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

Кафедра экономической информатики

РАСПРЕДЕЛЕННЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ

Лабораторный практикум
для студентов специальности 1-40 01 02-02
«Информационные системы и технологии (в экономике)»
всех форм обучения

Библиотека БГУИР

Минск БГУИР 2012

УДК 004.75(076.1)
ББК 32.973.202я73
Р24

А в т о р ы:
В. Н. Комличенко, Е. Н. Унучек,
А. О. Комаровский, В. М. Кузьмицкий

Р е ц е н з е н т:
доцент кафедры «Интеллектуальные информационные технологии»
учреждения образования «Белорусский государственный университет
информатики и радиоэлектроники»,
кандидат технических наук М. Д. Степанова

Р24 **Распределенные** информационные системы : лаб. практикум для студ. спец. 1-40 01 02-02 «Информационные системы и технологии (в экономике)» всех форм обуч. / В. Н. Комличенко [и др.]. – Минск : БГУИР, 2012. – 74 с. : ил.
ISBN 978-985-488-794-4.

Представлены краткие теоретические сведения, методические указания и варианты индивидуальных заданий, необходимые для разработки web-сервисов на базе языка JAVA, а также рассмотрены задания по разработке консольных, графических и web-приложений на базе платформы .NET.

УДК 004.75(076.1)
ББК 32.973.202я73

ISBN 978-985-488-794-4

© УО «Белорусский государственный университет информатики и радиоэлектроники», 2012

Библиотека БГУИР

Содержание

ЛАБОРАТОРНАЯ РАБОТА №1. РАЗРАБОТКА WEB-СЕРВИСА, РЕАЛИЗУЮЩЕГО RPC-ОРИЕНТИРОВАННОЕ ВЗАИМОДЕЙСТВИЕ....... 4

1.1. Теоретическая часть 5
1.2. Практическая часть... 7
Индивидуальные задания 10
Вопросы для самопроверки.. 10

ЛАБОРАТОРНАЯ РАБОТА №2. РАЗРАБОТКА WEB-СЕРВИСА, РЕАЛИЗУЮЩЕГО ДОКУМЕНТООРИЕНТИРОВАННОЕ ВЗАИМОДЕЙСТВИЕ 11

2.1. Теоретическая часть 12
2.2. Практическая часть... 16
Индивидуальные задания 28
Вопросы для самопроверки: 29

ЛАБОРАТОРНАЯ РАБОТА №3. РАЗРАБОТКА КОНСОЛЬНОГО ПРИЛОЖЕНИЯ НА ЯЗЫКЕ C#.. 30

3.1. Теоретическая часть 30
3.2. Практическая часть... 33
Индивидуальные задания 37
Вопросы для самопроверки.. 38

ЛАБОРАТОРНАЯ РАБОТА №4. РАЗРАБОТКА GUI ПРИЛОЖЕНИЯ В АРХИТЕКТУРЕ КЛИЕНТ–СЕРВЕР НА ЯЗЫКЕ C# 39

4.1. Теоретическая часть 39
4.2. Практическая часть... 40
4.2.1. Серверная часть программы 40
4.2.2. Клиентская часть программы 45
Индивидуальные задания 47
Вопросы для самопроверки.. 47

ЛАБОРАТОРНАЯ РАБОТА №5. РАЗРАБОТКА ПРИЛОЖЕНИЯ, ИСПОЛЬЗУЮЩЕГО ТЕХНОЛОГИЮ ADO.NET 49

5.1. Теоретическая часть 49
5.1.1. Преимущества и нововведения в ADO.NET 49
5.1.2. Практическое применение ADO.NET.................................. 50
5.2. Практическая часть... 55
Индивидуальные задания 60
Вопросы для самопроверки.. 61

ЛАБОРАТОРНАЯ РАБОТА №6. РАЗРАБОТКА WEB-ПРИЛОЖЕНИЯ, ИСПОЛЬЗУЮЩЕГО ТЕХНОЛОГИЮ ASP.NET.. 62

Библиотека БГУИР

6.1. Теоретическая часть 62
6.1.1. Обзор ASP.NET Framework 62
6.1.2. Обзор web-форм 63
6.1.3. Структура проекта и ASPX-файла 65
6.1.4. Жизненный цикл страниц в ASP.NET 67
6.2. Практическая часть 68
Индивидуальные задания 71
Вопросы для самопроверки 72
ЛИТЕРАТУРА 73

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №1
РАЗРАБОТКА WEB-СЕРВИСА, РЕАЛИЗУЮЩЕГО RPC-ОРИЕНТИРОВАННОЕ ВЗАИМОДЕЙСТВИЕ

Цель: научиться работать с технологией web-сервисов, используя RPC-ориентированный подход.

1.1. Теоретическая часть

Web-сервисы – это часть бизнес-логики приложения, размещенной в Internet, которая обеспечивает доступ по стандартным Интернет-протоколам, таким как HTTP или SMTP.

Главные технологии web-сервисов:

–*SOAP (Simple Object Access Protocol)* – стандарт «упаковки» XML-документов для транспортировки по протоколам SMTP, HTTP или FTP;

–*WSDL (Web Service Description Language)* – это XML-технология, описывающая интерфейсы web-сервисов. Помогает клиентам автоматически понять, как работать с сервисом;

–*UDDI (Universal Description, Discovery and Integration)* – обеспечивает всемирную регистрацию web-сервисов. Используется для обнаружения сервиса путем поиска по имени, категории и др.

Взаимодействие между технологиями:

- Клиент запрашивает у репозитория UDDI сервис по его имени или идентификатору.
- Клиент получает информацию о размещении WSDL-документа от UDDI-репозитория. Он содержит информацию о том, как связаться с сервисом, и формат запроса в XML.
- Согласно найденной в WSDL информации, клиент создает SOAP-сообщение и посылает на хост сервиса.

Провайдеры публикуют информацию (метаданные) о сервисе в репозитории. Согласно документу Web Services Conceptual Architecture компании IBM, для публикации используются несколько различных механизмов.

Библиотека БГУИР

Прямой

Инициатор запроса ищет описание сервиса прямо у провайдера сервиса, используя e-mail, FTP, CD. Провайдер доставляет описание сервиса и одновременно делает сервис доступным для инициатора запроса. Как такового, репозитория (брокера) здесь нет.

HTTP Get-запрос

Этот механизм используется на www.xmethods.com. Это публичный репозиторий web-сервисов. Инициатор запросов ищет описание сервиса прямо у провайдера, используя Get-запрос по протоколу HTTP. Эта модель имеет регистр (web-репозиторий), хотя только в неполном смысле этого слова.

Динамическое обнаружение

Этот механизм использует локальные публичные репозитории, чтобы хранить и программно отыскивать описания сервисов. Наиболее часто используемый репозиторий – UDDI, хотя есть и другие (например ebXML R). Эта модель наиболее универсальная и представляет наибольший интерес.

В данной лабораторной работе мы не будем использовать UDDI, т. е. механизм публикации прямой.

Apache Axis (Apache eXtensible Interaction System) – система для конструирования SOAP-процессоров, таких, как клиенты, серверы, шлюзы и др. SOAP – это механизм для коммуникации приложений посредством Интернет. Однако Axis не просто «движок» SOAP, он также включает:

- простой самостоятельный сервер;
- сервер, встраиваемый в контейнеры сервлетов;
- расширенную поддержку WSDL;
- инструменты, генерирующие Java-классы из WSDL;
- примеры программ;
- инструмент для отслеживания TCP/IP-пакетов.

Axis конвертирует Java-объекты в данные SOAP, когда посылает их по сети или получает результаты. Все ошибки, генерируемые сервером, Axis преобразовывает в Java-исключения.

Axis реализует JAX-RPC API – один из стандартных способов программирования Java-сервисов.

Axis поставляется в виде набора jar-библиотек. Реализация JAX-RPC API представлена в файлах *jaxrpc.jar* и *saaj.jar*. Также для работы приложений необходимы вспомогательные библиотеки для логгирования, обработки WSDL. Для работы Axis'а требуется наличие XML-парсера, совместимого с JAXP 1.1 (Java API for XML Processing), предпочтительно Xerces.

Библиотека БГУИР

1.2. Практическая часть

В качестве примера рассмотрим следующую задачу:

Разработать телефонный справочник, позволяющий проводить поиск по имени или номеру телефона.

Для выполнения задачи воспользуемся Apache Axis 1.3 и Tomcat 7.0. Распаковав дистрибутив, скопируем папку axis (находится в папке webapps) в папку Tomcat 7.0\webapps\. Проследите, чтобы в папке Tomcat 7.0\common\lib\ лежали библиотеки activation.jar, mail.jar, xmlsec-1.3.0.jar, jsse.jar – это опциональные компоненты, необходимые системе axis. Теперь запустим Tomcat. Чтобы проверить, правильно ли настроен axis, в строке браузера наберем http://localhost:8080/axis, после чего должна появиться приветственная страница axis. По ссылке Validation можно проверить, все ли библиотеки установлены. По ссылке View – просмотреть уже сконфигурированные web-сервисы.

Создадим класс сервиса – phone.java. В нем будут находиться две функции – *String phone_name(String phone),* которая по номеру телефона возвращает фамилию и *public String name_phone(String name)*, которая по фамилии возвращает номер телефона.

```
    //импорт библиотек
import java.io.*;
import java.util.*;

    //заголовок класса
public class Phone{

    //первая функция
    public String phone_name(String phone){
        String name= new String();;
        String line;
```

Создадим файл, в котором в каждой строчке записана пара фамилия–телефон через пробел. В функции FileReader замените путь на свой.

```
        try{
    FileReader fr =
    new FileReader("D:/work/phone/phone/phones.txt");
        BufferedReader in = new BufferedReader(fr);
```

Предусмотрим ситуацию, при которой соответствующего номера может не оказаться в файле. Для анализа прочитанной с файла строки удобно использовать функцию split, которая расщепляет строку на слова. Аргумент – это разделительный символ. Функция возвращает массив искомых слов.

```
            while ((line=in.readLine())!=null){
            String[] str = line.split(" ");
            if (str[0].equals(phone)){ name=str[1]; break;}else
```


Библиотека БГУИР

```
                    name="NO SUCH TELEPHONE";
                    }
            }catch(IOException e){
                    e.printStackTrace();
            }
            return name;
    }
```
Вторая функция (аналогично).
```
    public String name_phone(String name){
            String phone= new String();;
            String line;
            try{
            FileReader fr =
    new FileReader("D:/work/phone/phone/phones.txt");
            BufferedReader in = new BufferedReader(fr);
            while ((line=in.readLine())!=null){
        String[] str = line.split(" ");
        if (str[1].equals(name)){ phone=str[0]; break;}else
            phone="NO SUCH NAME";
            }
            }catch(IOException e){
            e.printStackTrace();
            }
            return phone;
    }
}
```

Данный файл переименуем в phone.jws и поместим в папку axis на Tomcat. Теперь Tomcat необходимо перезапустить. Класс должен откомпилироваться и файл с расширением class поместится в jwsClasses – папку, которая тоже создастся в папке axis/web-inf.

Клиентское приложение

Подключим библиотеки – некторые из них находятся в axis.jar: пакете, который размещается в папке axis/web-INF/lib.

```
import org.apache.axis.client.Service;
import org.apache.axis.client.Call;
import javax.xml.rpc.ServiceException;
import java.net.URL;
import java.net.MalformedURLException;
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;
```


Библиотека БГУИР

Функция main выбрасывает исключения, вызванные вызовом несуществующего сервиса и неправильно сформированным URL.

```
class phoneApp {
public static void main(String[] args) throws ServiceException, MalformedURLException {
```

Строка endpoint является строкой URL, по которому размещен сервис. Обратите внимание на название хоста, номер порта и название самого сервиса, которые соответствуют нашей реализации лабораторной работы.

```
String endpoint = "http://localhost:8080/axis/Phone.jws";
```

Следующие строки создают объекты сервиса, вызова именно нашего сервиса, и устанавливается целевой адрес сервиса через класс URL:

```
Service service = new Service();
Call call = (Call)service.createCall();
call.setTargetEndpointAddress(new URL(endpoint));
```

На экран пользователя выведется меню. Затем запускаем цикл, который останавливается, как только пользователь нажмет на «3», что в нашем меню означает «выход».

```
System.out.println("1 - enter the phone number");
System.out.println("2 - enter the name");
System.out.println("3 - exit");
try {
BufferedReader in = new BufferedReader
(new InputStreamRea er(System.in));
String line = null;
line = in.readLine();
while(!line.equals("3")){
if (line.equals("3")) break;
```

При нажатии на «1» прочитаем строку с телефоном с консоли.

```
if (line.equals("1")){
String phone = in.readLine();
```

Сформируем массив объектов, состоящий из одного объекта – введенной строки.

```
Object[] param1 = new Object[]{phone};
```


Библиотека БГУИР

Вызов сервиса осуществляется с помощью функции *invoke()* с параметрами «название функции» и массивом передаваемых значений.

```
String response = (String)call.invoke("phone_name", param1);
```

Выведем возвращенную строку на экран:

```
System.out.println("PHONE="+phone+"\n"+"NAME="+response);
```

Аналогично обработаем нажатие на кнопку «2».

```
if (line.equals("2")){
String name = in.readLine();
Object[] param2 = new Object[]{name};
String response = (String)call.invoke("name_phone", param2);
System.out.println("NAME="+name+"\n"+"PHONE="+response);
 };
```

Закончим клиентское приложение, повторив вывод меню и ввод значения, которое будет обрабатываться уже на следующей итерации цикла.

```
System.out.println("1 - enter the phone number");
System.out.println("2 - enter the name");
System.out.println("3 - exit");
line = in.readLine();
}
} catch (IOException e) {
      e.printStackTrace();
} }}
```

Для запуска всей системы в целом должен быть запущен Tomcat с axis'ом внутри, затем – клиентский класс. Чтобы компилятор мог использовать подключенные библиотеки, их jar-файлы должны быть прописаны в CLASSPATH.

Индивидуальные задания

Написать web-сервис, реализующий функционал приложения в соответствии с вариантом предметной области из лабораторной работы №4, посредством которого клиент сможет выполнять операции просмотра, добавления, удаления и поиска информации. Данные должны храниться в массиве на сервере.

Вопросы для самопроверки:

1. Понятие web-сервиса.
2. Как происходит взаимодействие между технологиями web-сервисов?
3. Для чего предназначена SOAP-технология?
4. Технологии web-сервисов.

Библиотека БГУИР

5. Какие механизмы используются для публикации информации о сервисе в регистре?
6. Когда функция main выбрасывает исключения?
7. Для чего предназначена функция invoke()?
8. Для чего предназначена WSDL-технология?
9. Понятие UDDI.
10. Какие протоколы для транспортировки использует *SOAP?*
11. Что такое RPC?

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №2 РАЗРАБОТКА web-СЕРВИСА, РЕАЛИЗУЮЩЕГО ДОКУМЕНТООРИЕНТИРОВАННОЕ ВЗАИМОДЕЙСТВИЕ

Цель: научиться работать с технологией web-сервисов, используя документоориентированный подход.

2.1. Теоретическая часть

В предыдущей лабораторной работе мы создали web-сервис, используя модель RPC. В этой лабораторной работе документы XML будут являться основным инструментом взаимодействия между вызывающим объектом и web-сервисом. Этот тип SOAP-разработки называется документоориентированным (*document-style*) подходом.

В случае документоориентированного программирования клиент передает web-сервису документ XML, который обрабатывается на сервере. Здесь также (как и в первой лабораторной работе) на макроуровне клиент передает запрос SOAP и получает ответ SOAP (рис. 2.1):

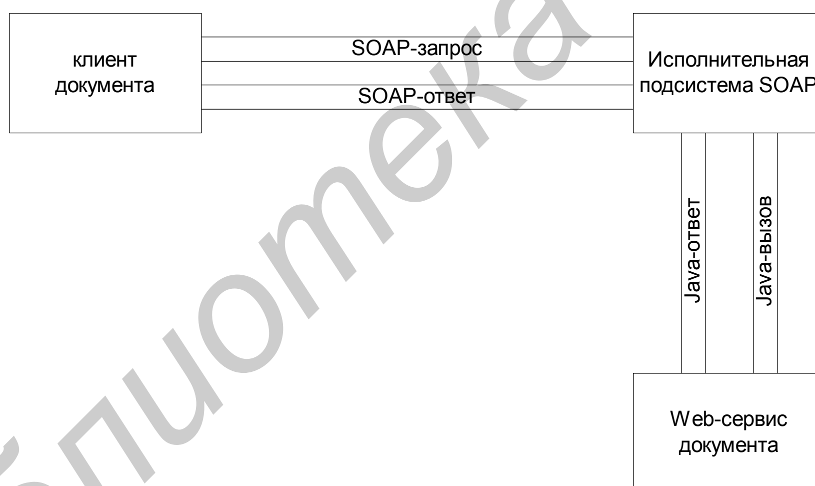

Рис. 2.1. Схема документоориентированного взаимодействия

Чтобы лучше увидеть различия между двумя подходами, рассмотрим их принцип действия подробнее (рис. 2.2 и 2.3):

Библиотека БГУИР

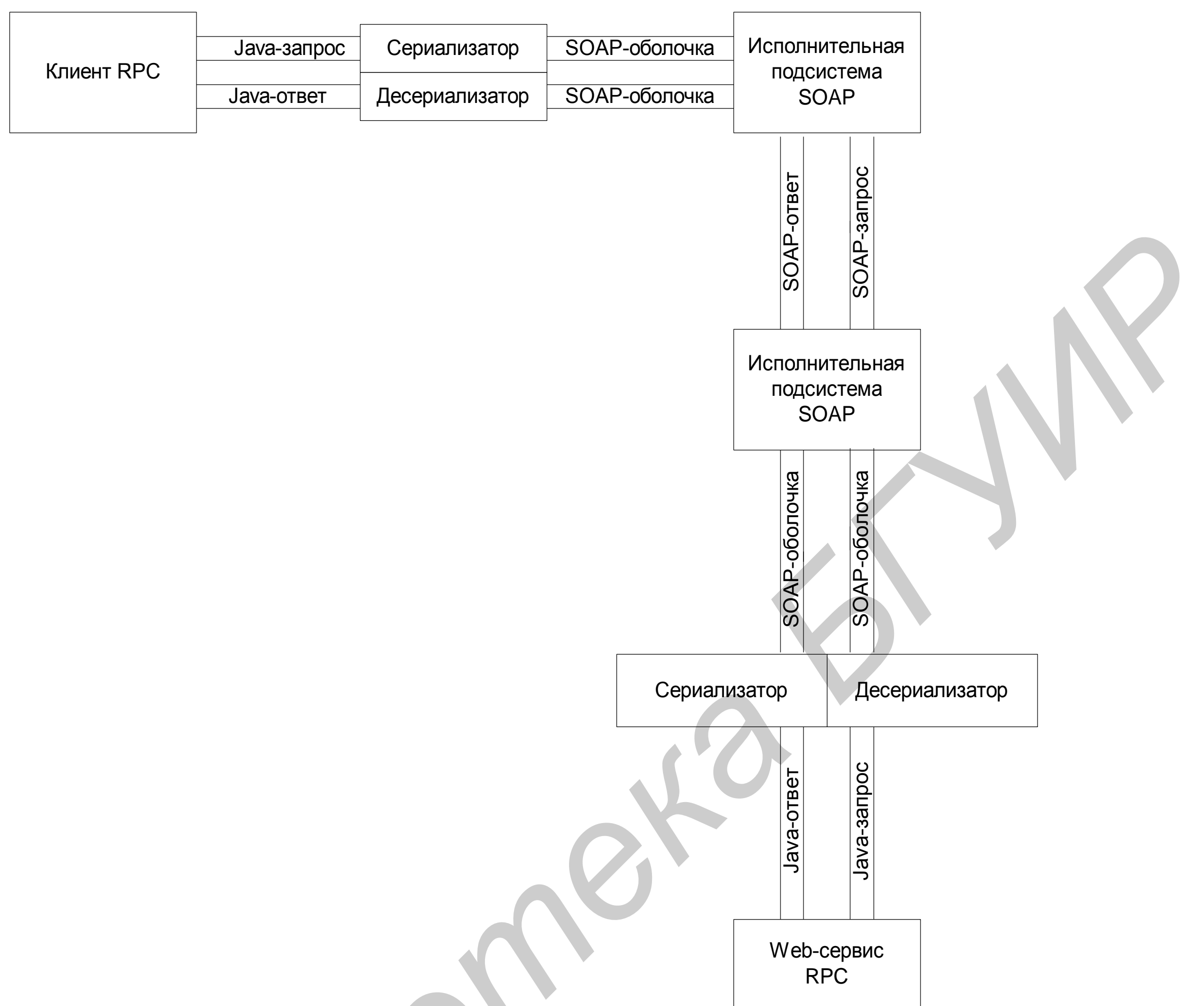

Рис. 2.2. Модель взаимодействия клиента и сервиса при RPC-подходе

Библиотека БГУИР

Рис. 2.3. Модель взаимодействия клиента и сервиса
при документоориентированном подходе

Библиотека БГУИР

Основные различия выражаются в механизмах сериализации/десериализации и семантики.

В документоориентированном SOAP-программировании клиенту не нужно сериализовать вызов Java и его аргументы в документ XML. И наоборот, серверу не нужно десериализовать документ XML в вызов Java и типы данных. В RPC-SOAP-программировании клиент и сервер должны придерживаться строго определенной программной модели: клиент вызывает метод с возможными аргументами, а сервер в ответ возвращает одно значение. В документоориентированном программировании в обмене участвует документ XML и значение каждого элемента интерпретируется участвующими во взаимодействии сторонами. Вдобавок к этому клиент и web-сервис могут объединить в запрос и ответ несколько документов. Так как процесс сериализации/десериализации отсутствует, приложения, разработанные посредством использования документоориентирванного программирования, должны быть более быстрыми, чем их RPC-аналоги. Тем не менее это не обязательно будет так. Это можно аргументировать тем, что при документоориентированном программировании сериализация зависит от разработчика, т. к. в некоторый момент нам может понадобиться преобразовать наши данные Java в XML и наоборот. А также задействованные документы XML потенциально могут быть намного больше, чем простые типы, используемые в RPC-модели.

Документоориентированное SOAP-программирование подходит, когда мы хотим осуществлять обмен данными между двумя и более сторонами. Это особенно справедливо в случае, когда у нас уже есть документ XML, представляющий данные, т.к. мы можем обмениваться самим документом XML в исходном виде без необходимости преобразовывать его структуры данных Java.

Отсутствие шагов сериализации/десериализации упрощает разработку и ускоряет обработку данных.

RPC-метод разработки SOAP срабатывает очень хорошо, если проблему можно легко смоделировать с помощью вызовов методов или если у нас уже есть функциональный API. Для приложений, которые манипулируют документами XML с помощью XSLT, документ XML более предпочтителен, чем структуры данных Java, т. к. процессор XSLT может автоматически преобразовывать этот документ.

SAAJ (SOAP with Attachments API for Java) используется главным образом для обмена SOAP-сообщениями в рамках JAX-RPC и JAXR-реализаций. К тому же это такой API, который разработчики могут использовать, если они хотят писать SOAP-приложения напрямую, нежели через JAX-RPC.

SAAJ API удовлетворяет спецификациям SOAP 1.1 и SOAP 1.2 и спецификации SOAP with Attachments.

Рассмотрим SAAJ применительно к двум аспектам: сообщения (*messages*) и соединения (*connections*).

SAAJ сообщения следуют стандартам SOAP. Существует два типа SOAP-сообщений: с прикреплениями (*with attachments*) и без. Мы использовали SOAP без прикреплений. Высокоуровневая структура таких сообщений показана на рис. 2.4.

Рис. 2.4. Объект SOAPMessage без прикреплений

SAAJ API предоставляет класс *SOAPMessage* для представления SOAP-сообщения, *SOAPPart* для представления части SOAP, *SOAPEnvelope* – для SOAP-конверта и т. д.

Библиотека БГУИР

Когда вы создаете объект *SOAPMessage,* он автоматически имеет разделы, которые требуются для SOAP-сообщения. Другими словами, в объект *SOAPMessage* входит объект *SOAPPart*, который содержит объект *SOAPEnvelope*. В свою очередь объект *SOAPEnvelope* автоматически содержит пустой *SOAPHeader* (заголовок), за которым следует пустой *SOAPBody* (тело).

Объект заголовка *SOAPHeader* может включать один или более заголовков, которые содержат метаданные о сообщении.

Все SOAP-сообщения отправляются и принимаются посредством соединения (*connection*). В SAAJ API соединение представлено объектом *SOAPConnection*, который связывает отправителя прямо с пунктом назначения. Такой тип соединения называется *point-to-point соединение*, потому что оно связывает две конечных точки. Сообщения, отправляемые с использованием SAAJ API, называются *request-response сообщения*. Они отправляются через объект *SOAPConnection* методом *call*, который отправляет сообщение (запрос), а затем блокируется в ожидании ответа.

Apache Axis вообще предусматривает 4 стиля сервисов. Кроме *RPC, Document*, это еще *Wrapped* и *Message*. Стили необходимо указывать в дескрипторе развертывания. В рамках лабораторной работы будем использовать стиль *Message* (он работает с XML-данными, как они написаны, не превращая их в Java-объекты), а не *Document*, как можно было подумать (*Document* привязывает Java-объекты к XML, так что разработчик имеет дело с Java-объектами, а не прямо с XML-конструкцией). Для объявления метода сервиса в стиле *message* существуют четыре действительные сигнатуры:

```
public Element [] method(Element [] bodies);
public SOAPBodyElement [] method (SOAPBodyElement [] bodies);
public Document method(Document body);
public void method(SOAPEnvelope req, SOAPEnvelope resp).
```

Теперь у нас достаточно теоретического основания для выполнения лабораторного задания.

2.2. Практическая часть

Задание

На сервере (например в папке D:\work\server) находится XML-документ, содержащий информацию о наличии товаров в магазине. Клиент отправляет следующий запрос к web-сервису:

```
<getfile>
    <file>ИМЯ_ФАЙЛА</file>
</getfile> ,
```

где ИМЯ_ФАЙЛА вводится пользователем. Web-сервис должен отправить клиенту файл с таким именем из сервера.

Библиотека БГУИР

1. Составление XML-документа с данными

Для начала необходимо составить XML-документ с данными, который должен храниться на сервере. Например такой:

```
<?xml version="1.0" encoding="windows-1251"?>
<?xml-stylesheet href="goods.xsl" type="text/xsl"?>

	<goods>
		<good ID="GSM">
			<type ID="Nokia">
				<code>100001</code>
				<model>3310</model>
				<price>40</price>
			</type>
			<type ID="Sony Ericsson">
				<code>100002</code>
				<model>T630</model>
				<price>150</price>
			</type>
		</good>
		<good ID="TV">
			<type ID="Horizont">
				<code>100003</code>
				<model>modellll</model>
				<price>170</price>
			</type>
			<type ID="Vityaz">
				<code>100004</code>
				<model>Europe</model>
				<price>185</price>
			</type>
		</good>
		<good ID="computer">
			<type ID="Dell">
				<code>100005</code>
				<model>Computers</model>
				<price>999</price>
			</type>
			<type ID="Apple">
				<code>100006</code>
				<model>Macintosh</model>
				<price>1999</price>
			</type>
```

Библиотека БГУИР

```
        </good>
    </goods>
```

Сохраним его в файл D:\work\server\goods.xml.

Здесь приведем текст XSL-файла, который используется для визуализации XML-документа. Хотя для выполнения задания он не нужен, однако может пригодиться для проверки правильности составления XML-документа, выполнения дополнительного задания преподавателя и т. д.

D:\work\server\goods.xsl имеет вид:

```
<?xml version="1.0" encoding="windows-1251"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>

<xsl:template match="/">
 <h1><b><center><font color="#ff0000">Товары:</font></center></b></h1>
    <table border="5">
    <xsl:for-each select="/goods/good">
     <tr bgcolor="cyan"><td colspan="3" width="600">
     <h2><font color="brown"><xsl:value-of select="@ID"/></font></h2>
     </td></tr>

     <xsl:for-each select="type">
      <tr>
         <td colspan="3" width="600"><center><h3><xsl:value-of
                              select="@ID"/></h3></center></td>
      </tr>
      <tr>
         <td><xsl:value-of select="code"/></td>
         <td><xsl:value-of select="model"/></td>
         <td><xsl:value-of select="price"/></td>
      </tr>
    </xsl:for-each>
    </xsl:for-each>
    </table>
</xsl:template>
</xsl:stylesheet>
```

2. Написание web-сервиса StoreService.java

Теперь напишем web-сервис StoreService.java, выбрав четвертую сигнатуру (по желанию, можно любую другую).

```
import org.apache.axis.message.SOAPBodyElement;
```


Библиотека БГУИР

```
import org.apache.axis.message.SOAPEnvelope;
import org.apache.axis.AxisFault;
import org.apache.axis.MessageContext;
import org.w3c.dom.*;
import org.w3c.dom.Node;
import org.w3c.dom.Text;
import org.xml.sax.SAXException;

import javax.xml.transform.TransformerException;
import javax.xml.soap.*;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import java.io.IOException;
```

Опишем класс web-сервиса. Главный его метод – это *storeService()*. Он получает запрос в виде SOAP-сообщения и автоматически формирует ответное SOAP-сообщение. Однако мы должны ответ изменить в соответствии с заданием, т. е. записать в него искомый XML-документ. Хотя сигнатура метода оперирует типами *SOAPEnvelope*, а не *SOAPMessage*, эти типы легко преобразуются один к другому. Мы же отталкиваемся от структуры *SOAPMessage* в связи с реализуемым стилем *Message*.

Методу *storeService()* необходимо проанализировать запрос, выделив из него имя файла, а потом составить SOAP-ответ, который и будет автоматически послан отправителю, поэтому нам удобно написать еще две функции, помимо главного метода: это *getFileName()* – она анализирует запрос и возвращает имя файла, и *createSOAPresponse()* – она формирует SOAP-ответ на основе уже предусмотренного для ответа конверта, зная имя файла с XML-документом.

```
public class StoreService {
```

Итак, выделим имя файла:

```
public String getFileName(SOAPEnvelope req) throws AxisFault,
SOAPException {
```

Из входного конверта получим тело сообщения. Для этого нам нужен его элемент. То есть корневой элемент, который для всех остальных элементов XML-запроса будет являться предком.

```
        SOAPBodyElement reqBody =  (SOAPBodyElement)
    req.getBodyElements().get(0);
      String str= "";
```


Библиотека БГУИР

Теперь в переменной *reqBody* находится структура <getfile><file>goods.xml</file></getfile>. Нам необходимо добраться до текстового узла, который в данном случае равен «goods.xml». Будем использовать иерархическую модель DOM XML-документа. При этом надо иметь в виду, что не только тег <file> является дочерним по отношению к тегу <getfile>, но и символ конца строки, символ перевода каретки, поэтому мы проверяем найденный узел: действительно ли он экземпляр класса *Element* в строке *if (currentNode instanceof Element)*.

Возьмем все дочерние узлы для <getfile>:

```
        NodeList nodes = reqBody.getChildNodes();
```

Циклом выделим узел, который является тегом <file>

```
        for (int i=0;i<nodes.getLength();i++){
       Node currentNode = nodes.item(i);
       if (currentNode instanceof Element){
```

Теперь *currentNode* равен *<file>goods.xml</file>+*.
Приведем этот узел к типу «элемент»:

```
          Element element = (Element) currentNode;
```

Теперь доберемся до дочернего текстового узла, который и является именем нужного файла. Для обозначения текстовых узлов имеется класс *Text*, поэтому делать проход с помощью циклов теперь нет необходимости. Взять сам текст у текстового узла можно с помощью функции *getData()*.

```
          Text textNode = (Text) element.getFirstChild();
     str = textNode.getData().trim();
       }
      }
```

Возвратим результат:

```
     return str;
   }
```

Следующий метод создает SOAP-конверт с ответом:

```
   public SOAPEnvelope createSOAPResponse(String fileName,
  SOAPEnvelope resp) throws SOAPException,
  ParserConfigurationException,
     IOException, SAXException {
```


Библиотека БГУИР

Java API for XML Processing (JAXP) предоставляет API для DOM-парсеров. Воспользуемся встроенными реализациями JAXP. Как альтернативу можно было использовать и классы анализатора *Xerces*. В последнем случае эту библиотеку надо скачивать отдельно или взять у преподавателя.

Класс *DocumentBuilderFactory* предоставляет фабричные методы для создания экземпляров *DocumentBuilder*. Класс *DocumentBuilder* предоставляет методы для синтаксического анализа документов XML в деревья DOM, создания новых документов XML и т. д.

```
        DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = dbFactory.newDocumentBuilder();
   Document document =
builder.parse("d:\\work\\server\\"+fileName);
```

Теперь в объекте *document* у нас находится документ, который и будет телом SOAP-ответа. Далее логика довольно проста. Мы получаем текущий контекст сообщения (не забываем, что при данной сигнатуре главного метода web-сервиса он формируется автоматически), получаем его SOAP-часть, конверт *envelope*, заголовочную часть, которую за ненадобностью отцепляем, и, наконец, раздел тела SOAP:

```
MessageContext msgCntxt = MessageContext.getCurrentContext();
SOAPMessage respMess = msgCntxt.getMessage();

SOAPPart soapPart = respMess.getSOAPPart();
SOAPEnvelope envelope = (SOAPEnvelope) soapPart.getEnvelope();
SOAPHeader header = resp.getHeader();

SOAPBody body = resp.getBody();
header.detachNode();
```

Теперь назначим корневым элементом тела сообщения корневой элемент нашего XML-документа (а он уже отождествлен с объектом *document*):

```
    SOAPBodyElement docElement = (SOAPBodyElement)
body.addDocument(document);
```

Возвратим в метод web-сервиса полученный результативный конверт:

```
    return envelope;
  }
```


Библиотека БГУИР

И, наконец, главный метод web-сервиса, согласно четвертой сигнатуре:

```
  public void storeService(SOAPEnvelope req,SOAPEnvelope resp)     throws
Exception,TransformerException, IOException {
 String fileName = getFileName(req);
 resp = createSOAPResponse(fileName, resp);
   }
}
```

3. Написание клиентского приложения *StoreMessage.java*

Общий алгоритм работы клиентского приложения выглядит так:
Ввод имени файла, который находится по договоренности в D:\work\server\

- Формирование SOAP-запроса (*createSOAPRequest()*).
- Установление соединения (*createConnection()*).
- Отправка/получение SOAP-сообщения (*call()*).
- Вывод на экран тела ответа, т. е. искомого XML-документа (*displayMessage()*).

```
import org.apache.axis.client.Call;

import javax.xml.soap.*;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPBodyElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.transform.TransformerException;

import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;
import java.net.URL;
import java.net.MalformedURLException;
import java.util.Iterator;

public class StoreMessage {

  static String fileXml;
  static SOAPMessage reqMess,respMess;
```

Составим запрос:

```
 public void createSOAPRequest() throws SOAPException,
                                              TransformerException {
```


Библиотека БГУИР

Создадим сообщение, используя объект *MessageFactory*. SAAJ API обеспечивает реализацию класса *MessageFatcory* так, чтобы упростить получение его экземпляра:

```
MessageFactory factory = MessageFactory.newInstance();
reqMess = factory.createMessage();
```

Доступ к элементам сообщения получаем аналогично тому, как объяснялось выше:

```
SOAPPart soapPart = reqMess.getSOAPPart();
SOAPEnvelope envelope = soapPart.getEnvelope();
SOAPHeader header = envelope.getHeader();
SOAPBody body = envelope.getBody();
header.detachNode();
```

Когда мы создаем новый элемент, мы должны создать объект *Name*, которым элемент будет уникально определен. Объект *Name* связан с объектами *SOAPBodyElement* и *SOAPHeaderElement*. Он может объявляться с уточнениями префикса для используемого пространства имен и URI пространства имен, помимо самого имени. Мы используем краткую форму.

```
Name bodyName = envelope.createName("GetFile");
SOAPBodyElement bodyElement = body.addBodyElement(bodyName);
Name name = envelope.createName("file");
SOAPElement fileName = bodyElement.addChildElement(name);
fileName.addTextNode(fileXml);
}
```

Функция для отображения тела документа на экран:

```
public static void displayMessage(SOAPMessage mess) throws
SOAPException{
SOAPBody body = mess.getSOAPBody();
Iterator it = body.getChildElements();
SOAPBodyElement bodyElement = (SOAPBodyElement)it.next();
System.out.println(bodyElement);
}
```

Считываем с входного потока (с консоли) имя XML-файла, который должен находиться в D:\work\server\ на той же машине, что и сервер с web-сервисом:

```
public void start(){
```

Библиотека БГУИР

```
   BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
   try {
       System.out.println("Enter the file XML:");
       fileXml = br.readLine();
      } catch (IOException e) {
       e.printStackTrace();
      }
   }
```

Главная функция – *main()*:

```
   public static void main(String[] args) throws SOAPException,  Mal-
formedURLException, TransformerException {
         StoreMessage client = new StoreMessage();
     client.start();
     client.createSOAPRequest();
     SOAPConnectionFactory soapConnectionFactory =
   SOAPConnectionFactory.newInstance();
   SOAPConnection connection =
     soapConnectionFactory.createConnection();
      URL endpoint = new
         URL("http://localhost:8080/axis/services/StoreService");
     respMess = connection.call(reqMess, endpoint);
     displayMessage(respMess);
    }
}
```

Web-сервис, реализованный для обмена сообщениями по принципу запрос–ответ, должен возвращать ответ на любое сообщение, которое он получает. Ответом является объект *SOAPMessage*, классу которого принадлежит и запрос. Некоторые сообщения могут не получать вообще никакого ответа. Сервис, который получает такое сообщение все еще должен отправить ответ, потому что необходимо разблокировать метод *call()*. В этом случае ответ не связан с содержимым сообщения – это просто сообщение для разблокировки метода *call()*.

4. Компиляция классов клиента и сервиса

Компиляцию нужно производить, подключая все задействованные библиотеки (переменную CATALINA_HOME присвойте своему месторасположению Tomcat):

set CATALINA_HOME=d:\Tomcat 7.0

Библиотека БГУИР

```
set classpath=.;%CATALINA_HOME%\webapps\axis\web-INF\ lib\    ax-
is.jar;%CATALINA_HOME%\webapps\axis\web-INF\ lib\
jaxrpc.jar;%CATALINA_HOME%\webapps\axis\web-INF\lib\commons-
logging-1.0.4.jar;%CATALINA_HOME%\webapps\axis\web-INF\
lib\commons-discovery-  0.2.jar; %CATALINA_HOME%\ webapps\ axis\ web-
INF\lib\saaj.jar; %CATALINA_HOME%\common\ endorsed\ xer-
cesImpl.jar;%CATALINA_HOME%\common\endorsed\ xmlParserAPIs.jar

javac StoreMessage.java
javac StoreService.java
```

5. Развертывание сервиса на Axis

Необходимо написать дескриптор развертывания (*StoreService.wsdd*):

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
<service name="StoreService" provider ="java:MSG">
<parameter name="className" value = "StoreService"/>
<parameter name="allowedMethods" value="storeService"/>
</service>
</deployment>
```

Теперь:

1) поместите класс web-сервиса (*StoreService.class*) в Tomcat 7.0\ webapps\axis\web-INF\classes\ в рабочей папке, в которой расположен класс web-сервиса;

2) создайте командный файл с именем, например *deploy.cmd,* с таким содержимым (переменную CATALINA_HOME присвойте своему месторасположению Tomcat):

```
set  CATALINA_HOME=d:\Tomcat 7.0
java -cp "%CATALINA_HOME%\webapps\axis\web-INF\lib\axis.jar;
%CATALINA_HOME%\webapps\axis\web-INF\lib\jaxrpc.jar;
%CATALINA_HOME%\webapps\axis\web-INF\lib\commons-logging-
1.0.4.jar;%CATALINA_HOME%\webapps\axis\web-INF\lib\commons-discovery-
0.2.jar;%CATALINA_HOME%\webapps\axis\web-INF\lib\saaj.jar;
%CATALINA_HOME%\common\lib\activation.jar;%CATALINA_HOME%\ com-
mon\lib\mail.jar" org.apache.axis.client.AdminClient StoreService.wsdd
pause
```

3) запустите Tomcat (он должен быть запущен по порту 8080);

Библиотека БГУИР

4) теперь запустите *deploy.cmd*. При успешном выполнении развертывания консольное окно должно выглядеть так, как показано на рис. 2.5;

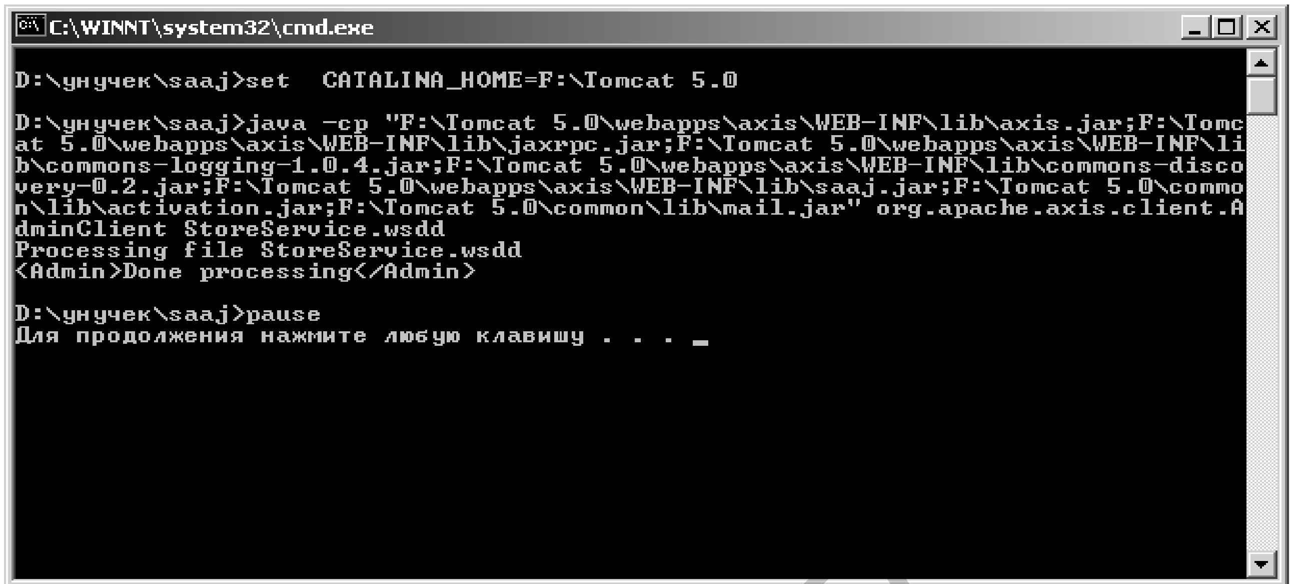

Рис. 2.5. Успешное развертывание сервиса классом AdminClient

5) после этого мы можем найти наш сервис в списке развернутых сервисов в Axis (рис. 2.6);

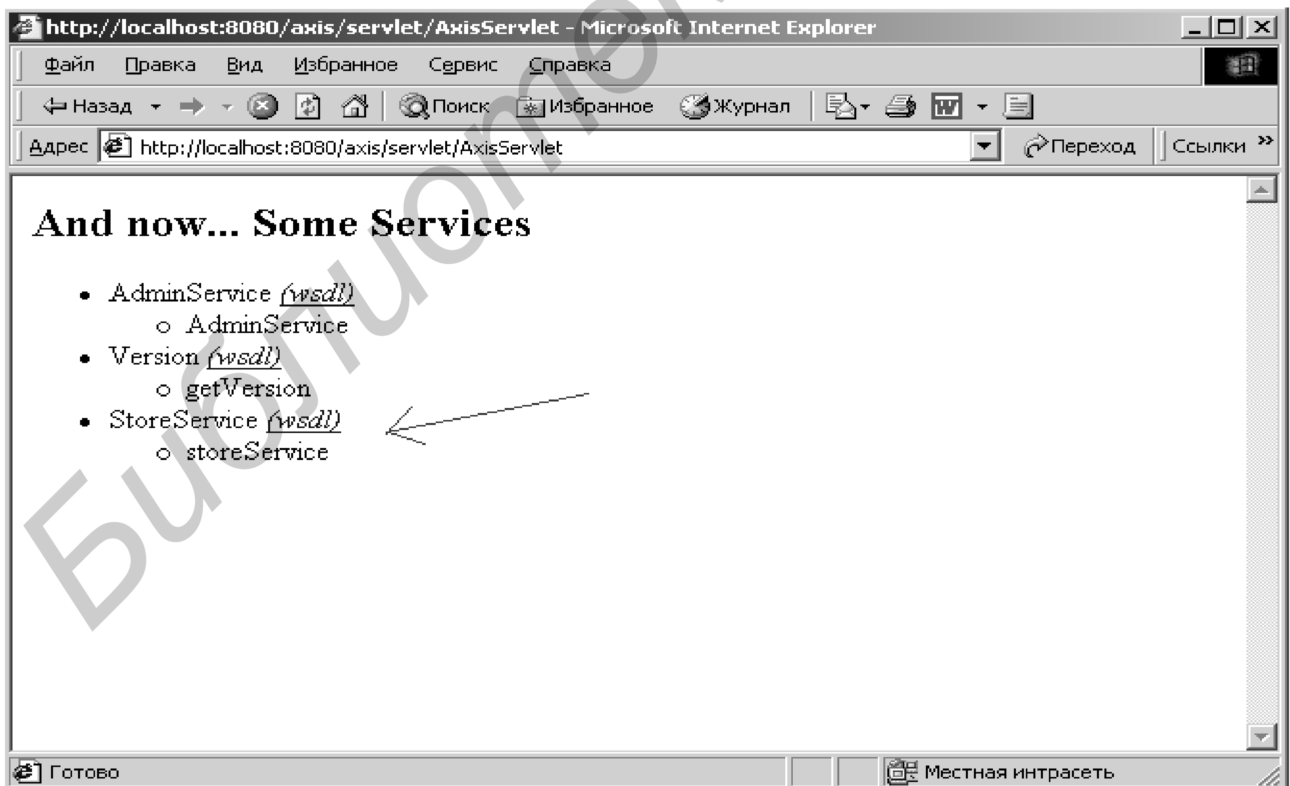

Рис. 2.6. Список развернутых сервисов

Библиотека Б

6) таким образом сервер запущен, запускаем клиента (переменную CATALINA_HOME присвойте своему месторасположению Tomcat):

set CATALINA_HOME=d:\Tomcat 7.0

set classpath=.;%CATALINA_HOME%\webapps\axis\web-INF\ lib\ ax-is.jar;%CATALINA_HOME%\webapps\axis\web-INF\lib\jaxrpc.jar; %CATALINA_HOME%\webapps\axis\web-INF\lib\commons-logging-1.0.4.jar;%CATALINA_HOME%\webapps\axis\web-INF\lib\commons-discovery-0.2.jar;%CATALINA_HOME%\webapps\axis\web-INF\lib\saaj.jar; %CATALINA_HOME%\common\endorsed\xercesImpl.jar;%CATALINA_HOME% \comon\endorsed\xmlParserAPIs.jar;%CATALINA_HOME%\common\lib\activation .jar;%CATALINA_HOME%\common\lib\mail.jar;%CATALINA_HOME%\webapp s\axis\web-INF\lib\wsdl4j-1.5.1.jar

java StoreMessage

После запуска появится приглашение ввести название XML-файла – вводим *goods.xml*. Вывод правильно работающей программы показан на рис. 2.7.

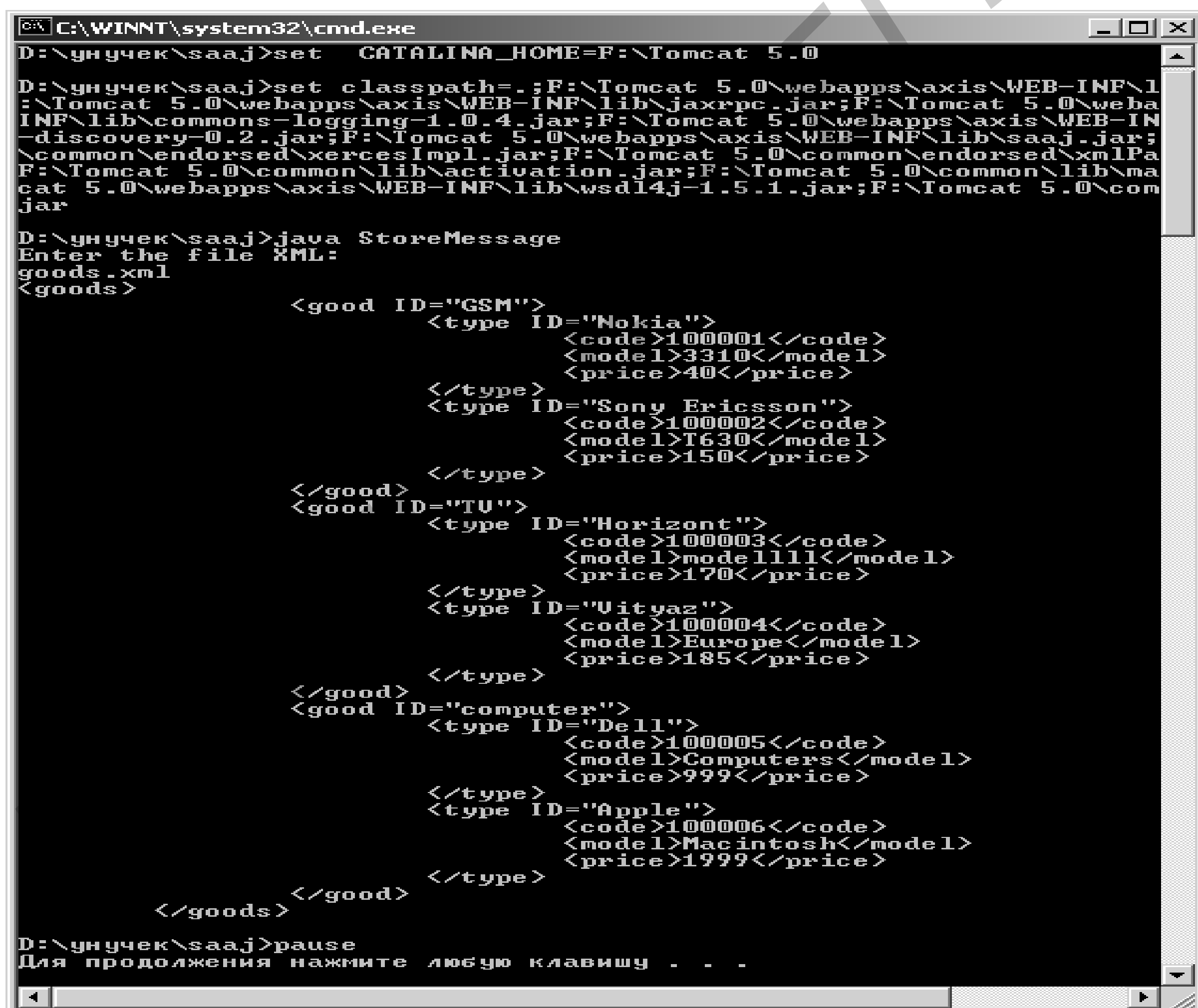

Рис. 2.7. Результат работы программы

Индивидуальные задания

1. Информация о телевизорах хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.
2. Информация о музыкальных форматах хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.
3. Информация о компьютерах хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.
4. Информация о мобильных телефонах хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.
5. Информация о сотрудниках хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.
6. Информация о книгах хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.
7. Информация о гостиницах хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.
8. Информация о курсах валют хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.
9. Информация о туристических направлениях хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.
10. Информация о предприятии хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.
11. Информация об оценках студентов хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.
12. Информация о вокзале хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.
13. Информация о больнице хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.
14. Информация о зоопарке хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.

Библиотека БГУИР

15. Информация об автомобилях хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.

Вопросы для самопроверки:

1. Что является основным инструментом взаимодействия при использовании документоориентированного подхода?
2. Как работает документоориентированное взаимодействие?
3. В чем его различие с RPC-взаимодействием?
4. Почему при документоориентирванном программировании не требуется сериализация/десериализация объектов?
5. Когда лучше применять RPC-модель, а когда документную модель?
6. Объясните структуру SOAP-сообщения.
7. Каким образом происходит соединения клиента с web-сервисом (при документоориентированном взаимодействии)?
8. Какие стили предусматривает Apache Axis? Почему message-style кажется наиболее подходящим для выполнения этой лабораторной работы?
9. Перечислите все допустимые сигнатуры методов на стороне сервиса при реализации стиля сообщений.
10. Как работает DOM-парсер? Почему мы используем цикл даже при доступе к единственному дочернему элементу?

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №3
РАЗРАБОТКА КОНСОЛЬНОГО ПРИЛОЖЕНИЯ НА ЯЗЫКЕ С#

Цель: приобрести навыки разработки консольных приложений на языке С# и ознакомиться с основными принципами разработки приложений для платформы .NET.

3.1. Теоретическая часть

Платформа разработки **.NET** Framework позволяет создавать web-сервисы, приложения Web Forms, Win32 GUI, Win32 GUI (с консольным интерфейсом пользователя), службы (управляемые Service Control Manager), утилиты и автономные компоненты.

.NET Framework – это управляемая среда для разработки и исполнения приложений, **обеспечивающая контроль типов**.

Эта среда управляет выполнением программы, она:

- выделяет память под данные и команды;
- назначает разрешения программе или отказывает в их предоставлении;
- начинает исполнение приложения и управляет его ходом;
- отвечает за освобождение и повторное использование памяти, занятой ресурсами, более не нужными программе.

.NET Framework состоит из двух основных компонентов: общеязыковой исполняющей среды (CLR) и библиотеки классов Framework Class Library (FCL).

CLR можно рассматривать как среду, управляющую исполнением кода и предоставляющую ключевые функции, такие, как компиляция кода, выделение памяти, управление потоками и сбор мусора. Благодаря использованию ***общей системы типов*** **(common type system, CTS)** CLR выполняет строгую проверку типов, а защита по правам доступа к коду позволяет гарантировать исполнение кода в защищенном окружении.

Библиотека классов **.NET Framework содержит набор полезных типов, разработанных специально для CLR** и доступных для многократного использования. Типы, поддерживаемые .NET Framework, являются объектно-ориентированными, полностью расширяемыми и обеспечивают эффективную интеграцию приложений с .NET Framework.

При компиляции кода для .NET Framework компилятор (рис. 3.1) генерирует код на общем промежуточном языке (common intermediate language, CIL), а не традиционный код, состоящий из процессорных команд.

Библиотека БГУИР

Рис. 3.1. Компиляция исходных файлов

При исполнении CLR транслирует CIL в команды процессора (отличие от JAVA).

Общеязыковая спецификация (Common Language Specification, CLS)– стандарт, который определяет минимальный набор правил, для разработчиков компиляторов, чтобы их языки интегрировались с другими. Microsoft разработала ряд таких языков: C++ с управляемыми расширениями, C #, Visual Basic .NET (сюда относятся и Visual Basic Scripting Edition или VBScript и Visual Basic for Applications или VBA), а также JScript и др.(см. рис. 3.1).

Все эти механизмы позволяют создавать собственные классы, упрощают установку приложений, предоставляют возможность работы на нескольких платформах, предоставляют возможность интеграции языков программирования и, соответственно, сервис сторонним приложениям; вследствие этого упрощается повторное использование кода и создается большой рынок готовых компонентов, реализуется автоматическое управление памятью (сбор мусора), гарантируется корректное обращение к существующим типам, единая структура обработки исключений для восстановления основного алгоритма, обеспечивается взаимодействие с существующим кодом.

Благодаря использованию общеязыковой исполняющей среды межъязыковая совместимость позволяет компонентам, реализованным с применением .NET Framework, взаимодействовать друг с другом независимо от языка, на котором они написаны. Так, приложение на Visual Basic .NET может обращаться к DLL, написанной на C#, а та, в свою очередь, способна вызвать ресурсы, созданные на управляемом C++ или любом другом .NET-совместимом языке. Межъязыковая совместимость поддерживается и для наследования в ООП (объектно-ориентированном программировании), например, на основе C#–класса можно объявлять классы в программах на Visual Basic .NET и наоборот.

Основные отличия и преимущества платформы .NET

1. Единая программная модель (представляется единая общая объектно-ориентированная программная модель подключения функций ОС).

2. Упрощенная модель программирования (разработчику не нужно разбираться с реестром, глобально-уникальными идентификаторами (QUID), lUnknown, A d d R e f , Release, HRESUIT и т. д. Но в случае если пишется при-

Библиотека БГУИР

ложение .NET Framework, которое взаимодействует с существующим не-.NET кодом, нужно разбираться во всех этих концепциях.

3. Отсутствие проблем с версиями. Архитектура .NET Framework позволяет изолировать прикладные компоненты, так что приложение всегда загружает версии компонент, с которыми оно строилось и тестировалось. Если приложение работает после начальной установки, оно будет работать всегда (в отличие от сложностей с версиями DLL).

4. Упрощенная разработка. Компоненты .NET Framework (их называют просто *типами)* теперь не связаны с реестром. По сути, установка приложений .NET Framework сводится лишь к копированию файлов в нужные каталоги и установку ярлыков в меню Start, на рабочем столе или на панели быстрого запуска задач. Удаление же приложений сводится к удалению файлов.

5. Работа на нескольких платформах. При компиляции кода для .NET Framework компилятор генерирует код на общем промежуточном языке (common intermediate language, CIL), а не традиционный код, состоящий из процессорных команд конкретного процессора. Это значит, что можете развертывать приложение для .NET Framework на любой машине, где работает версия CLR и FCL.

6. Интеграция языков программирования. COM позволяет разным языкам *взаимодействовать.* .NET Framework позволяет разным языкам *интегрироваться,* т. е. одному языку использовать типы, созданные на других языках. На основе предоставляемой CLR общей системы типов (Common Type System, CTS).

7. Упрощенное повторное использование кода .NET Framework позволяет создавать собственные классы, предоставляющие сервис сторонним приложениям для многократного использования кода.

8. Автоматическое управление памятью (сбор мусора). CLR автоматически отслеживает использование ресурсов, гарантируя, что не произойдет их утечки. По сути, она исключает возможность явного «освобождения» памяти, используя сборщик мусора.

9. Проверка безопасности типов. CLR может проверять безопасность использования типов в коде, что гарантирует корректное обращение к существующим типам. Если входной параметр метода объявлен как 4-байтное значение, CLR обнаружит и предотвратит применение 8-байтного значения для этого параметра. Безопасность типов также означает, что управление может передаваться только в определенные точки (точки входа методов). Невозможно указать произвольный адрес и заставить программу исполняться начиная с этого адреса. Совокупность всех этих защитных мер избавляет от многих распространенных программных ошибок.

10. Развитая поддержка отладки. Поскольку CLR используется для многих языков, можно написать отдельный фрагмент программы на языке, наиболее подходящем для конкретной задачи, – CLR полностью поддерживает отладку многоязыковых приложений.

Библиотека БГУИР

11. Единый принцип обработки сбоев. В CLR обо всех сбоях сообщается через исключения, которые позволяют отделить код, необходимый для восстановления после сбоя, от основного алгоритма. Такое разделение облегчает написание, чтение и сопровождение программ. Кроме того, исключения работают в многомодульных и многоязыковых приложениях. CLR также предоставляет встроенные средства анализа стека, заметно упрощающие поиск фрагментов, вызывающих сбои.

12. Безопасность. Используется «кодоцентрический» способ контроля за поведением приложений. Такой подход реализован в модели безопасности доступа к коду.

13. Взаимодействие с существующим кодом. В .NET Framework реализована полная поддержка доступа к COM-компонентам и Win32-функциям в существующих DLL. Конечные пользователи не смогут непосредственно оценить CLR и ее возможности, но они обязательно заметят качество и возможности приложений, построенных для CLR. CLR позволяет разрабатывать и развертывать приложения быстрее и с меньшими накладными расходами, чем это было в прошлом.

3.2. Практическая часть

В данной лабораторной работе мы познакомимся с основными базовыми принципами разработки приложений на платформе .NET. В данной работе будут рассмотрены базовые операции языка CSharp и работа с файлами на данной платформе.

При разработке приложений на языке CSharp имеется два пути разработки соответствующих приложений:

1. Использование для написания исходных кодов программ любого текстового редактора и последующая компиляция исходных кодов программ с помощью компилятора ***cvs.exe***, входящего в состав Microsoft .NET Framework SDK v2.0 (при этом способе написания приложений необходима лишь установка Microsoft .NET Framework SDK v2.0 или более новой версии).
2. Использование для написания исходных кодов программ среды разработки Microsoft Visual Studio 2003/2005/2008/2010. Данная среда разработки обладает мощными средствами разработки приложений, исчерпывающей помощью по всем компонентам, входящим в ее состав.

Последний способ более предпочтителен благодаря более высокой степени удобства использования, поэтому разработку наших работ будем производить с помощью Microsoft Visual Studio.

Приступим к созданию нашего первого приложения. По своему типу оно будет консольным, будет предусматривать работу с данными, которые будут храниться в файлах.

После открытия Visual Studio перед нами отобразится главное окно программы:

Библиотека БГУИР

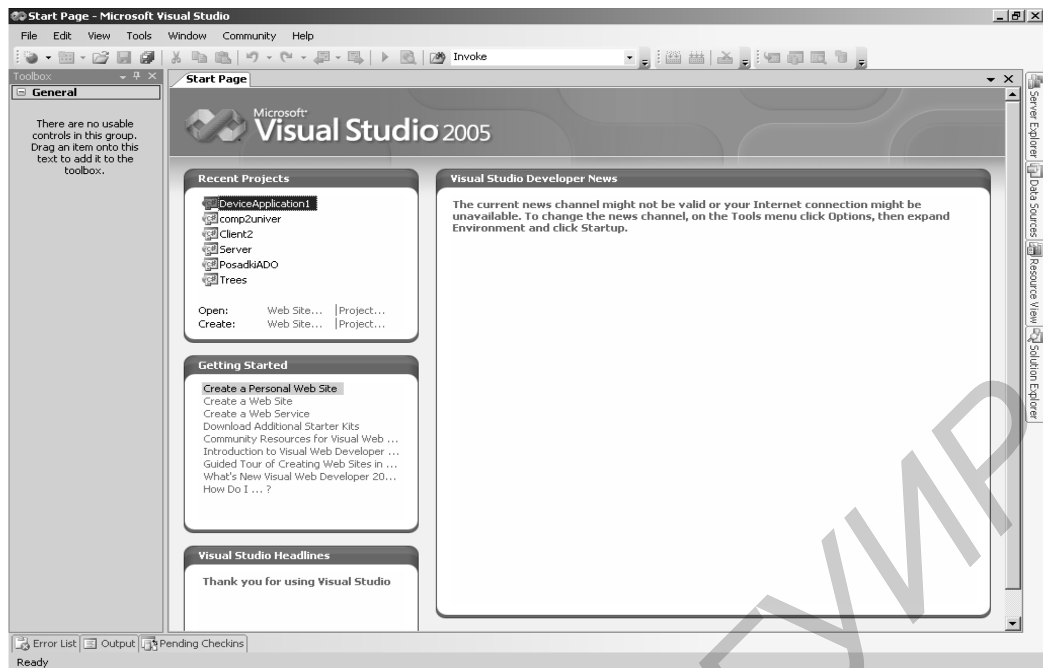

Рис. 3.2. Главное окно MS Visual Studio 2005

В данном окне мы видим список наших проектов. Для создания нового проекта необходимо выбрать команду меню **File->New Project…** после чего перед нами отобразится окно создания новых проектов (рис. 3.3).

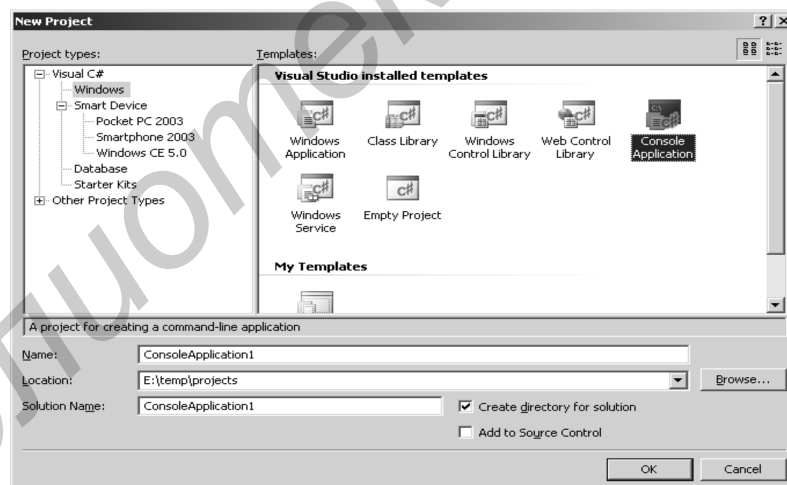

Рис. 3.3. Окно создания нового проекта

В данном окне мы выбираем язык программирования (в нашем случае Visual C#), тип создаваемого приложения – **Console Application**, указываем название нашего проекта и его местоположение.

После того как все параметры заданы, завершаем создание нашего проекта нажатием на клавишу **OK**.

После этого перед пользователем отобразится главное окно разработки приложений (рис. 3.4).

Библиотека БГУИР

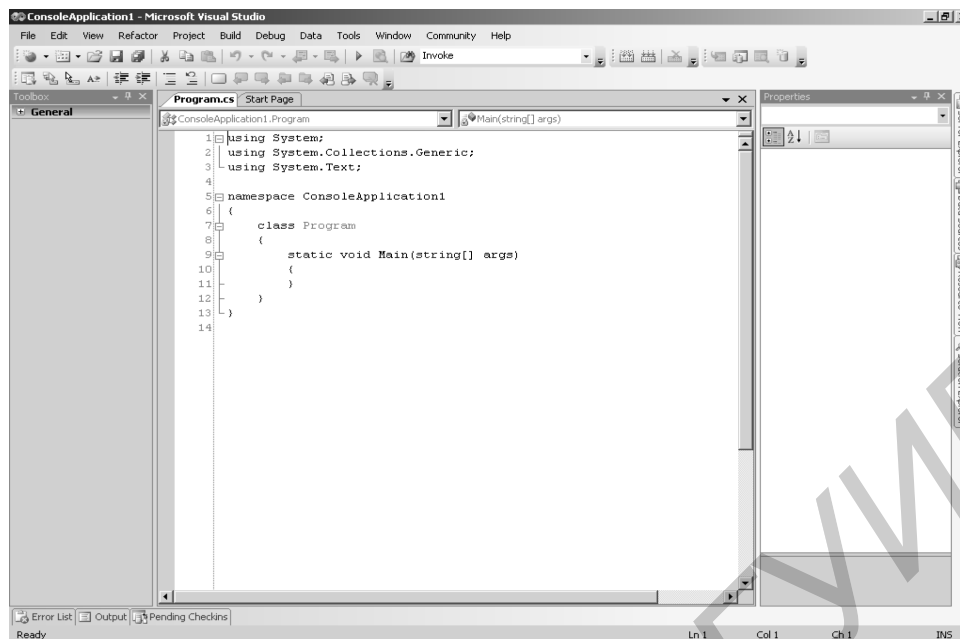

Рис. 3.4. Окно редактора кода

Следует заметить, что после создания проекта система сама формирует начальный шаблон программы и пользователю остается наполнять его необходимым содержимым.

Как видно на рисунке, система сама добавила базовые необходимые библиотеки с помощью директивы **using**. Однако, так как мы планируем вести работу с файлами, нам также необходима дополнительная библиотека **System.IO**, поэтому добавим ее в список подключаемых библиотек.

После этого объявим необходимые нам переменные после объявления класса:

```
    ….
class Laba
{
    private String region;
    private String tkind;
    private long quant;
     ….
```

Затем приступим к реализации бизнес-логики нашего приложения. Реализация описывается в функции ***Main()***нашего приложения.

Рассмотрим подробнее формирование начального пользовательского меню с помощью данного фрагмента кода:

```
    ….
Console.WriteLine("Vyberite punkt:\n 1 - Vvesti novuju zapis\n 2 - Posmotret' posadki derevjev v rajonax\n          3-Posmotret konkretnij rajon\n"+" 4-Udalit zapis\n 5 - Redaktirovat' zapis\n  6 - Sortirovat' po rajonu\n 7  - Exit");
String choice=Console.ReadLine();
```

Библиотека БГУИР

```
switch(choice)
{
....
```

В данном фрагменте кода с помощью функции ***Console.WriteLine()*** пользователю отображается меню выбора функций.

После того как пользователь делает свой выбор, мы с помощью функции ***Console.ReadLine()*** присваиваем переменной **choice** выбор пользователя (т. е. номер выбранного пункта меню). После этого с помощью конструкции **switch(choice)** мы обрабатываем пользовательские запросы.

Рассмотрим запрос пользователя на добавление новой записи в файл:

```
        case "1":
                    {
Console.WriteLine("Vvedite rajon, porodu dereva i kolichestvo:");
//чтение вводимых данных
String region=Console.ReadLine();
String tkind=Console.ReadLine();
long quant=System.Int64.Parse(Console.ReadLine());
                              Laba obj=new Laba(region,tkind,quant);
//открытие файла с данными на запись
FileStream fstr = new FileStream("e:\\file.txt", FileMode.OpenOrCreate,
FileAccess.Write);
fstr.Seek(0,SeekOrigin.End);
StreamWriter sw=new StreamWriter(fstr);
//запись в файл
sw.WriteLine(obj.region + " " +obj.tkind+ " "+obj.quant.ToString()+" ");
sw.Close();
fstr.Close();
break;
}
```

Рассмотрим подробнее представленный выше фрагмент кода: здесь пользователю программы предлагается ввести новые данные, которые будут помещены в файл. С помощью функциий ***Console.ReadLine()*** мы считываем пользовательский ввод и создаем экземпляр нашего класса с полученными данными.

После этого с помощью строки

```
FileStream fstr = new FileStream("e:\\file.txt", FileMode.OpenOrCreate,
FileAccess.Write)
```

мы создаем объект класса **FileStream**, в качетсве параметров передавая ему путь к файлу, в котором хранятся данные, выбираем с помощью метода **FileMode** статус файла: в нашем случае это **OpenOrCreate** (что означает, что данный файл будет открыт, или, в случае его отсутствия, будет создан соотвествующий файл), указывем с помощью метода **FileAccess** тип доступа к файлу: **Write**(так как мы планируем запись данных в файл). После того как

Библиотека БГУИР

объект **FileStream** создан мы с помощью функции ***Seek(0,SeekOrigin.End)*** указываем, что запись данных будет производиться в конец файла.

Далее необходимо создать объект класса **StreamWriter** и в качестве параметра передать созданный нами ранее объект класса **FileStream**. После этого мы можем с помощью метода ***WriteLine()*** класса **StreamWriter** внести необходимые нам данные в файл.

Все остальные операции реализуются аналогичным образом, поэтому рассматривать их детально мы не будем.

Индивидуальные задания

1. Разработать приложение, реализующее учет продажи телевизоров.
2. Разработать приложение, реализующее учет закупки компьютеров университетом.
3. Разработать приложение, реализующее учет посещаемости студентами лекций.
4. Разработать приложение, реализующее учет покупок мороженого в магазине.
5. Разработать приложение, реализующее учет количества сотрудников в подразделениях предприятия.
6. Разработать приложение, реализующее учет зверей в зоопарке.
7. Разработать приложение, реализующее учет машин в автосалоне.
8. Разработать приложение, реализующее учет продажи конфет кондитерской фабрикой.
9. Разработать приложение, реализующее учет продажи канцелярских товаров в магазине.
10. Разработать приложение, реализующее учет отгрузки овощей на торговом складе.
11. Разработать приложение, реализующее учет поставок топлива на автозаправки.
12. Разработать приложение, реализующее учет поставок учебников в школы.
13. Разработать приложение, реализующее учет нагрузки по предметам.
14. Разработать приложение, реализующее учет успеваемости студентов.
15. Разработать приложение, реализующее учет посещаемости студентов.

Общее требование ко всем лабораторным работам: приложение должно предусматривать хранение исходных данных в файле, должно предусматривать возможность добавления, изменения, удаления просмотра, поиска и сортировки информации.

Библиотека БГУИР

Вопросы для самопроверки

1. Чем обусловлена упрощенная модель программирования в среде .NET?
2. За счет чего обеспечивается отсутствие проблем с версиями программного обеспечения, используемого при разработке приложений на платформе .NET?
3. Каким образом реализуется взаимодействие с существующим кодом?
4. Чем обусловлена мультиплатформенность приложений, создаваемых с помощью платформы .NET?
5. Что такое общеязыковая спецификация?
6. Какие приложения можно создавать с помощью платформы .NET и какие объектно-ориентированные языки при этом используются?
7. Каковы основные компоненты платформы .NET?
8. Какие функции несет в себе .NET Framework?
9. Что такое CLR?
10. Что такое общеязыковая спецификация?

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №4
РАЗРАБОТКА GUI ПРИЛОЖЕНИЯ В АРХИТЕКТУРЕ КЛИЕНТ–СЕРВЕР НА ЯЗЫКЕ C#

Цель: приобрести навыки разработки GUI приложений на языке C# в архитектуре клиент–сервер для платформы .NET.

4.1. Теоретическая часть

Сокеты появились тогда, когда начали создавать первые системы распределенных вычислений. Их любят многие за гибкость и простоту в использовании. На их основе можно очень быстро и просто сделать первый прототип любого распределенного приложения. Присутствуют они и в .NET (**System.Net.Sockets**). В следующем листинге представлен простейший пример взаимодействия «клиент–сервер»:

```
private static void OnBeginAccept(IAsyncResult ar)
{
 Socket listener = (Socket)ar.AsyncState
 using (Socket client = listener.EndAccept(ar))
 {
  byte[] buffer = new byte[_bufferSize];
  // Получаем данные от клиента.
  client.Receive(buffer);
  buffer = Encoding.UTF8.GetBytes(string.Format("Hello, {0}",
   Encoding.UTF8.GetString(buffer)));
  // Отправляем клиенту ответ.
  client.Send(buffer);
 }
}
static void Main(string[] args)
{
 EndPoint endPoint = new IPEndPoint(IPAddress.Loopback, 8320);
 using (Socket listener = new Socket(endPoint.AddressFamily,
  SocketType.Stream, ProtocolType.Tcp))
 using (Socket client = new Socket(endPoint.AddressFamily,
  SocketType.Stream, ProtocolType.Tcp))
 {
  // Сокет будет слушать порт 8320 на локальном адресе
  listener.Bind(endPoint);
  // Максимальный размер очереди подключений.
  // Нам для теста достаточно одного.
  listener.Listen(1);
  // Запускаем поток, который ждет подключения клиента.
  listener.BeginAccept(new AsyncCallback(OnBeginAccept), listener);
  // Подсоединяемся клиентом.
```

Библиотека БГУИР

```
   client.Connect(endPoint);

   Console.Write("Введите сообщение: ");
   string request = Console.ReadLine();
   // Отсылаем данные на сервер.
   int count = client.Send(Encoding.UTF8.GetBytes(request));
   byte[] buffer = new byte[_bufferSize];
   // Получаем данные с сервера.
   client.Receive(buffer);
   string response = Encoding.UTF8.GetString(buffer);
   int index = response.IndexOf('\0');
   Console.WriteLine(string.Format("Ответ сервера: {0}",
    response.Remove(index)));
  }
    }
```

В .NET есть и более высокоуровневые надстройки над сокетами. Классы **TcpClient** и **TcpListener** предоставляют функциональность, осуществляющую коммуникации не на уровне сокетов, а на уровне потока ввода/вывода. Для этого используется класс **NetworkStream.** Данный класс является наследником **System.IO.Stream,** но переопределенные им методы чтения и записи не записывают данные ни в какие внутренние хранилища (файлы, массивы байт), а передают прямо в сокет на сервер.

В .NET есть надстройка над сокетом, общающимся через UDP. Это класс **UdpClient**. Но пусть слово «Client» не вводит вас в заблуждение. UdpListener в .NET нет. **UdpClient**, по сути, является одновременно и клиентом, и сервером. Соответственно, его методы ***Receive()*** и ***Send()*** нужны именно для двунаправленного общения.

Теперь рассмотрим подробнее работу по созданию GUI приложений для платформы .NET. Создание приложений для Windows во многом упрощено благодаря использованию среды разработки Microsoft Visual Studio. Создание графических приложений на языке C# во многом аналогично созданию таковых на языке Visual C++;

Рассмотрим работу с формами Windows и сокетами на примере.

4.2. Практическая часть

В данной лабораторной работе мы создадим два графических приложения для Windows(клиентское приложение и серверное приложение), которые будут взаимодействовать между собой через сокеты.

4.2.1. Серверная часть программы

Вначале рассмотрим пример реализации серверного приложения:

Запустим Microsoft Visual Studio и с помощью команды **File->New Project…** перейдем в диалоговое окно создания нового проекта:

Библиотека БГУИР

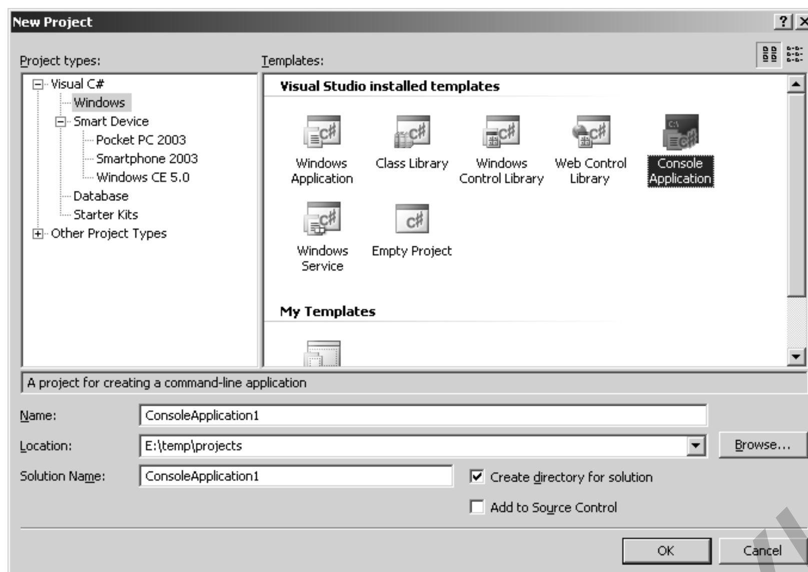

Рис. 4.1. Окно создания нового проекта

В данном окне нам необходимо выбрать тип создаваемого приложения: в нашем случае это **Windows Application**.

После завершения формирования нашего проекта перед нами отобразится редактор форм. По середине экрана мы видим главную форму нашего приложения.

На экране слева расположены элементы управления. Все элементы управления сгруппированы по своим функциям: например, базовые элементы управления, такие как Кнопка, Список, Поле Ввода Текста, расположены в группе **Common Controls** (общие элементы управления), элементы для работы с данными – в группе **Data**(данные) и т.д. В нашей лабораторной работе мы будем использовать только общие элементы управления.

Добавим на нашу форму элементы управления **RichTextBox** и **Button**. В списке будет отображаться служебная информация, а Кнопка будет использоваться для инициализации сервера. Окно с нашей формой в редакторе форм примет следующий вид (рис. 4.2).

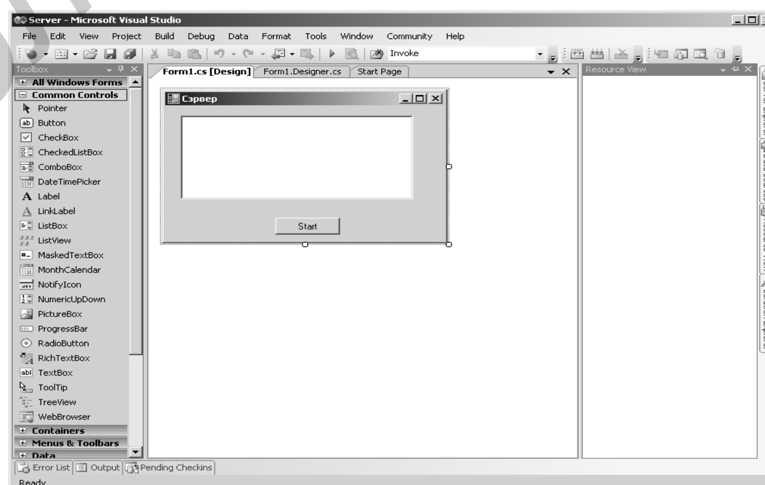

Рис. 4.2. Готовая форма в редакторе форм

Библиотека БГУИР

После того как элементы управления добавлены на форму, присвоим им соответствующие названия для дальнейшей наглядности и удобства. Делается это следующим образом: на форме выбирается необходимый элемент управления, затем правым щелчком мыши вызывается контекстное меню, в котором выбирается пункт **Properties** (Свойства). В окне свойств присвоим параметрам **Name (**имя**)** значения *button1* и *rtb1* для кнопки и поля для текста соответственно.

Далее приступим к непосредственному наполнению программы кодом. Начнем с обработки события нажатия кнопки. Наша кнопка будет предназначена для инициализации серверного приложения, поэтому в обработчике ее нажатия будет содержаться весь основной код приложения. Перейдем в окно редактирования кода обработчика событий для нашей кнопки путем выполнения двойного щелчка по ней. Однако перед началом написания кода нашего обработчика подключим необходимые нам библиотеки:

```
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.Threading;
```

Первые две библиотеки содержат необходимые классы для организации работы по сети, библиотека **System.IO** предназначена для работы с потоками ввода/вывода, а библиотека **System.Threading** – для работы с потоками.

Далее в конструкторе класса определим необходимые нам переменные:

```
//Путь к файлу, в котором будет храниться информация
String fileName = "e:\\file1.txt";
int fileCount = 0;
//Создание объекта класса TcpListener
TcpListener listener = null;
//Создание объекта класса Socket
Socket socket = null;
//Создание объекта класса NetworkStream
NetworkStream ns = null;
//Создание объекта класса кодировки ASCIIEncoding
ASCIIEncoding ae = null;
```

Объекты класса **TcpListener** предназначены для прослушивания и принятия запросов от клиентских приложений. Для начала прослушивания используется метод ***Start()***. Данный метод помещает входящие запросы на соедниение от клиентов в очередь. После этого с помощью методов ***AcceptSocket() AcceptTcpClient()*** объекты класса принимают входящие запросы на соединение.

Объект класса **Socket** является стандартным объектом типа Socket.

Объект класса **NetworkStream** предоставляет методы для отправки и получения данных при помощи потоковых сокетов. Для использования

Библиотека БГУИР

объектов данного класса должен быть создан подсоединенный сокет. При закрытии объекта **NetworkStream** сокетное соединение не закрывается. Для отправки и получения данных с помощью объектов класса **NetworkStream** используются методы ***Write() Read()*** данного класса.

Перейдем к написанию обработчика нажатия кнопки запуска сервера. Рассмотрим следующий фрагмент кода:

```
// Создаем новый TCP_Listener, который принимает запросы от любых IP-адресов и слушает по порту 5555
listener = new TcpListener(IPAddress.Any, 5555);
// Активация listen'ера
listener.Start();
socket = listener.AcceptSocket();
```

В данном фрагменте кода создается новый объект класса **TcpListener**, который принимает запросы от клиентов по любым IP-адресам, и прослушивающий порт 5555 на предмет запросов на соединение. Как говорилось выше, с помощью метода ***Start()*** происходит активация данного объекта. После того как прослушивание началось, с помощью метода ***AcceptSocket()*** класса **Socket** мы ассоциируем созданный нами объект класса Socket с запросами на соединение от клиентских приложений.

Далее, если сокет соединен, то нам необходимо создать новый сетевой поток для обмена информацией и новый поток для обработки нескольких клиентов. Выглядит это следующим образом:

```
if (socket.Connected)
 {
ns = new NetworkStream(socket);
ae = new ASCIIEncoding();
//Создаем новый экземпляр класса ThreadClass
ThreadClass threadClass = new ThreadClass();
//Создаем новый поток
Thread thread = threadClass.Start(ns, fileName, fileCount, this);
}
```

Далее определим наш класс потока **ThreadClass**: данный класс будет содержать функцию запуска потока ***public Thread Start()***, которая, в свою очередь, будет содержать специальную определенную нами функцию ***void ThreadOperations()***, в которой будет производиться обработка входящих клиентских запросов.

Ключевым моментом при определении нашего класса потока является создание экземпляра нашей формы в данном классе. Это необходимо для доступа к элементам формы из определяемого нами класса.

```
Form1 form = null;
```

Рассмотрим создаваемую нами функцию запуска потока:

```
public Thread Start(NetworkStream ns, String fileName, int fileCount, Form1 form)
{
this.ns = ns;
```


Библиотека БГУИР

```
ae = new ASCIIEncoding();
this.fileName = fileName;
this.fileCount = fileCount;
this.form = form;
//Создание нового экземпляра класса Thread
Thread thread = new Thread(new ThreadStart(ThreadOperations));
//Запуск потока
thread.Start();
return thread;
}
```

При описании функции запуска потока в качестве параметров в нее передаются объект класса **NetworkStream**, имя файла, содержащего информацию, счетчик файлов и экземпляр нашей формы.

Далее при создании нового экземпляра класса **Thread** мы указываем, что при запуске нового потока в нем должна выполняться функция ***ThreadOperations()***, которая будет определена нами ниже.

В функции ***ThreadOperations()*** определены все операции, которые будут работать в потоке, который будет создаваться для каждого из клиентских приложений. Данная функция содержит ряд обработчиков клиентских запросов. Данные обработчики функционально аналогичны, поэтому остановим наше внимание на некоторых из них. Работа нашей функции начинается с чтения запроса из потока:

```
//Создаем новую переменную типа byte[]
byte[] received = new byte[256];
//С помощью сетевого потока считываем в переменную received данные от
клиента
ns.Read(received, 0, received.Length);
String s1 = ae.GetString(received);
int i = s1.IndexOf("|",0);
String cmd = s1.Substring(0, i);
```

В данном фрагменте кода мы считываем получаемую от клиента информацию с помощью объекта класса **NetworkStream**, выделяем из полученных данных команду, которая отделена от остальных данных с помощью символа "|". После того как команда выделена, в программе идет обработка каждой полученной команды. Например, если от клиента поступает запрос на просмотр информации, то функция обработки данного запроса выглядит следующим образом:

```
if (cmd.CompareTo("view") == 0)
 {
// Создаем переменную типа byte[] для отправки ответа клиенту
byte[] sent = new byte[256];
```


Библиотека БГУИР

```
//Создаем объект класса FileStream для последующего чтения информации из
файла
FileStream fstr = new FileStream(fileName, FileMode.Open, FileAccess.Read);
StreamReader sr = new StreamReader(fstr);
//Запись в переменную sent содержания прочитанного файла
sent = ae.GetBytes(sr.ReadToEnd());
sr.Close();
fstr.Close();
//Отправка информации клиенту
ns.Write(sent, 0, sent.Length);
 }
```

В данном фрагменте кода мы создаем переменную для хранения прочитанной информации в байтовом виде, записываем в нее содержание прочитанного нами файла и отправляем данную информацию клиентскому приложению с помощью метода ***Write()*** класса **NetworkStream**.

Аналогично происходит обработка всех остальных клиентских запросов.

4.2.2. Клиентская часть программы

Приступим к созданию клиентского приложения. Создадим новый проект типа **Windows Application** в Microsoft Visual Studio 2005. Наше клиентское приложение будет использовать основные типы для элементов управления для формирования запросов, отображения получаемых данных и служебной информации.

Добавим на нашу форму следующие элементы управления: элементы типа **TextBox** для ввода параметров запросов, элементы типа **RadioButton** для выбора операций, элемент типа **ListBox** для вывода служебной информации и элемент типа **CheckBox**, который будет отвечать за отображение/скрытие списка со служебной информации. После того как все элементы добавлены, наша форма должна иметь следующий вид:

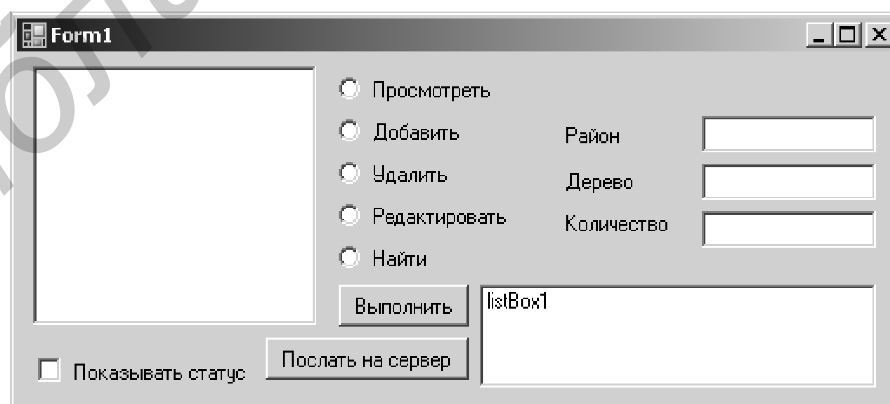

Рис. 4.3. Сформированная форма клиентского приложения

Библиотека БГУИР

Далее для удобства понимания программы всем элементам управления можно присвоить свои специфические имена.

После окончания визуального формирования формы можно приступать к написанию исходного кода клиентской программы. Создадим объекты классов **TcpClient** и **ASIIEncoding**:

```
TcpClient tcp_client = new TcpClient("localhost", 5555);
ASCIIEncoding ae = new ASCIIEncoding();
```

Первый класс предоставляет простые методы для соединения, отправки и получения инфомации по сети в синхронном режиме. При создании объекта данного класса мы указываем, что соединение будет происходить с локальным сервером по порту 5555.

Второй класс предназначен для для создания объектов кодировки, в которой будут отправляться и получаться данные.

После этого укажем начальное состояние определенных компонетов при загрузке нашей формы:

```
public Form1()
{
 InitializeComponent();
 region.Enabled = false;
 kind.Enabled = false;
 quant.Enabled = false;
 listBox1.Visible = false;
 }
```

Данные строки кода указывают, что определенные компоненты в момент инцализации формы будут недоступными.

Далее приступим к обработке событий нажатия кнопки выбора операций и соответствующих RadioButtons. Принцип работы нашей программы будет следующим: сначала выбирается необходимый **RadioButton**, соответсвующий необходимой операции, затем по нажатии кнопки действия вызывается обработчик соответствующего события. Рассмотрим данный принцип на примере команды просмотра информации, получаемой с сервера:

```
private void operation_Click(object sender, EventArgs e)
    {
//Если выбран RadioButton просмотра информции то…
if (radio_view.Checked == true)
{
//Создаем объект класса NetworkStream и ассоциируем его объектом класса
TcpClient
NetworkStream ns = tcp_client.GetStream();
String command = "view";
String res = command + "|";
//Создаем переменные типа byte[] для отправки запроса и получения результата
```


Библиотека БГУИР

```
byte[] sent = ae.GetBytes(res);
byte[] recieved = new byte[256];
//Отправляем запрос на сервер
ns.Write(sent, 0, sent.Length);

//Получаем результат выполнения запроса с сервера
ns.Read(recieved, 0, recieved.Length);
//Отображаем полученный результат в клиентском RichTextBox
richTextBox1.Text=ae.GetString(recieved);
String status = "=>Command sent:view data";
//Отображаем служебную информацию в клиентском ListBox
listBox1.Items.Add(status);
}
```

Прокомментируем данный фрагмент кода. В данном фрагменте происходит обработка сразу двух событий: нажатия кнопки действия и выбора соответствующего **RadioButton**. Первоначально при обработке данных событий создается объект класса **NetworkStream**, который с помощью метода ***GetStream()*** ассоциируется с созданным нами ранее объектом класса **TcpClient**. Метод ***GetStream()*** возвращает объект класса **NetworkStream**, который затем используется для отправки и получения данных. После создания данного объекта появляется возможность использовать методы ***Write()*** и ***Read()*** для отправки и получения данных соответственно. После того как отправлен запрос и получен ответ, данные, содержащиеся в ответе, отображаются в форме путем их загрузки в элемент управления **RichTextBox**. Также в элементе управления **ListBox** отображается служебная информация об отправке запроса на сервер.

Остальные обработчики реализованы аналогичным образом, поэтому не будем останавливаться на их подробном описании.

Индивидуальные задания

В соответствии с заданием, полученным в лабораторной работе №3, разработать графическое приложение на языке C# в архитектуре клиент–сервер. В разрабатываемом приложении предусмотреть многопоточность, хранение информации в файле на серверной стороне, а со стороны клиентской части приложения предусмотреть возможность просмотра, добавления, удаления, редактирования и поиска информации, хранящейся на сервере.

Вопросы для самопроверки

1. Что такое форма в .NET.? Особенности ее создания.
2. Для чего предназначен и как используется элемент управления RadioBox?
3. Каким образом происходит заполнение и чтение данных из элементов управления RichTextBox и TextBox?
4. Для чего предназначен класс TCP_Client?

Библиотека БГУИР

5. Что указывается в параметрах создаваемого экземпляра объекта TCP_Client?
6. Какова последовательность действий при создании GUI приложения?
7. Для чего предназначен и как используется контрол ListBox?
8. Каким образом организуется работа контролов CheckBox и RadioBox?
9. Для чего предназначен класс TCP_Listener?
10. Что указывается в параметрах создаваемого экземпляра объекта TCP_Listener?

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №5
РАЗРАБОТКА ПРИЛОЖЕНИЯ, ИСПОЛЬЗУЮЩЕГО ТЕХНОЛОГИЮ ADO.NET

Цель: приобрести навыки разработки приложений с использованием технологии ADO.NET

5.1. Теоретическая часть

5.1.1. Преимущества и нововведения в ADO.NET

Использование разъединенной модели доступа к данным

В клиент-серверных приложениях традиционно используется технология доступа к источнику данных, при которой соединение с базой поддерживается постоянно. Однако после широкого распространения приложений, ориентированных на Интернет, выявились некоторые недостатки такого подхода. Попробуем выявить некоторые из них.

Соединения с базой данных требуют выделения системных ресурсов, что может быть критично при большой нагрузке сервера. Хотя постоянное соединение позволяет несколько ускорить работу приложения, общий убыток от растраты системных ресурсов сводит преимущество на нет.

Специфика web-приложений не позволяет серверу в каждый момент времени знать, что необходимо пользователю. То есть до следующего запроса сервер не имеет представления, нужно ли еще поддерживать соединение.

Опыт разработчиков показал, что приложения с постоянным соединением с источником данных чрезвычайно трудно поддаются масштабированию.
Хотя существуют и другие недостатки, приведенные наиболее существенны. Все эти проблемы порождаются постоянным соединением с базой данных и решаются в ADO.NET, где используется другая модель доступа. Теперь соединение устанавливается лишь на то короткое время, когда необходимо проводить операции над базой данных.

В основе ADO.NET лежит новая объектная модель, в основе которой, в свою очередь, два разных подхода к работе с данными: присоединенный и отсоединенный.

Присоединенные объекты (такие как Connection, Transaction, DataReader, Command) предназначены для управления соединением, транзакциями, выборки данных и передачи изменений в БД. **Отсоединенные объекты** удобны тем, что они позволяют работать с данными автономно (и манипулировать данными в произвольном порядке).

Хранение данных в объектах DataSet

При работе с базой данных нам чаще всего приходится работать не с одной, а несколькими записями. Более того, данные эти могут собираться из различных таблиц. В разъединенной модели доступа к базе данных не имеет смысла соединяться с источником данных при каждом обращении. Исходя из этого,

Библиотека БГУИР

представляется логичным хранить несколько строк и обращаться к ним при необходимости. Для этих целей и используется **DataSet**.

DataSet представляет собой, по сути, упрощенную реляционную базу данных и может выполнять наиболее типичные для таких баз данных операции. Теперь, в отличие от **Recordset,** мы можем хранить в одном **DataSet** сразу несколько таблиц, связи между ними, выполнять операции выборки, удаления и обновления данных. Безусловно, разъединенная модель не позволяет постоянно отслеживать изменения в базе данных, производимые другими пользователями. Это может привести к ошибкам в таких приложениях, где информация должна обновляться каждый момент, – заказ билетов или продажа ценных бумаг. Однако в любую секунду может быть получена свежая информация из базы данных через вызов метода ***FillDataSet***. Таким образом, **DataSet** остается чрезвычайно удобным для самого широкого класса приложений, когда необходимо получить данные из базы и как-либо обработать их.

Глубокая интеграция с XML

Все более широко распространяющийся язык XML играет важнейшую роль в технологии ADO.NET и приносит еще несколько преимуществ по сравнению с традиционным подходом.

Заметим для начала, что практически любой XML-файл может быть использован как источник данных и на его основе может быть создан **DataSet**. Точно так же при передаче данных между компонентами или сохранении их в файл используется XML.

Программист, работающий с ADO.NET, не обязательно должен иметь опыт работы с XML или познания в этом языке. Все операции остаются прозрачными для разработчика.

Так как XML имеет текстовое представление, это позволяет передавать его по протоколам типа HTTP через брандмауэры. Дело в том, что системы защиты обычно настроены на фильтрацию двоичной информации, текстовая же информация легко пропускается, что облегчает создание распределенных приложений. XML представляет собой промышленный стандарт, поддерживаемый практически любой современной платформой, что позволяет передавать данные любому компоненту, умеющему работать с XML и выполняющемуся под любой операционной системой.

При передаче больших объемов информации при использовании технологии COM возникает проблема приведения типов данных, так как COM поддерживает лишь ограниченный их набор. Действительно, COM-маршаллинг может требовать длительной обработки, что негативно сказывается на производительности приложения. XML же поддерживает неограниченное число типов и не требует их конверсии, что позволит ускорить процесс передачи данных.

5.1.2. Практическое применение ADO.NET

ADO.NET поддерживает два типа источников данных – SQL Managed Provider и ADO Managed Provider. SQL Managed Provider применяется для рабо-

Библиотека БГУИР

ты с Microsoft SQL Server 7.0 и выше, ADO Managed Provider – для всех остальных баз данных.

SQL Managed Provider работает по специальному протоколу, называемому TabularData Stream (TDS), и не использует ни ADO, ни ODBC, ни какую-либо еще технологию. Ориентированный специально на MS SQL Server, протокол позволяет увеличить скорость передачи данных и тем самым повысить общую производительность приложения.

ADO Managed Provider предназначен для работы с произвольной базой данных. Однако за счет универсальности есть проигрыш по сравнению с SQL Server Provider, так что при работе с SQL Server рекомендовано использовать специализированные классы. В данной работе мы коснемся ADO Managed Provider лишь мельком, указав только существующие незначительные различия.

Вне зависимости от того, какой поставщик вы будете использовать, вам придется работать с тремя общими объектами для взаимодействия с БД:

- Command;
- Dataset;
- Connection.

Объект Connection

Данный объект предназначен для установления с источником данных. Следующие примеры кода иллюстрируют инициализацию соединения с БД:

```
using System.Data;
using System.Data.SQL;

public class SQLConnect {
  private String connString = "";
  private SQLConnection dataConn = null;

  public string openConnection(String dbConnectString) {
    connString = dbConnectString;
    try {
      dataConn = new SQLConnection(connString);
      dataConn.Open();
      return "SQL Server Data Connection Opened";
    }
    catch (Exception e) {
      return(e.ToString());
    }
    finally {
      if (dataConn != null) {
        dataConn.Close();
```


Библиотека БГУИР

```
      }
    }
  }
        } // End Class
```

Объект Object

Хотя установка соединения с БД важна, но нам необходимо средство для получения данных. Здесь нам поможет объект **Command.**

Ниже показаны примеры использования объекта **Command**:

```
using System.Data;
using System.Data.SQL;

public class SQLConnect {
  private SQLConnection dataConn = null;
  private SQLDataReader reader = null;
  public string openConnection(HttpResponse Response,String
  dbConnectString,String cmdString) {
    try {
      dataConn = new SQLConnection(dbConnectString);
      SQLCommand sqlCmd = new SQLCommand(cmdString,dataConn);
      dataConn.Open();
      sqlCmd.Execute(out reader);
      Response.Write("<table><tr><td><b>ID</b></td>");
      Response.Write("<td><b>Name</b></td></tr>");
      while (reader.Read()) {
        Response.Write("<tr>");
        Response.Write("<td>" +reader["CategoryID"].ToString());
        Response.Write("</td>");
        Response.Write("<td>");
        Response.Write(reader["CategoryName"].ToString());
        Response.Write("</td>");
        Response.Write("</tr>");
      }
      Response.Write("</table>");
      return "<p>SQL Server Data Connection Opened";
    }
    catch (Exception e) {
      return(e.ToString());
    }
    finally {
      if (reader != null) {
        reader.Close();
      }
      if (dataConn != null) {
```


Библиотека БГУИР

```
          dataConn.Close();
        }
      }
   }
} // End Class
```

Объект **Command** имеет множество конструкторов, принимающих различные параметры. В наших примерах мы передавали строку с SQL-скриптом и объект класса **Connection**.

Пара слов о объекте **DataReader**: в отличие от объектов **Dataset** он использует forwardonly доступ и хранит только одну строку в памяти в каждый момент времени. Так как он читает построчно, то вынужден поддерживать соединение (очень похоже на обычный **Recordset**).

При инициализации объекта устанавливаются различные свойства по умолчанию. Например, поле ***CommandType*** устанавливается в ***CommandType.Text***. Однако легко можно вызывать и сохраненные процедуры.

```
using System.Data;
using System.Data.SQL;

public class SQLConnect {
  private SQLConnection dataConn = null;
  private SQLDataReader reader = null;

  public string openConnection(HttpResponse Response,String
                        dbConnectString,String cmdString) {
  try {
    dataConn = new SQLConnection(dbConnectString);
    SQLCommand sqlCmd = new SQLCommand(cmdString,dataConn);
    sqlCmd.CommandType = CommandType.StoredProcedure;
    SQLParameter param = sqlCmd.Parameters.Add(new
    SQLParameter("@CustomerID",SQLDataType.Char, 5));
    param.Direction = ParameterDirection.Input;
    sqlCmd.Parameters["@CustomerID"].Value = "ALFKI";
    dataConn.Open();
    sqlCmd.Execute(out reader);
    Response.Write("<table><tr><td><b>Product Name</b></td>");
    Response.Write("<td><b>Total</b></td></tr>");
    while (reader.Read()) {
    Response.Write("<tr>");
    Response.Write("<td>"+reader["ProductName"].ToString());
    Response.Write("</td>");
    Response.Write("<td>"+reader["Total"].ToString());
    Response.Write("</td></tr>");
    }
    Response.Write("</table>");
```


Библиотека БГУИР

```
    return "<p>SQL Server Data Connection Opened";
  }
  catch (Exception e) {
    return(e.ToString());
  }
  finally {
    if (reader != null) {
      reader.Close();
    }
    if (dataConn != null) {
      dataConn.Close();
    }
  }
  }
} // End Class
```

Объект DataSet

Основные особенности объекта **DataSet**:

1. Dataset не взаимодействует напрямую с источником данных.
2. Поддерживает отношения между различными таблицами.
3. Вы можете перемещаться по таблицам с помощью этих отношений.
4. Для передачи данных используется XML.
5. Помимо расширения передаваемых типов, это позволяет передаваемому объекту не испытывать воздействия брандмауэров.

Ниже (рис. 5.1) приведена иллюстрация объектной модели Dataset

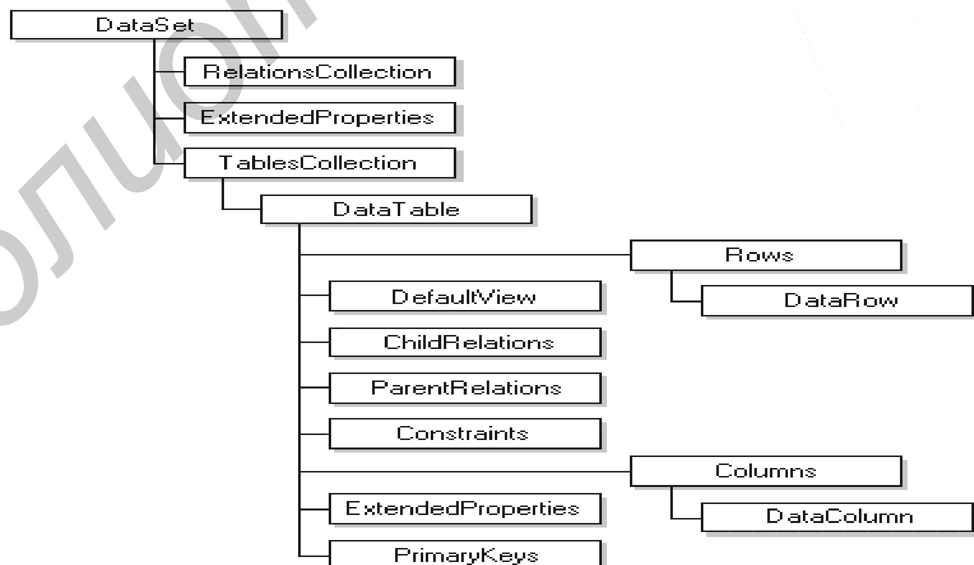

Рис. 5.1. Объектная модель DataSet

Как видно, объект **DataSet** имеет более сложную структуру, нежели объект **Recordset**.

Библиотека БГУИР

Схема наглядно показывает возможности хранения нескольких таблиц и программного добавления связей между ними.

Для заполнения объекта **DataSet** в первый раз используется метод ***FillDataset()*** объекта **DatasetCommand**. Метод получает два параметра – имя объекта **DataSet,** который должен быть заполнен, и имя таблицы, которая будет добавлена в **DataSet**.

Код ниже иллюстрирует заполнение Dataset на примере SQL provider:

```
public class SQLConnect {
    public string openConnection(HttpResponse Response,String
                        dbConnectString,String cmdString) {
  try {
    Dataset dsOrders = new Dataset();
    SQLConnection dataConn = new
 SQLConnection(dbConnectString);
    SQLDatasetCommand dsCmd = new
SQLDatasetCommand(cmdString,dataConn);
    dsCmd.FillDataset(dsOrders,"Orders");
    // Loop through Dataset "Orders" table
    Response.Write("<table><tr><td><b>Order ID</b></td></tr>");
    foreach (DataRow Order in dsOrders.Tables["Orders"].Rows) {
      Response.Write("<tr>");
      Response.Write("<td>" + Order["OrderId"].ToString() + "</td>");
      Response.Write("</tr>");
    }
    Response.Write("</table>");
    return "<p>SQL Server Data Connection Opened";
  }
  catch (Exception e) {
    return(e.ToString());
  }
  }
} //End Class
```

Код во многом похож на использованный при работе с DataReader, хотя соединение и не поддерживается во время перебора записей. Видно, что после заполнения **DataSet** мы просматриваем все строчки таблицы.

5.2. Практическая часть

Приступим к написанию программы, реализующей взаимодействие с базой данных по средствам механизмов ADO.NET.

Создадим новое приложение Windows. Для этого в главном меню среды программирования Visual Studio 2005 выберем элемент меню **File->New Project…** В появившимся окне в поле выберем язык разработки Visual C#, а в качестве типа приложения – **Windows Application**. В поле Name зададим имя

Библиотека БГУИР

нашего приложения, а в поле **Location** – его месторасположение. После проведения вышеописанных операций подтвердим свои действия нажатием на кнопку OK и тем самым завершим создание нового проекта.

Установим соединение с нашей базой данных. В окне **Toolbox** выберем закладку **Data** и в ней элемент управления **OleDbConnection** и мышью перетащим данный элемент в форму приложения (рис. 5.2).

Рис. 5.2. Элементы управления для работы с данными

После этого в нижней части формы приложения появится объект с именем **oleDbConnection1** (рис. 5.3).

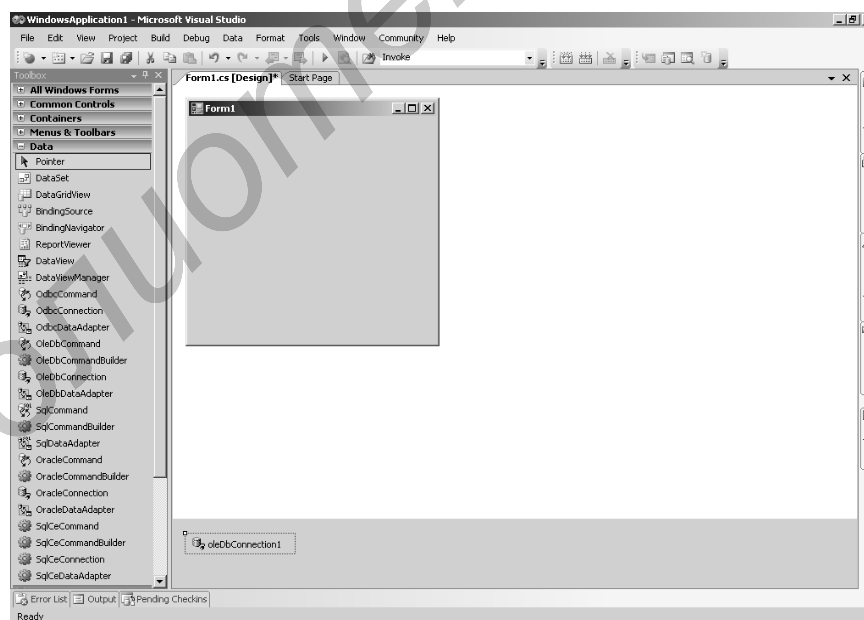

Рис. 5.3. Форма для создания приложения

Этот объект соединения с источником данных OLE DB. Сделаем этот объект текущим в форме, для чего щелкнем по нему мышью. В окне **Properties** найдем свойство **ConnectionString**(строка соединения) и в поле данных

Библиотека БГУИР

нажмем кнопку выпадающего списка. В появившемся списке выберем строку **New Connection** (рис 5.4).

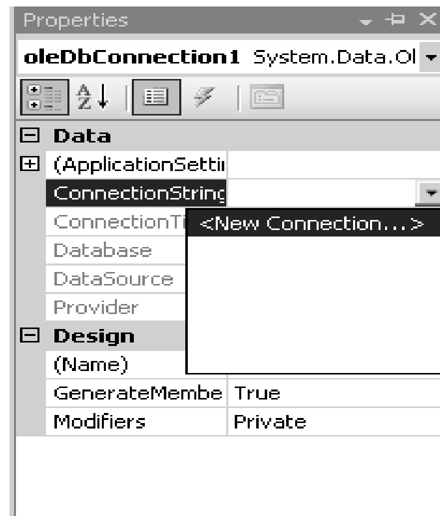

Рис. 5.4. Создание новой строки соединения

В появившимся окне Add Connection укажем месторасположение нашей базы данных на жестком диске, а также логин и пароль к нашей базе данных (при необходимости). Проверку соединения с базой данных можно осуществить с помощью кнопки Test Connection.

Строка соединения с базой данных будет выглядеть так:

Provider=Microsoft.Jet.OLEDB.4.0;DataSource=C:\comp2univer.mdb

Далее создадим адаптер данных – промежуточное звено между набором данных (еще не созданным) и таблицей Kaf нашей базы данных. В окне Toolbox на закладке Data выберем объект **OleDbDataAdapter** и перетащим его в форму. На экране появится окно мастера настройки и создания адаптеров данных (Data Adapter Configuration Wizard). Нажмем кнопку Next. В следующем окне выберем радиокнопку Use SQL Statements и нажмем кнопку Next. В следующем окне (Generate SQL Statements) в поле данных введем следующую команду языка SQL:

```
SELECT kaf_id, kaf_name,comps
FROM Kafedra
```

В следующем окне нажмем кнопку Finish. Наш адаптер данных сконфигурирован.

Выберем созданный нами адаптер данных **oleDbDataAdaper1** и перейдем в окно его свойств. Изменим имя адаптера на **KafAdapter**.

Сгенерируем набор данных (DataSet). Выберем адаптер данных и правым щелчком мыши перейдем в контекстное меню, в котором выберем пункт Generate DataSet… (рис. 5.5).

Библиотека БГУИР

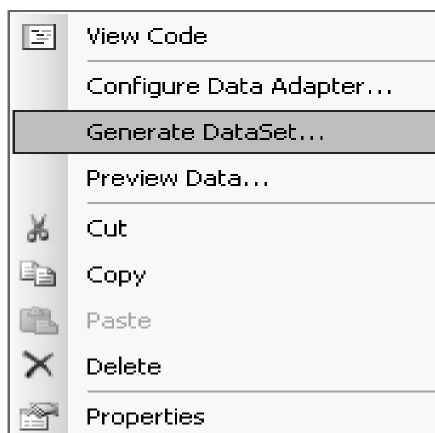

Рис. 5.5. Контекстное меню объекта DataAdapter

В появившемся окне установим значения полей, как показано на рис. 5.6.

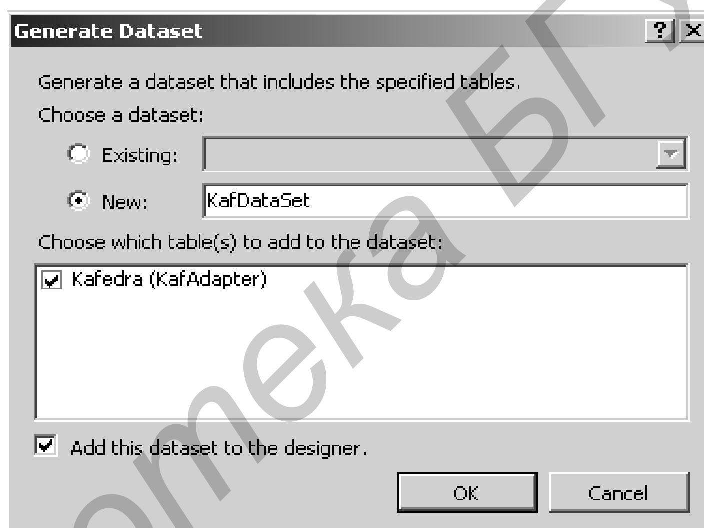

Рис. 5.6. Окно генерации набора данных

После этого в приложение будет добавлен класс набора данных с именем **KafDataSet**.

Зальем в нашу таблицу Kaf набора данных **KafDataSet1** записи из одноименной таблицы в нашей базе данных. Для этого дважды щелкнем мышью по полю формы и в обработчике события загрузки формы зададим выделенный код:

```
private void Form1_Load(object sender, EventArgs e)
{
  KafAdapter.Fill(kafDataSet1);
 }
```

Далее перейдем в окно **Toolbox** на закладку **Data**, выберем элемент управления **DataGridView** и перетащим его на нашу форму. Сразу после

Библиотека БГУИР

перенесения элемента на форму перед нами появится окно выбора набора данных, с которым будет ассоциирован наш **DataGridView** (рис. 5.7).

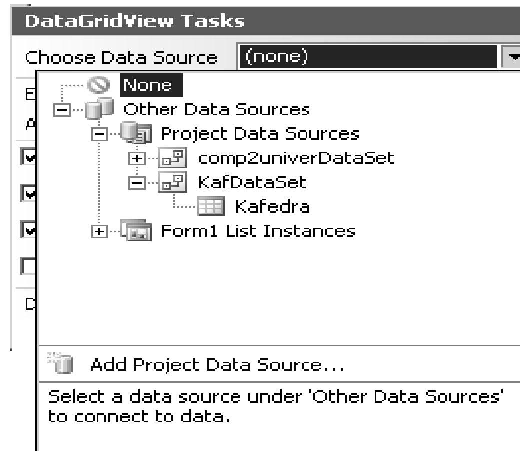

Рис. 5.7. Окно выбора набора данных

В списке «адаптер данных» выберем созданный нами набор данных **KafDataSet**, а в нем таблицу, с которой он ассоциирован – Kafedra. После завершения вышеописанной операции наша форма должна принять следующий вид (рис. 5.8).

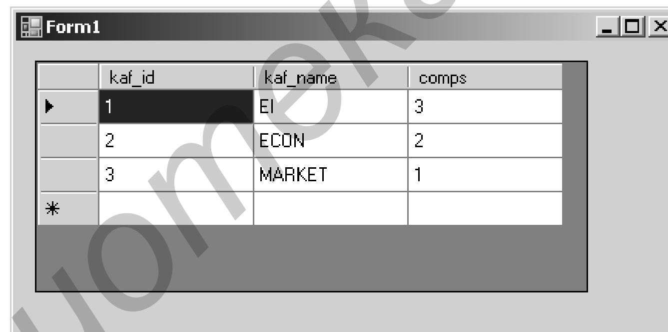

Рис. 5.8. Форма с загруженной в нее информацией

Теперь на нашей форме отображаются данные из таблицы Kafedra нашей базы данных.

Добавим на нашу форму две кнопки для сохранения в базе данных изменений, вносимых через **DataGridView1,** и для отмены этих изменений. В окне **Toolbox** выберем закладку **Common Controls**, а в ней – элементы управления Button и перетащим две кнопки на нашу форму. В окне свойств кнопок присвоим кнопкам новые названия: *btn_save* и *btn_cancel* соответственно. После добавления кнопок форма должна принять следующий вид (рис. 5.9)

Библиотека БГУИР

Рис. 5.9. Итоговый вид формы

Добавим в обработчик события выделенный код:

```
private void btn_save_Click(object sender, EventArgs e)
  {KafAdapter.Update(kafDataSet1, "Kafedra");
}
```

Прокомментируем данный обработчик: здесь с помощью метода ***Update()*** адаптера данных мы обновляем нашу таблицу базы данных данными из созданного нами объекта **DataSet**, передавая названия таблицы и набора данных в данную функцию в качестве параметров.

Добавим в обработчик событий для отмены изменений, вносимых в таблицу базы данных, выделенный код:

```
private void btn_cancel_Click(object sender, EventArgs e)
 {
  kafDataSet1.Kafedra.RejectChanges();
 }
```

В данном фрагменте кода мы с помощью метода ***RejectChanges()*** производим откат изменений, вносимых в таблицу Kafedra набора данных **kafDataSet1**.

В итоге у нас получилось приложение, отображающее данные из выбранной нами таблицы, предусматривающее добавление новых данных в таблицу базы данных, редактирование, удаление и сортировку данных, а также сохранение изменений в базе данных и откат нежелательных изменений.

Индивидуальные задания

1. Разработать приложение учета продажи авиабилетов в аэропорту.
2. Разработать приложение учета продаж компьютеров.
3. Разработать приложение учета посадок деревьев в районах города Минска.
4. Разработать приложение учета продажи лотерейных билетов различных лотерей.

Библиотека БГУИР

5. Разработать приложение учета отгрузки стройматериалов.
6. Разработать приложение учета закупок оргтехники подразделениями университета.
7. Разработать приложение учета закупки канцелярских товаров кафедрами.
8. Разработать приложение учета продажи компакт-дисков.
9. Разработать приложение учета сдачи студентами лабораторных работ.
10. Разработать приложение учета посещаемости лекций студентами.
11. Разработать приложение учета сдачи спортивных нормативов студентами.
12. Разработать приложение учета регистрации абитуриентов.
13. Разработать приложение учета регистрации книг в библиотеке.
14. Разработать приложение учета отгрузки товаров на складе.
15. Разработать приложение учета отправки писем адресатам в организации.

Вопросы для самопроверки

1. Особенности архитектуры ADO.NET .
2. Отсоединенные объекты.
3. Для чего предназначен объект DataTable?
4. Для чего предназначены методы Fill и Update и что они получают в качестве параметров?
5. Для чего предназначен объект Parameter?
6. Для чего предназначен объект DataSet?
7. Для чего предназначен объект Connection?
8. Каких поставщиков данных можно использовать в ADO.NET?
9. В чем заключается различие между поставщиками данных OLE DB и ODBC?
10. С помощью какого элемента управления в форме отображаются данные, хранящиеся в таблице базы данных?

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №6
РАЗРАБОТКА WEB-ПРИЛОЖЕНИЯ, ИСПОЛЬЗУЮЩЕГО ТЕХНЛОГИЮ ASP.NET

Цель: приобрести навыки разработки web-приложений с использованием технологии ASP.NET.

6.1. Теоретическая часть

6.1.1. Обзор ASP.NET Framework

ASP.NET – это технология создания web-приложений и web-служб от компании Майкрософт. Она является составной частью платформы Microsoft .NET и развитием более старой технологии Microsoft ASP. Кроме совместимости синтаксиса ASP.NET и ASP, новая платформа предоставляет новую модель программирования и инфраструктуру для более безопасных, масштабируемых и стабильных приложений. Можно легко дополнить существующее приложение ASP, последовательно добавив в него функциональные возможности ASP.NET.

ASP.NET является компилируемой средой на основе технологии .NET, в которой доступно создание приложений на любом совместимом с .NET языке, включая Visual Basic .NET, C# и JScript .NET. Кроме того, среда .NET Framework полностью доступна для любого приложения ASP.NET. Разработчики могут использовать все преимущества этих технологий, включающих в себя управляемую общеязыковую среду выполнения, строгую типизацию, наследование и т. д.

ASP.NET – часть глобальной платформы .NET. Эта платформа – часть новой стратегии Microsoft, соответствующая всем современным стандартам разработки как распределенных систем, так и настольных приложений.

.NET Framework предоставляет интерфейс приложениям, сама непосредственно взаимодействуя с операционной системой. Выше лежит интерфейс ASP.NET приложений, на котором в свою очередь базируются web-формы (ASP.NET страницы) и web-сервисы. Интерфейс .NET Framework позволяет стандартизировать обращение к системным вызовам и предоставляет среду для более быстрой и удобной разработки. В новую платформу встроены такие необходимые возможности, как контроль версий и важная для сетевых решений повышенная безопасность. Среда выполнения кода включает в себя сборщик мусора и набор библиотек, готовых к использованию.

Код для .NET Framework компилируется в общий промежуточный язык (Intermediate Language-IL). В случае ASP.NET код компилируется при первом обращении к странице и сохраняется для последующих вызовов. При выполнении оболочка компилирует промежуточный код в бинарный и выполняет его.

Кэширование готового бинарного кода позволяет улучшить эффективность. Intermediate Language позволяет создавать системы на любом удобном

Библиотека БГУИР

языке. И независимо от того, используется C#, VB.NET, JScript.NET или Perl.NET, в результате получается код, готовый к выполнению.

.NET Framework предоставляет и общий интерфейс обращения к базам данных – ADO.NET. Он тесно интегрирован с XML, что дает дополнительные преимущества при разработке распределенных приложений.

6.1.2. Обзор web-форм

Web-формы – это новая для ASP технология. web-формы введены для удобства разработки приложений. Они позволяют создавать компоненты интерфейса пользователя для многократного использования, что упрощает работу разработчика. web-элементы управления инкапсулируют html код, что позволяет писать код с более четкой логической структурой. Наконец, элементы управления упрощают создание средств WYSIWYG-разработки. Из всех средств, тестировавшихся нами ранее для ASP, только Dreamweaver UltraDev позволял это в приемлемой мере, и то, по сравнению со средствами RAD-разработки, его возможности по быстрому созданию качественного кода не впечатляли.

Web-формы – это обычные страницы. Помимо динамического содержания, создаваемого вашим кодом, вы можете включать в них web-элементы управления.

Итак, простейшая web-форма:

```
simple_form.aspx
<html>
  <head>
  </head>
  <script language="C#" runat="server">
    void SubmitBtn_Click(Object sender, EventArgs e) {
      Message.Text = "Hello, world!";
    }
  </script>
  <body>
    <center>
    <form method="post" runat="server">
      <asp:button type=submit text="hello" OnClick="SubmitBtn_Click"
runat="server"/><br>
      <asp:label id="Message" runat="server"/>
    </form>
    </center>
  </body>
    </html>
```

Заметьте, что у формы стоит свойство **runat=«server»**. При запросе страницы сервер обрабатывает такие элементы управления и выдает клиенту соответствующий html-код. Здесь же был написан простейший обработчик событий для web-форм. При нажатии на кнопку, когда форма отправляется на сервер, ASP.NET выполняет наш метод ***SubmitBtn_Click***. Это задается в свойстве

Библиотека БГУИР

OnClick кнопки submit. Наш же метод присваивает полю Text созданного нами элемента управления Message-текст.

Сорок восемь элементов управления поставляется с ASP.NET, они включают в себя различные компоненты от календаря до валидаторов. Кроме того, вы всегда можете написать свои для повторного использования, а также использовать созданные другими.

Наиболее часто вы, скорее всего, будете применять валидаторы и элементы управления, связанные с отображением данных. **DataGrid** позволит вам быстро вывести содержание выборки пользователю, в то время как **DataList** и **Repeater** позволят вам сделать это любым способом через шаблоны (поддерживаются шаблоны заголовков, футеров, самого куска данных и разделителя).

Покажем также пример использования валидатора.

form_validation.aspx

```
<html>
  <head>
  </head>
  <body>
     <form method="post" runat="server">
        <asp:textbox id="Text" runat="server"/><br>
        <asp:RequiredFieldValidator ControlToValidate="Text"
        Display="Dynamic" errormessage="You must enter text in control!"
runat=server/><br>
        <asp:button type=submit text="submit" runat="server"/>
     </form>
  </body>
</html>
```

В данном примере имеется одно текстовое поле и **RequiredFieldValidator**. Это самый простой из валидаторов, он проверяет, имеются ли в заданном ему поле (в данном случае оно называется Text) какие-нибудь данные. Если нет, и пользователь нажмет submit, то при проверке перед отправкой скрипт выведет сообщение об ошибке заполнения (вы сами указываете это сообщение) и форма не будет отправлена. Если же у пользователя старый браузер, то проверка будет произведена на сервере. В вашем коде вы можете проверить правильность заполнения всех полей с помощью поля **Page.IsValid** и вывести суммарное сообщение об ошибках с помощью ***ValidationSummary***.

Упомянем о замечательной возможности – разделение кода и представления, которая позволит разделить процессы разработки, упростить локализацию приложения и полностью использовать объектно-ориентированный подход. В данном случае нашу простейшую web-форму можно было бы переписать так:

form.aspx

```
<%@ Page Inherits="SimpleCode" Src="form.cs" %>
<html>
  <head>
```


Библиотека БГУИР

```
  </head>
  <body>
     <form method="post" runat="server">
        <asp:label id="text" runat="server"/><br>
        <asp:button type=submit text="submit" OnClick="SubmitBtn_Click"
runat="server"/><br>
     </form>
  </body>
</html>
```

form.cs

```
using System;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

public class SimpleCode : Page {
   public Label      text;
   public void SubmitBtn_Click(Object sender, EventArgs e) {
   text.Text = "text";
   }
}
```

6.1.3. Структура проекта и ASPX-файла

ASP.NET-проект, создаваемый в Microsoft Visual Studio, состоит из нескольких файлов. Вот основные из них:

Default.aspx	Экранная форма web-страницы
Default.aspx.cs	Исходный код формы

ASPX-файл является, по сути, обычным HTML, в котором можно использовать специальные теги. Дополнительные теги делятся на две категории: служебные, задающие параметры страницы и позволяющие внедрять код, и теги контролов, которые представляют собой новые интерфейсные конструкции, настройка которых инкапсулирована в тег.

Первому типу принадлежат серверные директивы, теги <%= %>, <% %>,<%# %>, <script runat="server">...</script>. Сначала рассмотрим директивы. Их синтаксис имеет вид

<% @ Directive %>

Базовой директивой является @ Page, которая описывает основные параметры страницы, такие, как файл с исходным текстом, язык кода, параметры трассировки. Также используются директивы @ Import, @ Assembly, @ OutputCache и др.

Библиотека БГУИР

Теги <%= %> используются для выставки вычисляемых значений. Например, конструкция

```
<%= System.DateTime.Now %>
```

после компиляции будет вместо себя подставлять текущее время.

Теги <% %> позволяют вставлять код в страницу – то, что в основном используется в ASP и JSP. Например

```
<%if (User.Identity.Name == "Admin")
      {
          %>
 <a href="adminpage.htm">Перейти к странице администрирования<a>
      <%          }
      %>
```

отображает ссылку только для пользователя Admin.

Конструкция <%= %> является аналогом <%= %> для компонент с привязкой к данным. Они подставляют вместо себя значение, зависящее от текущего контекста данных. Приведем пример, который, однако, здесь подробно разобран не будет: привязка к данным будет обсуждена в отдельном разделе.

```
<asp:Repeater id="lstData" runat="server" >
  <ItemTemplate>
    <%# (Container.DataItem as DateTime).ToShortDateString() %>
  </ItemTemplate>
 </asp:Repeater>
```

Также в страницу можно вставлять дополнительные методы с помощью тега <script runat="server">:

```
      <script language="C#" runat="server" >
       void SubmitBtn_Click( object sender, EventArgs e )
       {
         txtMessage.Text = "Hello, world";
       }
      </script>
```

Параметр language тега script задает язык, на котором он написан.

Теперь рассмотрим теги контролов – они позволяют вставлять в страницу различные элементы управления. Многие из вас сразу вспомнят про стандартные элементы управления вроде кнопок или полей ввода. На самом деле контролы ASP.NET могут представлять не только стандартные элементы, кодируемые одним HTML-тегом, но и довольно сложные DHTML-конструкции. При этом если при написании таких конструкций в обычном HTML вы получаете громоздкий и нерасширяемый код, то при использовании ASP.NET все это для вас инкапсулировано в одном теге. При этом код, который будет подставляться, генерируется на сервере, что позволяет изменять его в зависимости, например, от типа браузера.

Теги серверных контролов допустимы только внутри серверных форм – конструкций вида <form runat=«server»> ... </form>. Сами теги контролов пи-

Библиотека БГУИР

шутся в XML-стиле: с обязательным закрыванием и пространствами имен. Общий вид такого тега:

<пространство_имен:имя_тега runat="server" > ... </пространство_имен:имя_тега >

Пространства имен используются для того, чтобы случайно не совпали имена тегов. Стандартные элементы управления от Microsoft находятся в пространстве имен asp: <asp:Button />, <asp:TextBox /> и т.п.

6.1.4. Жизненный цикл страниц в ASP.NET

Жизненный цикл страницы ASP.NET начинается с получения и обработки Web-сервером IIS-запроса к данной странице и передачи этого запроса среде выполнения ASP.NET. В момент получения запроса среда выполнения загружает класс вызываемой страницы, устанавливает свойства класса страницы, выстраивает дерево элементов, заполняет свойства **Request** и **Response** и вызывает метод ***IHttpHandler.ProcessRequest()***. После этого среда выполнения проверяет, каким образом была вызвана эта страница, и если страница вызвана путем передачи данных с другой страницы, о чем будет рассказано далее, то среда выполнения устанавливает свойство **PreviousPage**. Стоит отметить также, что помимо рассмотренных ниже этапов выполнения страницы существуют еще и этапы уровня приложения, не специфичные для страницы (см. таблицу).

Этапы жизненного цикла приложения

Этап	Описание
1	**2**
Запрос страницы	Запрос страницы осуществляется до начала жизненного цикла страницы. Когда пользователь осуществляет запрос, среда выполнения ASP.NET устанавливает, необходимо осуществить компиляцию страницы и начать жизненный цикл либо можно выдать в качестве ответа страницу из кеша, таким образом не выполняя страницы
Начало жизненного цикла	На этом этапе происходит установка свойства Response и Request и свойства UICulture. Также на этом этапе устанавливается, была ли эта страница запрошена в результате постбэка (отправления данных на сервер), и соответствующим образом устанавливается свойство IsPostBack
Инициализация страницы	Ко времени инициализации страницы все дочерние пользовательские элементы управления уже созданы и имеют установленное свойство UniqueID. В это же время к странице применяются темы оформления. Если страница вызвана в результате постбэка, то данные, отправленные на сервер, еще не загружены в свойства элементов управления на этом этапе
Загрузка	Если страница вызвана в результате постбэка, то на этом этапе устанавливаются свойства элементов управления на основании информации о состоянии (ViewState и ControlState)

Библиотека БГУИР

Окончание таблицы

1	2
Валидация	Вызывается метод Validate() для всех находящихся на странице валидаторов
Обработка постбэка	Вызываются обработчики событий (при условии, что постбэк произошел)
Рендеринг	Сохраняется информация о состоянии, затем класс страницы вызывает соответствующие методы дочерних элементов управления для генерации HTML-представления и передачи его в Response.OutputStream
Выгрузка	Выгрузка происходит после того, как создано HTML-представление для всей страницы

Во время прохождения этапов жизненного цикла возникают события, подписавшись на которые разработчик может выполнять свой собственный код.

6.2. Практическая часть

Приступим к созданию нашего первого приложения на платформе ASP.NET. Запустим Visual Studio и выберем в меню **File** пункт **New Web Site** (рис.6.1):

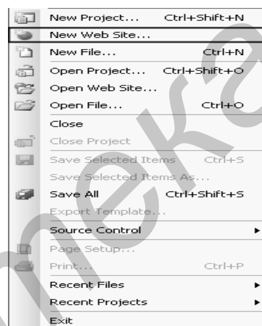

Рис. 6.1. Выбор создания нового web-сайта

В появившемся диалоге выберем тип проекта **ASP.NET Web Site**. В поле **Location** выбираем **File System**, что означает, что наш сайт будет храниться на жестком диске компьютера (если на вашей машине установлен сервер IIS, то в качестве месторасположения создаваемого сайта можно указать его). В качестве языка программирования в поле **Language** выбираем C# (рис. 6.2).

Библиотека БГУИР

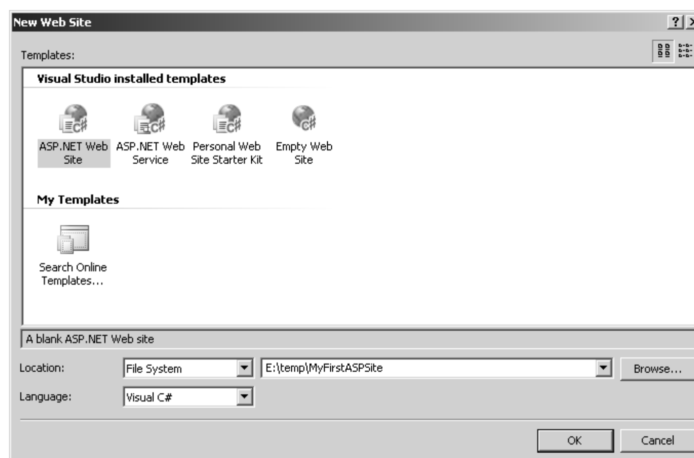

Рис. 6.2. Окно создания нового ASP.NET-проекта

Visual Studio создаст проект по умолчанию. Его структура будет видна в окне Solution Explorer (рис. 6.3).

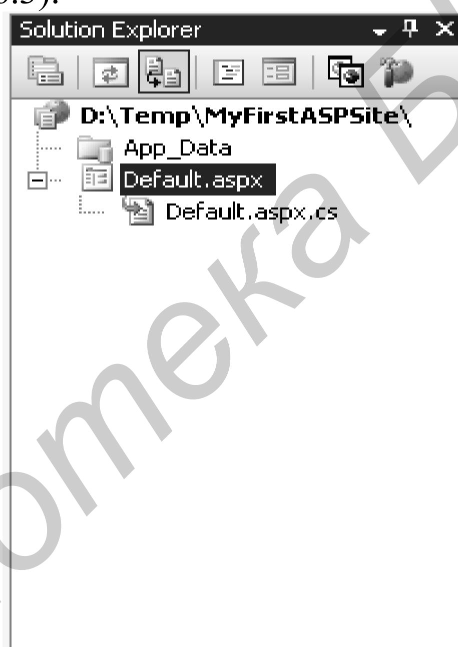

Рис. 6.3. Окно Solution Explorer созданного проекта

Здесь можно увидеть два файла:

Default.aspx	Экранная форма web-страницы
Default.aspx.cs	Исходный код формы

По сути, приложение создано. Но оно абсолютно ничего не делает. Добавим к нашему приложению простейший функционал. Для этого добавим на нашу форму поле ввода, метку и кнопку. Чтобы это сделать, откроем панель **Toolbox** (рис. 6.4).

Библиотека БГУИР

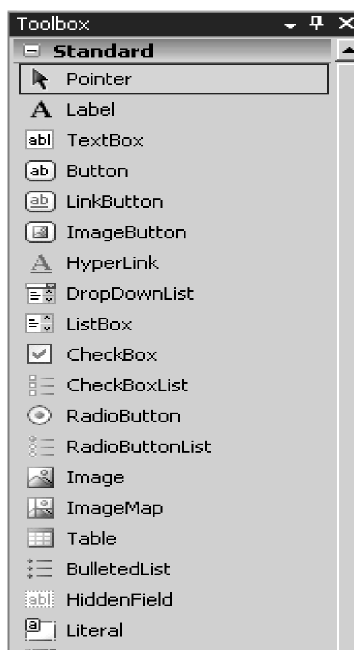

Рис. 6.4. Элементы управления, используемые в ASP.NET

Сначала добавим поле ввода. Для этого перетащим элемент **TextBox** с панели на нашу страницу. В панели свойств установим для нашего поля ввода идентификатор *txtName*.

Таким же способом добавим на страницу **Label** и кнопку **Button**. Присвоим метке ID – l*blMessage*, **Text** – пустую строку, а **Button** – *btnHello* и **«Поздороваться!»** соответственно. После этого наша страница примет следующий вид (рис. 6.5).

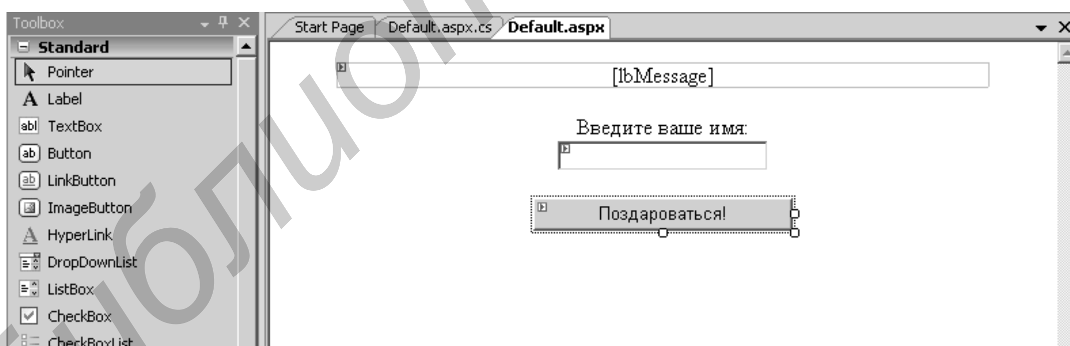

Рис. 6.5. Созданная web-форма

Теперь необходимо добавить некоторую логику. Наша страница будет здороваться с пользователем по имени, после того как он введет имя в поле ввода и нажмет кнопку. Для этого нужно написать обработчик события «пользователь нажал на кнопку» и вывести в метке соответствующий текст. Для создания обработчика дважды щелкните мышкой на кнопке. Visual Studio откроет файл с именем *WebForm1.aspx.cs*. К каждой форме невидимо привязываются файлы с *.aspx.cs* и *.aspx.resx*, но их не показывают в проекте, чтобы не нагружать его. В файле *.cs* содержится код страницы. После того как вы открыли его

Библиотека БГУИР

двойным щелчком на кнопке, в него добавился соответствующий обработчик. Введите в него следующий код:

```
protected void btnHello_Click(object sender, EventArgs e)
{
    lbMessage.Text = "Добрый день," + txtName.Text + " добро    пожаловать в ASP.NET!";
}
```

Теперь нужно скомпилировать проект. Это делается комбинацией клавиш Ctrl+Shift+B или командой меню **Build -> Build WebSite**.

После запуска приложения перед нами откроется окно web-браузера (рис. 6.6):

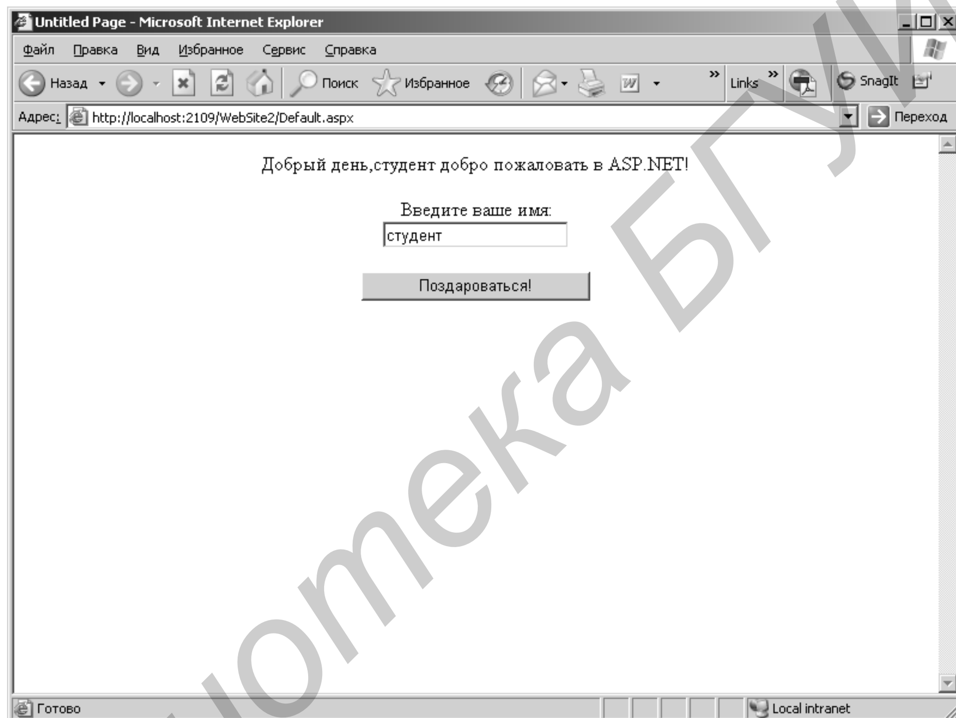

Рис. 6.6. Результат работы программы

Индивидуальные задания

1. Разработать web-страницу-календарь, которая отображала бы текущие дату и (или) время по выбору пользователя.
2. Разработать web-страницу с функциями калькулятора.
3. Разработать web-страницу, реализующую простейшую доску объявлений с фиксированным числом записей (не менее 7).
4. Разработать web-страницу, реализующую простейший учет продаж компьютеров. Данные хранятся в форме.
5. Разработать web-страницу, реализующую конвертер валют.
6. Разработать web-страницу, реализующую функцию подсчета годовых процентов по акциям.
7. Разработать web-страницу, реализующую подсчет скидок по товарам.

Библиотека БГУИР

8. Разработать web-страницу, реализующую расчет состояния автомобиля в зависимости от пробега.
9. Разработать web-страницу, реализующую функцию учета посещения студентами занятий. Предусмотреть использование DataGridView и подключение к источнику данных.
10. Разработать web-страницу, реализующую функцию учета закупки компьютеров кафедрами университета. Предусмотреть использование DataGridView и подключение к источнику данных.
11. Разработать web-страницу, реализующую функцию учета продажи авиабилетов в аэропорту. Предусмотреть использование DataGridView и подключение к источнику данных.
12. Разработать web-страницу, реализующую функцию учета продажи сотовых телефонов в салоне связи. Предусмотреть использование DataGridView и подключение к источнику данных.
13. Разработать web-страницу, реализующую функцию учета отпуска трикотажной продукции на фабрике. Предусмотреть использование DataGridView и подключение к источнику данных.
14. Разработать web-страницу, реализующую функцию учета посещения студентами занятий. Предусмотреть использование DataGridView и подключение к источнику данных.
15. Разработать web-страницу, реализующую функцию учета продажи билетов в кинотеатре. Предусмотреть использование DataGridView и подключение к источнику данных.

Вопросы для самопроверки

1. Что такое платформа ASP.NET?
2. Какие функции выполняет платформа ASP.NET?
3. Как происходит компиляция кода в ASP.NET?
4. Что такое web-форма?
5. Для чего предназначены web-формы и какие функции они выполняют?
6. Какие основные элементы управления используются на web-формах?
7. Какую структуру имеет ASPX-файл?
8. Для чего предназначена конструкция <%= %>?
9. На каких языках программирования возможно написание приложений для ASP.NET-платформы?
10. Назовите основные файлы, содержащиеся в создаваемом проекте ASP.NET?

Библиотека БГУИР

ЛИТЕРАТУРА

1. Хорстманн, К. Java 2. Библиотека профессионала. Т. 2 : Тонкости программирования / К. Хорстманн, Г. Корнелл. – М. : Издат. дом «Вильямс», 2010.
2. Java : основы Web-служб / Г. Беккет [и др.] ; пер. с англ. – М. : КУДИЦ-ОБРАЗ, 2004.
3. Шумаков, П. ADO.NET и создание приложений в среде Microsoft Visual Studio.Net / П. Шумаков. – М. : Диалог-МиФи, 2003.
4. Лабор, В. В. Си Шарп – создание приложений для Windows / В. В. Лабор. – Минск : Харвест, 2003.
5. Рихтер, Дж. Программирование на платформе .NET Framework / Дж. Рихтер. – М. : Русская редакция, 2003.
6. Мак-Дональд, М. Microsoft ASP.NET 3.5 с примерами на C# / М. Мак-Дональд, М. Шпушта. – М. : Издат. дом «Вильямс», 2010.
7. Рейли, Д. Создание приложений Microsoft ASP.NET / Д. Рейли. – М. : Русская редакция, 2002.

Библиотека БГУИР

Св. план 2011, поз. 134

Учебное издание

Комличенко Виталий Николаевич
Унучек Евгений Николаевич
Комаровский Антон Олегович
Кузьмицкий Владимир Михайлович

РАСПРЕДЕЛЕННЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ

Лабораторный практикум
для студентов специальности 1-40 01 02-02
«Информационные системы и технологии (в экономике)»
всех форм обучения

Редактор И. П. Острикова
Корректор Е. Н. Батурчик
Компьютерная верстка Ю. Ч. Клочкевич

Подписано в печать 20.03.2012. Формат 60x84 1/16. Бумага офсетная.
Гарнитура «Таймс». Отпечатано на ризографе. Усл. печ. л. 4,53.
Уч.-изд. л. 4,0. Тираж 150 экз. Заказ 324.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0494371 от 16.03.2009. ЛП №02330/0494175 от 03.04.2009.
220013, Минск, П. Бровки, 6

Библиотека БГУИР