ПРОГРАММИРОВАНИЕ

# Парадигмы программирования

Парадигмы программирования – свод принципов, правил, стиля и возможностей реализации при написании кода. Существует большое количество методологий для написания программ, которые использовались при разработке языков программирования. Данные парадигмы использовались для обеспечения возможности разработки необходимых продуктов с оглядкой на современные технологоии и их возможности решать определенные задачи.



Рисунок n – многообразие языков программирования, разработанных с использованием указанных парадигм

Современные языки программирования не используют конкретную парадигму, но опираются на особенности некоторых из них. Рассмотрим наиболее используемые принципы разработки и языки, в которых они используются.

**Императивное программирование**

В данном принципе разработчик фокусируется на том, как решить определенную задачу – каков алгоритм получения результата.

Задача: программа получает массив целых чисел и должна преобразовать каждый элемент – увеличить его значение на 2. То есть ожидается преобразование типа: [1, 2, 3] -> [2, 4, 6].

var numbers: Array<Int> = arrayOf(1,2,3)  
  
println(numbers.joinToString())  
  
var result: Array<Int> = Array<Int>(3) { 0 }  
  
for ((index, elem) in numbers.withIndex()) {  
 result[index] = elem \* 2;  
}  
  
println(result.joinToString())

В логике данной парадигмы важны инструкции при помощи которых разработчик получает результат. Шаги должны быть понятными и последовательными. Так же последовательно должно происходить мутирование данных.

**Декларативное программирование**

Декларативное программирование выделяет не столько путь к достижению результата, сколько сам результат.

Все программы так или иначе, производят операции с данными, которые записываются в память. В процессе исполнения, программа обращается в необходимые ячейки памяти. Эти ячейки имеют адрес, именно по нему программа получает доступ к данным.

В процессе раюоты, ОС Android отслеживает состояние памяти и проводит операции для предотвращения перегрузки. Это означает, что данные могут перезаписываться в другие места. Как следствие – изменяется адрес к ним. Однако, разработчику нет необходимости заниматься отслеживанием адресов данных.

Базовый элемент для отслеживания адреса конкретных данных – переменная.

Переменная – именованная область памяти, в которую пишется адрес ячейки с информацией. По данному адресу происходит получение, запись, мутирование данных и т.д.

Разработчику необходимо объявить переменную и передать данные для записи. Определение ячеек и адреса операционная система выполняет сама.

В языках Java и Kotlin **объявление переменной** предполагает описание:

- типа значений

- имени переменной.

Языки со строгой типизацией диктуют необходимость указания типа данных при объявлении переменной. Именно поэтому в таких языках как Java и Kotlin разработчик обязан указывать тип новой переменной при ее объявлении.

При определении типа данных, система рассчитывает необходимое количество ячеек. Например, количество памяти для целочисленных значений меньше, чем для чисел с плавающей точкой.

Есть так же разделение данных на примитивные типы (целочисленные, числа с плавающей точкой, логические, символы) и ссылочные типы – указывают на расположение объектов классов или данных. Описание второго типа данных рассмотрим ниже.

После того, как переменная объявлена, для дальнейшей работы с ней необходимо присвоить ей значение через оператор присваивания =. Такую операцию называют инициализацией переменной - первое присвоение значения переменной после (или во время) ее объявления.

myAge: Int = 18;