ORES

Научно-практический электронный журнал
Оригинальные исследования (ОРИС), *ISSN 2222-0402*
www.ores.su
original-research.ru

УДК 519.6

ОБ ОСНОВНЫХ ПАРАДИГМАХ СОВРЕМЕННОГО ПРОГРАММИРОВАНИЯ

Смольянинова Мария Олеговна,
Студент, 2 курс, физико-математический факультет
Воронежский государственный педагогический университет
Россия, г. Воронеж

Сидорова Оксана Анатольевна,
Кандидат педагогических наук
Воронежский государственный педагогический университет
Россия, г. Воронеж

Аннотация

В данной статье рассматриваются основные парадигмы программирования. Описаны особенности императивного, функционального, декларативного, объектно-ориентированного программирования.

Ключевые слова: программирование, парадигмы, объектно-ориентированный стиль, функциональное программирование, императивное программирование.

ABOUT THE MAIN PARADIGMS OF MODERN PROGRAMMING

Maria O. Smolyaninova,
Student, 2nd year, Faculty of Physics and Mathematics
Voronezh State Pedagogical University
Russia, Voronezh

Oksana A. Sidorova
Candidate of Pedagogical Sciences
Voronezh State Pedagogical University
Russia, Voronezh

ABSTRACT

This article discusses the main programming paradigms. The features of imperative, functional, declarative, object-oriented programming are described.

Keywords: programming, paradigms, object- oriented style, functional programming, imperative programming.

В современном мире изучение парадигм программирования имеет большое значение и актуально по нескольким причинам. Знание основ различных парадигм позволяет расширить набор инструментов и подходов при разработке программного обеспечения, что ведет к более эффективным и инновационным решениям и позволяет программистам быть гибкими и адаптироваться к изменяющимся требованиям, выбирая наиболее подходящие парадигмы для решения конкретных задач. Освоение парадигм программирования требует от программистов углубленного понимания концепций, принципов и особенностей каждой парадигмы. Это помогает развить навыки анализа, проектирования и разработки программного обеспечения. Также знание парадигм помогает работающим с кодом лучше понимать друг друга и сотрудничать более успешно. Разные парадигмы могут использоваться разными людьми или командами в одном проекте, и понимание различий между ними помогает улучшить коммуникацию и сотрудничество.

Парадигмы программирования — это различные подходы, совокупность концепций и идей, которые определяют подход к организации и написанию компьютерных программ. Каждая парадигма имеет свои принципы и концепции, которые определяют способ представления и выполнения кода [1].

Парадигмы программирования определяются набором основных концепций, правил и идей, которые указывают на то, как следует подходить к процессу разработки программного кода. Они определяют, каким образом нужно организовывать и структурировать программы, какие инструменты и методы использовать, и какие принципы и подходы следует применять для достижения конкретных целей. Концепции и идеи парадигмы программирования также представляют собой решения для общих проблем, с которыми сталкиваются разработчики при написании программного кода.

Однозначное определение парадигмы программирования не связано с конкретным языком программирования; почти все современные языки программирования позволяют использовать различные парадигмы в разной степени (мультипарадигменное программирование) [2].

На данном этапе развития программирования выделяют основные его парадигмы: императивно-процедурное, декларативное, функциональное, логическое, объектно-ориентированное и т. п.

Императивная парадигма программирования является одной из самых основных и широко используемых парадигм. Она основывается на концепции последовательного выполнения инструкций, которые изменяют состояние программы. В императивной парадигме программа описывается как последовательность команд, которые выполняются с определенной последовательностью и изменяют состояние программы в соответствии с заданными правилами. Основной управляющей структурой является последовательность, которая предписывает порядок выполнения команд. Также в императивной парадигме используются условные выражения и циклы, которые позволяют осуществить ветвление и повторение операций, соответственно. Императивное программирование часто используется в различных областях, включая разработку операционных систем, системного программирования, написание алгоритмов и других задач, требующих точного контроля выполнения инструкций.

Основными преимуществами императивной парадигмы являются простота понимания и использования, контроль изменения состояния программы. Императивная парадигма следует привычному человеку порядку мышления и последовательному выполнению действий. При использовании императивной парадигмы разработчик имеет

полный контроль над состоянием программы и может точно определить последовательность операций.

Однако среди плюсов императивной парадигмы выделяются и недостатки. Императивное программирование может быть сложно масштабировать и параллелизировать, поскольку оно ориентировано на последовательность операций. При разработке больших и сложных программ императивная парадигма может стать непрактичной из-за сложности управления изменениями состояния и контроля.

Тем не менее, императивная парадигма программирования остается важной и широко применяемой, особенно в задачах, требующих структурированного и последовательного выполнения операций.

Декларативное программирование — это парадигма программирования, в которой программист описывает, что должна делать программа, а не как это делать. В декларативном программировании мы фокусируемся на описании желаемых результатов, а не на последовательности шагов для их достижения. При этом, необходимые операции и логика вычислений скрываются от программиста за абстракциями [1].

Преимуществами декларативного программирования является читаемость и понятность кода. Декларативные программы часто написаны более формализировано, что делает их более понятными для программистов и облегчает сопровождение кода. В декларативном программировании программист описывает, что нужно сделать, а не как это сделать. Это позволяет сокрыть детали реализации и делегировать их исполнение системе или библиотекам, что упрощает разработку и улучшает масштабируемость программы. Поскольку порядок вычислений и множество деталей реализации скрыты, система может оптимизировать программу автоматически, повышая ее производительность.

Однако, декларативное программирование не подходит для всех задач. В декларативных языках часто невозможно описать сложные последовательности операций или управлять состоянием программы. Также, декларативное программирование требует от программиста понимания принципов работы системы, которая будет исполнять описанный код [3].

Функциональное программирование (ФП) — это парадигма программирования, которая сосредоточена на составлении программ из функций. Основная идея состоит в том, чтобы рассматривать программу как серию математических функций, которые принимают аргументы и возвращают значения, а не как последовательность команд, изменяющих состояние. В отличие от императивного программирования, где основными концепциями являются изменяемые переменные и последовательные операции, ФП строится на неизменяемости данных и отсутствии побочных эффектов. Это означает, что функции в ФП не зависят от состояния программы и всегда возвращают одинаковое значение для одних и тех же аргументов.

Функциональное программирование ставит своей целью разделение ответственности в программе на отдельные функции и минимизацию побочных эффектов. Вместо того, чтобы изменять состояние программы, ФП ставит акцент на преобразование данных через функции. Это позволяет создавать программы, которые проще понимать, тестировать и разрабатывать.

Основными принципами ФП являются неизменяемость данных после создания, использование функций вместо строительных блоков, функции в не должны иметь побочных эффектов, таких как изменение глобальных переменных или ввод-вывод. Это позволяет структурировать программу в виде чистых функций, которые всегда возвращают одинаковое значение для одних и тех же аргументов. Использование рекурсии вместо циклов и Лямбда-функции, использование высокоуровневых операций, таких как

отображение (map), фильтрация (filter), свёртка (reduce) для обработки списка или коллекции данных.

Преимуществами функционального программирования являются более простой и понятный код, так как функции рассматриваются как математические выражения и не имеют побочных эффектов, лучшая сопровождаемость и тестируемость программы, так как отсутствие побочных эффектов делает процесс отладки и тестирования более предсказуемым, чистые функции могут быть легко комбинированы и переиспользованы, что способствует повторному использованию кода и модульности, параллелизация кода легче, так как функции не зависят от состояния программы.

Однако, функциональное программирование также имеет свои недостатки. Чистые функции требуют больше памяти, так как они создают новые значения вместо изменения существующих. Настройки среды разработки и инфраструктуры для функционального программирования может быть высокой. К тому же не все задачи хорошо подходят для функционального программирования, особенно те, которые требуют множественных изменений состояния.

В целом функциональное программирование представляет собой мощную и элегантную парадигму программирования, которая может быть особенно полезной для написания чистого, понятного и модульного кода. Оно может быть применимо во многих задачах программирования, особенно в областях, где требуется большая надежность и дальнейшая сопровождаемость кода.

Еще одним не менее популярным и полезным является объектно-ориентированное программирование. Объектно-ориентированное программирование (ООП) — это парадигма программирования, которая базируется на концепции объектов, которые являются экземплярами классов, и связанных с ними методов и свойств.

В объектно-ориентированном программировании, данные и код объединяются вместе в единую сущность, которая называется объектом. Объекты могут быть созданы на основе шаблона, называемого классом, который определяет структуру и поведение объекта.

ООП строится на основных принципах:

Инкапсуляция: ООП поддерживает сокрытие информации и реализацию через объединение данных и методов в одну единицу, называемую классом. Доступ к данным осуществляется только через методы класса, что позволяет контролировать доступ к ним и защищает от внесения их неправильных изменений.

Наследование: ООП позволяет создавать классы на основе уже существующих, добавляя или изменяя их функциональность. Класс, созданный на основе другого класса, называется производным классом, а родительский класс - базовым классом. Наследование позволяет повторно использовать код и создавать иерархии классов.

Полиморфизм: ООП поддерживает использование одного и того же интерфейса для множества различных типов. Например, у разных классов может быть одинаковый метод, который будет работать по-разному для каждого класса. Полиморфизм позволяет упрощать код и улучшать его читаемость и расширяемость.

К плюсам объектно-ориентированного программирования относят: модульность и повторное использование кода: ООП позволяет разбить программу на отдельные компоненты (классы), которые можно легко модифицировать и использовать в других проектах. Данная парадигма позволяет создавать код, который легко читать и понимать, что упрощает разработку и отладку программы. Также ООП имеет возможность создавать иерархию классов, в которой классы-наследники могут наследовать свойства и методы родительских классов. Данная парадигма позволяет легко добавлять новую функциональность в программу, модифицируя или расширяя существующие классы.

К минусам использования данной парадигмы относят высокую абстрактность: ООП может быть сложным для понимания и изучения, особенно для новичков в программировании. При использовании ООП может возникнуть некоторая потеря производительности из-за дополнительных слоев абстракции и динамического разрешения методов. Внутренние связи и зависимости между классами могут сложить отладку кода при возникновении ошибок. Создание экземпляров классов может потребовать большого объема памяти, особенно если включены наследование и полиморфизм. Использование ООП требует особого подхода к разработке программы, что может быть сложным для неподготовленных разработчиков.

Объектно-ориентированное программирование представляет собой мощный подход для разработки программного обеспечения, который облегчает организацию и управление кодом, позволяет повторно использовать его, а также делает программы более модульными и расширяемыми. ООП является одной из самых распространенных парадигм программирования в современной разработке программного обеспечения [3].

Таким образом, каждая парадигма имеет свои преимущества и недостатки, и выбор конкретной парадигмы зависит от решаемой задачи, требуемой производительности и командных предпочтений. Среди выделенных парадигм, наиболее удобной для новичков является императивный стиль написания программ, в то время как функциональное программирование и ООП позволяют значительно увеличить производительность кода и наиболее подходят для работы с большим количеством данных.

Список литературы:

1. Городняя, Л. В. Парадигма программирования: курс лекций / Л.В. Городняя; Новосиб. гос. Ун-т. – Новосибирск: РИЦ НГУ, 2015. 206 с. 7.
2. Городняя, Л. В. Парадигмы параллельного программирования в университетских образовательных программах и специализации // Всероссийская научная конференция «Научный сервис в сети Интернет: решение больших задач». – Новороссийск-Москва, 2008. с. 180-184.
3. Шилов, Н. В. Заметки о трёх парадигмах программирования //КИО. - 2010.- №2.

References:

1. Gorodnya, L. V. Programming paradigm: a course of lectures / L.V. Gorodnya; Novosibirsk State University. Novosibirsk: RIC NSU, 2015. 206 p. 7.
2. Gorodnya, L. V. Paradigms of parallel programming in university educational programs and specializations // All-Russian scientific conference "Scientific service on the Internet: solving big problems". – Novorossiysk-Moscow, 2008. pp. 180-184.
3. Shilov, N. V. Notes on three programming paradigms //KIO. - 2010.- No. 2.