С.Сейфуллиннің 125 жылдығына арналған **«Сейфуллин оқулары – 15: Жастар, ғылым, технологиялар: жаңа идеялар мен перспективалар»** атты халықаралық ғылыми-теориялық конференциясының материалдары = Материалы Международной научно-теоретической конференции **«Сейфуллинские чтения – 15: Молодежь, наука, технологии – новые идеи и перспективы»**, приуроченной к 125-летию С.Сейфуллина. -2019. - Т.II, Ч 1 - С.178-180

МНОГОЯЗЫЧНОЕ ПРОГРАММИРОВАНИЕ И ЯЗЫКИ ПРЕДМЕТНЫХ ОБЛАСТЕЙ

Сабитов Т.

Феномен языков предметных областей (они же – предметно-ориентированные языки или DSL) и, в особенности, текстовые языки, в настоящее время активно исследуется. DSL-языки предназначены для решения некоторых из проблем, которые можно решать и средствами многоязычного программирования. Такие методы сводятся к построению абстракций, оптимально приспособленных для решения конкретной проблемы. Тем не менее, языки DSL являются исключительно специализированными, создаются в рамках предметной области на основе конкретных практических случаев. Например, возвращаясь к описанной выше проблеме, можно написать предметно-ориентированный язык, который будет реализовывать функции планировщика в вашем приложении[1].

Ниже приведен пример простого предметно-ориентированного языка, который является внутренним DSL, написанным на базе C# (внутренний DSL – это предметно-ориентированный язык, целиком написанный на базе другого языка).

В компании ThoughtWorks нам как-то раз потребовалось написать приложение, которое должно было выдавать подробные характеристики железнодорожных вагонов (в целях тестирования). Сначала мы написали вот такой код: ICar car = new Car(); IMarketingDescription desc = new MarketingDescription(); desc.Type = "Box"; desc.Subtype = "Insulated"; desc.Length = 50.5; desc.Ladder = "Yes" desc.LiningType = Lining.Cork; desc.Description = desc; Но бизнес-аналитикам этот код не понравился.

Когда мы показали им код, они даже не попытались его читать, поскольку он слишком напоминал язык C#, а вникать в тонкости бизнес-аналитики не пожелали. Итак, мы переписали код следующим образом: ICar car = Car.describedAs() .Box .Insulated .Includes(Equipment.Ladder) .Has(Lining.Cork); Мы не применяли никаких особых магических фреймворков и расширений языка C#. Вместо того, чтобы создавать стандартные свойства, мы просто сделали свойства “Get” которые вызывали побочные «мутации» и создавали методы для установки значений (каждый из таких методов возвращал this). Добавьте несколько красивых отступов – и у вас получится значительно более удобочитаемый код. Это очень простой пример, но бывают и более сложные случаи[2].

На самом деле, LINQ – это просто внутренний предметно-ориентированный язык, обрабатывающий операции запроса структурированных данных. Многоязычное программирование и использование DSL – это скорее взаимно-дополняющие, а не антагонистичные подходы.

Ничего не мешает вам написать DSL как внутренний предметно-ориентированный язык (то есть, язык, полностью построенный на синтаксисе другого языка), воспользовавшись при этом функциональным языком, например, F#. Достаточно вспомнить хотя бы Scala – еще один функциональный язык, работающий на виртуальной машине Java. В свою очередь, язык .NET. включает ряд таких свойств, которые способствуют созданию DSL внутри языка1,2].

Нил Форд предположил, что в будущем стабильные языки программирования по Бини станут поддерживать функциональный стиль в большем объеме. Этот стиль программирования входит в моду: программы бухгалтерского учета пока еще не пишут на Haskell, но многие языки начинают поддерживать операции с функциями высшего порядка. Такие понятия, как «декларативный», «чистые функции», «кадрирование» проникают в лексикон все большего количества программистов.

Форд считает, что в будущем типизация уже не будет играть столь большую роль, определяющим фактором станет чистота реализации функциональной парадигмы. Типизация останется, однако будет прерогативой самого программиста.

Современные языки идут по пути поддержки мультипарадигмальности. ЯП, исторически поддерживающие парадигму процедурного и объектно-ориентированного (ОО) программирования, начинают вводить элементы поддержки парадигмы функционального стиля. Функциональные языки, наоборот, расширяют свои возможности, вводя поддержку ОО-парадигмы[2,3].

Мультипарадигмальность увеличивает мощь языка, но может и привести к проблемам. Когда один проект разрабатывается разными командами с использованием различных парадигм, то существует риск разработки несовместимых библиотек. Разработка в ОО-парадигме стимулирует использование структур, а в функциональной – композицию и функции высшего порядка. В результате смешения парадигм могут получиться различающиеся алгоритмы, с которыми нельзя работать без взаимной адаптации. Эти условия усложнят проект. Подобные проблемы команды разработчиков уже испытывали при переходе с Java на Ruby или с C на C++.

В сложившейся ситуации программисту недостаточно знать только один язык программирования. Работая над проектом на одном языке, нельзя исключать вероятность, что код придется писать и на другом ЯП, выбранном для решения соответствующих задач. Так какой же язык имеет смысл знать или изучать? Однозначного ответа на этот вопрос нет.

Во всех языках помимо различий есть и общее. Знание сходства позволят сэкономить время вхождения в новый язык, сконцентрировавшись лишь на различиях.

Общим для всех языков является синтаксис, а различие кроется в семантике. В любом языке могут быть структурные синтаксические конструкции, условия, вызовы, структуры данных, классы и т.д. Несмотря на кажущуюся одинаковость синтаксических конструкций, наличие семантических различий может привести к ошибкам.

В языке C++ и в Java есть определение классов. Одна и та же синтаксическая конструкция «ТКласс Класс» в первом случае определит ссылку на объект указанного класса, а во втором – создаст сам объект. Попытка работать на C++ со ссылкой как с инициализированным объектом приведет к сбою программы.

Список литературы

1. Абрамов, С.А. Математические построения и программирование / С.А. Абрамов. - М.: Наука, 2016. - 192 с.
2. Бекишев, Г.А. Элементарное введение в геометрическое программирование / Г.А. Бекишев, М.И. Кратко. - М.: Наука. Главная редакция физико-математической литературы, 2017. - 144 с
3. U.S. Department of Defense, The DoD Cyber Strategy, April 2015, Washington, DC. World Economic Forum, Partnering for C.

Научный руководитель: старший преподаватель кафедры ИКТ
Джумагалиева А.М.