровневого (на уровне байтов) доступа к файлам и средства высокоуровневого доступа, при этом высокоуровневая объектная поточная переменная создается на основе низкоуровневой. Следующий фрагмент служит пояснением:

```
FileOutputStream fout=new FileOutputStream("e:/my.txt");
DataOutputStream dout=new DataOutputStream(fout);
dout.writeUTF("hello, File World! ");
dout.close();
```

Переменная `fout` является низкоуровневой файловой поточной переменной. Она используется в конструкторе `new DataOutputStream` для создания высокоуровневой переменной `dout`,с помощью которой и осуществляется запись в файл `e:/my.txt`. Метод `writeUTF` выполняет запись текстовых данных информации в формате `UNICODE`. Чтобы прочитать записанные в файл текстовые данные, используем следующий код:

Библиотека БГУИР

```
FileInputStream finp=FileInputStream("e:/my.txt");
DataInputStream dinp=DataInputStream(finp);
System.out.println(""+dinp.readUTF());
Dis.close();
```

Создадим оконное приложение для иллюстрации возможностей работы с файлами.

```
package fileexample;

import java.awt.event.*;
import java.io.*;
import java.awt.*;

  public    class    FileExample    extends    Frame    implements
ActionListener
{
  Button bt=new Button("Выход");
  Button bt1=new Button("Записать");
  Button bt2=new Button("Прочитать");
  TextField tf=new TextField();

  public  FileExample()
{
  super("Work with Files");
  setLayout(null);
  setBackground(new Color(250,200,120));
  setSize(250,200);
  setVisible(true);
  add(bt);
  add(bt1);
  add(bt2);
  add(tf);
  bt.addActionListener(this);
  bt1.addActionListener(this);
  bt2.addActionListener(this);
  bt.setBounds(20,40,100,20);
  bt1.setBounds(20,65,100,20);
  bt2.setBounds(20,90,100,20);
  tf.setBounds(20,115,150,20);

}
  public void actionPerformed(ActionEvent ae)
{

   if(ae.getSource()==bt1)
  {
  try{
  FileOutputStream fout=new FileOutputStream("e:/work/my.txt");
  DataOutputStream dout=new DataOutputStream(fout);
  dout.writeUTF("Hello, File World!");
  dout.close();
```


Библиотека БГУИР

```
}
 catch(Exception ex){}
  }
   else
    if(ae.getSource()==bt)
   System.exit(0);

 else
  if(ae.getSource()==bt2)
  {
  try{
  FileInputStream finp= new FileInputStream("e:/work/my.txt");
    DataInputStream dinp=new DataInputStream(finp);
    String s=dinp.readUTF();
    dinp.close();
    tf.setText(s);
     }
catch(Exception ex) {}
}}

public static void main(String[] args)
{
 new FileExample();}}
```

Для записи строк в формате `ASCII` используется поточный класс `PrintStream`, как показано ниже:

```
try{
   FileOutputStream fout=new FileOutputStream("e:/work/my.txt");
   PrintStream pstr=new PrintStream(fout);
   pstr.println("Hello, File World");
   pstr.close();
}
 catch(Exception ex){}
  }
 else
  if(ae.getSource()==bt2)
  {
  try{
    FileInputStream finp= new FileInputStream("e:/work/my.txt");
    DataInputStream dinp=new DataInputStream(finp);
    String s=dinp.readLine();
    dinp.close();
    tf.setText(s);
  }
catch(Exception ex) {}
}
```

Библиотека БГУИР

Несколько изменим наше приложение, с тем чтобы выбирать файл из окна файлового диалога. Окно файлового диалога создается и отображается следующим образом:

```
FileDialog fd=new FileDialog(this, "FileOpening");
fd.show();
```

В конструкторе `FileDialog` указывается режим открытия файла (`FileOpening`). Имя выбранного файла получаем с помощью команды

```
String s=fd.getFile();
```

Для произвольного доступа используется класс `RandomAccessFile`. Для доступа к данным используем метод `seek(long position)`. Положение указателя возвращает метод `long getFilePointer()`. Пример записи в файл целых чисел командой `writeInt` дает следующий код:

```
package mylabs;
import java.awt.*;
import java.awt.event.*;
import java.io.*;

   class lab1_10 extends Frame implements ActionListener
{
  Button b=new Button("Exit");
  Button b1=new Button("Write to File");
  Button b2=new Button(" Read from  File");

  public  lab1_10()
{ setLayout(null);
  setBackground(new Color(240,230,100));
  setSize(300,300);
  setVisible(true);
  add(b);
  add(b1);
  add(b2);
  b.addActionListener(this);
  b1.addActionListener(this);
  b2.addActionListener(this);
  b.setBounds(20,30,100,20);
  b1.setBounds(20,60,100,20);
  b2.setBounds(20,90,100,20);
}
  public void actionPerformed(ActionEvent ae)
{
   if(ae.getSource()==b)
   System.exit(0);
   else
   if(ae.getSource()==b1)
  {
```

Библиотека БГУИР

```
  try{
   RandomAccessFile raf= new RandomAccessFile("temp.dat","rw");
  for (int i=0;i<10;i++)
   raf.writeInt(i);
   raf.close();}
 catch(Exception ex){}}
 else
  if(ae.getSource()==b2)
  {
  try{RandomAccessFile raf=
 new RandomAccessFile("temp.dat","rw");
    Graphics g=getGraphics();
    for (int i=0;i<10;i++)
       {
         raf.seek(i*4);
         int z=raf.readInt();
         g.drawString(""+z,130,50+20*i);}
    raf.close(); }
catch(Exception ex) {}}}}
public class lab1_11{
public static void main(String[] args)
{lab1_10  app=new lab1_10();}}
```

Для записи в файл используем фрагмент

```
  try{
   RandomAccessFile raf= new RandomAccessFile("temp.dat","rw");
  for (int i=0;i<10;i++)
   raf.writeInt(i);
   raf.close();}
 catch(Exception ex){}}
```

Запись целых чисел выполняет метод `writeInt`.

Для записи чисел в формате с плавающей точкой используют метод `writeFloat`. Для записи чисел с фиксированной точкой используют метод `writeDouble`. Для записи строки в формате `Unicode` – метод `writeUTF`. Чтение осуществляется соответственно методами `readInt`, `readFloat`, `readDouble`, `readUTF`. Однако чтение нужно выполнять с позиционированием головки чтения/записи с помощью метода `seek`. Метод `seek` устанавливает указатель на указанное смещение в байтах относительно начала. Целое число занимает 4 байта. Поэтому выполняем умножение `i*4`.

Библиотека БГУИР

ЛИТЕРАТУРА

1 Хабибуллин, И. Создание распределенных приложений на Java 2. / И. Хабибуллин. – СПб. : БХВ-Петербург, 2002. – 692 с.
2 Перроун, П. Дж. Создание корпоративных систем на основе Java 2 Enterprise Edition. Руководство разработчика / П. Дж. Перроун, Венката С. Р. Кришна, Р. Чаганти. – М. : Изд. дом «Вильямс», 2001. – 1184 с.
3 Цимбал, А. А. Технологии создания распределенных систем. Для профессионалов / А. А. Цимбал, М. Л. Аншина. – СПб. : Питер, 2003. – 576 с.
4 Чапел, Л. TCP/IP. Учебный курс / Л. Чапел, Э. Титтел. – СПб. : БХВ-Петербург, 2003. – 976 с.
5 Хеффельфингер, Д. Java EE и сервер приложений GlassFish 3 / Д. Хеффельфингер. – М. : ДМК, 2013. – 416 с.
6 Герман, О. В. Java и интернет-бизнес / О. В. Герман, Ю. О. Герман. – Минск : Бестпринт, 2010. – 384 с.
7 Герман, О. В. Программирование на Java и c# для студента / О. В. Герман, Ю. О. Герман. – СПб. : БХВ-Петербург, 2005. – 512 с.
8 Холл, М. Программирование для WEB / М. Холл, Л. Браун – М. : Изд. дом «Вильямс», 2002. – 1264 с.
9 Секреты программирования для Интернет на Java / М. Томас [и др.] [Электронный ресурс]. – Режим доступа : www. Books-Shop.com.
10 Блинов, И. Н. Java. Промышленное программирование / И. Н. Блинов, В. С. Романчик. – Минск : УниверсалПресс, 2007. – 704 с.
11 Олифер, В. Г. Сетевые операционные системы / В. Г. Олифер, Н. А. Олифер. – СПб. : Питер, 2001. – 544 с.
12 Гарнаев, А. Ю. Excel, VBA, Internet в экономике и финансах / А. Ю. Гарнаев. – СПб. : БХВ-Петербург, 2002. – 816 с.
13 Кровчик, Э . NET. Сетевое программирование для профессионалов / Э. Кровчик, В. Кумар. – М. : Лори, 207. – 417 с.
14 Мак-Дональд, М. ASP.NET 4 с примерами на c# 2010 для профессионалов / М. Мак-Дональд, А. Фримен, М. Шпушта. – М. : Изд. дом «Вильямс», 2011. – 1424 с.

Библиотека БГУИР

Св. план 2016, поз. 27

Учебное издание

Герман Олег Витольдович
Герман Юлия Олеговна

АДМИНИСТРИРОВАНИЕ И ПРОГРАММИРОВАНИЕ РАСПРЕДЕЛЕННЫХ ПРИЛОЖЕНИЙ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Библиотека БГУИР

Редактор *Е. И. Герман*
Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *А. В. Бас*

Подписано в печать 01.11.2016. Формат 60x84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л.   Уч.-изд. л. 15,0. Тираж 120 экз. Заказ 22.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
ЛП №02330/264 от 14.04.2014.
220013, Минск, П. Бровки, 6