# РАСШИРЕНИЕ ВОЗМОЖНОСТЕЙ ЯЗЫКА ПРОГРАММИРОВАНИЯ ЗА СЧЕТ ДОБАВЛЕНИЯ СРЕДСТВ ПОДДЕРЖКИ РАЗЛИЧНЫХ ПАРАДИГМ ПРОГРАММИРОВАНИЯ

Рябинкин Г. М., Деренчук В. И.
Факультет компьютерных систем и сетей, Белорусский государственный университет информатики и радиоэлектороники
Минск, Республика Беларусь
E-mail: herman.rabinkin@gmail.com

*Парадигма программирования, используемая при разработке программного продукта, определяет его внутреннюю структуру и зависит от используемого языка программирования и выбранного разработчиками подхода к организации кода. Но так ли тесно каждый конкретный язык связан с определенными парадигмами, с которыми его ассоциируют, и есть ли возможность расширить этот список? В докладе обобщается опыт использования языков программирования совместно с парадигмами, которые не поддерживаются ими напрямую, а также рассматривается целесообразность привнесения в язык средств поддержки дополнительных парадигм.*

## Введение

Сфера разработки программного обеспечения на современном этапе весьма богата разнообразными инструментами, применяющимися как в прикладном, так и системном программировании. Речь идет не только о вспомогательном программном обеспечении, используемом в процессе разработки, но и о самих языках программирования и поддерживаемых ими конструкциях.

Каждый язык программирования предоставляет программисту, кроме сложившейся вокруг него экосистемы разнообразных инструментов, языковые средства, позволяющие опираться на конкретные возможности языка в процессе создания программного обеспечения. Совокупность предоставляемых средств определяет природу языка и побуждает программиста писать код в том или ином стиле, то есть позволяет разработчику думать о своем приложении в определенных категориях. Часто в таком случае говорят о стиле или парадигме программирования, которые могут поддерживаться конкретным языком.

## I. Использование парадигмы без прямой поддержки средствами языка

Однако нередко из поля зрения исследователей исчезает тот факт, что парадигма программирования – это, в первую очередь, категория, определяемая отношением программиста к своему коду, его взглядом на программу, а не особенностями реализации того или иного языка программирования.

Для иллюстрации этого утверждения давайте рассмотрим практику написания кода на языке ассемблера. Исходя из набора имеющихся в нем конструкций, можно утверждать, что язык ассемблера предполагает возможность написания программ исключительно в рамках императивной парадигмы.

Вместе с этим, возможность оставлять метки в исходном тексте программы и их использование в качестве операндов некоторых команд позволяет программисту абстрагироваться от работы с адресами машинных инструкций и данных в оперативной памяти и мыслить категориями высокоуровневых языков программирования – переменными, составными операторами и т.д.

Кроме того, этот язык не имеет явных средств поддержки процедурного программирования, однако практика написания кода на ассемблере предполагает широкое использование подпрограмм, которые могут быть реализованы посредством операторов безусловного перехода и стека процессора.

Все это позволяет программисту размышлять о программе на более высоком уровне абстракции в категориях процедурного программирования, а не машинных команд, исполняемых непосредственно центральным процессором. Вместе с тем, этой возможности явно недостаточно, чтобы отнести язык ассемблера к процедурной парадигме, но и утверждать, что программа на ассемблере не может быть написана в процедурном стиле, исходя из вышеизложенного, мы тоже не имеем права.

Таким образом, мы можем говорить о связи языка программирования и некоторой парадигмы не только в категориях «поддерживает», «не поддерживает», но и расширить эту классификацию в зависимости от степени поддержки языком рассматриваемого стиля программирования.

## II. Взаимосвязь языка и парадигмы

Возможна и ситуация, когда средства языка, воспрещают организацию исходного текста программы в соответствии с определенной парадигмой. В качестве примера такой связи между языком программирования и парадигмой мож-

но привести классический BASIC [1], в котором нет возможности создавать локальные переменные внутри подпрограмм, что делает невозможным использование неразрушающих парадигм.

Следовательно, язык программирования может делать абсолютно невозможным использование парадигмы программирования (что встречается довольно редко), либо допускать ее применение с разной степенью удобства для программиста и предоставляемыми преимуществами от ее использования.

Говоря о связи языка с определенной парадигмой программирования, создатель C++ Бьерн Страуструп приводил некоторые критерии «совместимости», которые, при должном обобщении, можно свести к идее, что используемые для реализации определенного стиля конструкции должны быть естественны для языка и поддерживаться им по умолчанию, либо быть комбинацией поддерживаемых языком конструкций, а также не вести к возникновению дополнительных расходов [2–3].

Допустимость накладных расходов, связанных с использованием конструкций языка, можно оценивать в контексте круга решаемых программой задач. Так, приложения на относительно медленных интерпретируемых языках в общем случае не пострадают от создания дополнительных переменных или обращения к функции, созданной для повышения читаемости кода, а вот аналогичные накладные расходы для кода, написанного на языке C++, в некоторых случаях могут ощутимо сказаться на производительности и тем самым не позволить программисту использовать отдельные конструкции языка.

Таким образом, даже предоставляемые по умолчанию языком средства поддержки определенной парадигмы программирования не могут использоваться в абсолютно любом контексте, что говорит о еще более слабой связи между языком программирования и «поддерживаемыми» им парадигмами, а также влиянии на эту связь и некоторых внешних по отношению к системе «язык –парадигма» факторов. Равно можно и утверждать, что конструкции, не предоставляемые языком по умолчанию, но получаемые путем комбинации доступных конструкций, могут расширять список поддерживаемых данным языком парадигм при допустимости привнесенных в программу накладных расходов на их использование.

## III. Добавление в язык средств поддержки различных парадигм

Основываясь на общих требованиях к задачам, решаемым в области прикладного программирования, можно сказать, что в большинстве случаев накладные расходы, связанные с привнесением в язык новых конструкций, будут считаться несущественными.

Более того, в современной практике промышленного программирования широко распространены инструменты, расширяющие возможности языка за счет добавления средств поддержки различных парадигм программирования. Примером такой практики является современная экосистема, сложившаяся вокруг языка программирования Java, получившего признание в т.ч. благодаря поддержке развитых средств объектно-ориентированного программирования (далее – ООП). Де-факто, стандартом разработки приложений на этом языке стало использование фреймворка Spring [4], который навязывает разработчикам необходимость использовать аннотации, т.е. добавлять метаинформацию к компонентам приложения, которая в дальнейшем обрабатывается внутренними модулями фреймворка для выполнения неких операций в ходе исполнения программы.

Когда программист для задания поведения компонентов приложения не пытается его переопределить через характерные для ООП инструменты – наследование и полиморфизм, а использует аннотации, то он перестает думать о взаимодействии компонентов своего приложения и модулей фреймворка в категориях, характерных для ООП. Речь уже идет о взгляде на приложение через парадигму метапрограммирования, средств поддержки которой по умолчанию Java не предоставляет. Пример Spring показывает, насколько удобным и практичным может стать привнесение в язык средств поддержки слабо связанных с языком парадигм и их применение для решения специальных задач.

## Заключение

Таким образом, добавление в язык программирования средств поддержки различных парадигм, способно расширить возможности языка и сделать исходный текст программ более гибким, а также снизить трудозатраты на разработку сложных программных систем.

Данная возможность особенно актуальна для языков, широко применяемых при решении задач прикладного программирования – C#, Java, Python, JavaScript и др. Они, в отличие от условно низкоуровневых языков, применяемых для системного программирования, в значительно меньшей степени чувствительны к накладным расходам, связанным с добавлением в язык конструкций, не поддерживаемых им нативно.


1. Столяров, А. В. Программирование: введение в профессию. IV: Парадигмы / А. В. Столяров // Москва: МАКС Пресс, 2020. – С. 68-69.
2. Stroustrup, B. A Tour of C++ / B. Stroustrup // 2nd ed. – Boston : Addison-Wesley. – 2018. – P. 93–98.
3. Stroustrup, B. Programming – Principles and Practice Using C++ / B. Stroustrup // 2nd ed. – Boston : Addison-Wesley. – 2014. – P. 806–818.
4. Snyc – JVM Ecosystem Report 2021 [Electronic resource] / Ed. B. Vermeer. – Utrecht University M.S., 2021. – Mode of access: https://snyk.io/jvm-ecosystem-report-2021/. – Date of access: 03.10.2022.