Московский государственный технический университет им. Н.Э. Баумана.

Факультет «Информатика и управление»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Домашнему заданию
«Изучение языка программирования Kotlin»

Выполнил:
студент группы ИУ5-32Б
Плюшко Дмитрий Андреевич

Проверил:
преподаватель каф. ИУ5

Подпись и дата:

Подпись и дата:

Москва, 2024 г.

Задание

Выберите язык программирования (который Вы ранее не изучали) и напишите по нему реферат с примерами кода

Необходимо установить на свой компьютер компилятор (интерпретатор, транспилятор) этого языка и произвольную среду разработки

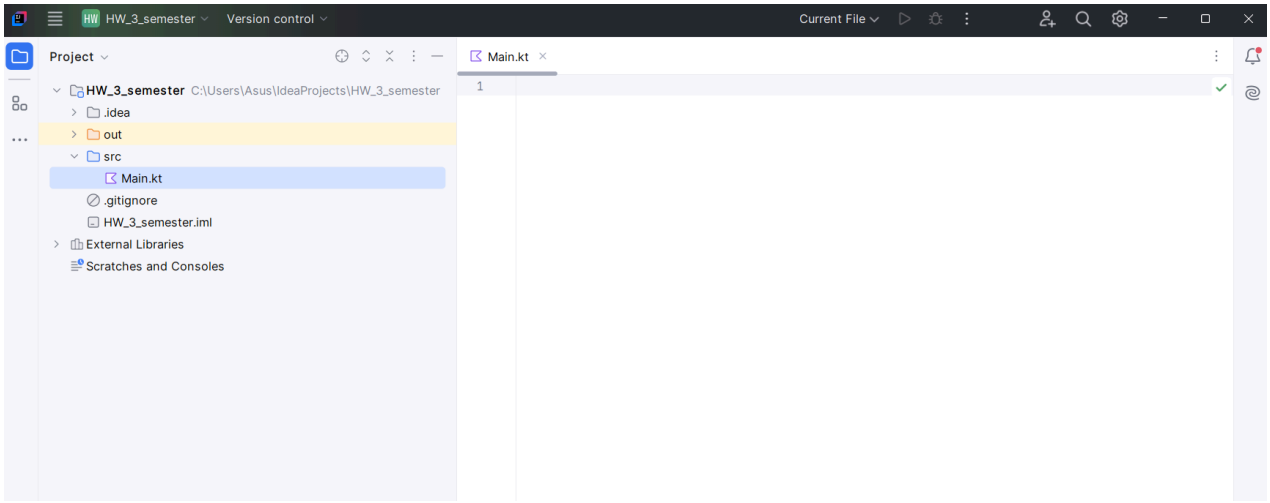
В случае написания реферата необходимо разработать и откомпилировать примеры кода

При написании реферата необходимо изучить и корректно использовать особенности парадигмы языка и основных конструкций данного языка.

Изучение языка программирования Kotlin

Kotlin — это современный язык программирования, разработанный компанией JetBrains, который стал официальным языком для разработки приложений на платформе Android.

Установка среды программирования IntelliJ IDEA и создание проекта в нем для компиляции написанного кода на языке Kotlin



Одной из ключевых особенностей Kotlin является статическая типизация, что позволяет выявлять ошибки на этапе компиляции, а не во время выполнения программы. Второй особенностью является система типов, которая помогает избежать распространенных ошибок, связанных с нулевыми значениями. В Kotlin есть возможность явно указывать, может ли переменная принимать значение null или нет, что значительно снижает вероятность возникновения исключений NullPointerException. Это делает код более безопасным и предсказуемым.

```
fun main() {
```

```
    // var nonNullString: String = "Hello!" // Нельзя присвоить
null
    // nonNullString = null // Это вызовет ошибку компиляции
    var nullString: String? = "Hello!" // Можно присвоить null
    println(nullString)

    nullString = null
    println(nullString) // Вывод: null

    // Использование безопасного вызова (?.) для предотвращения
NullPointerException
    val length: Int? = nullString?.length // Вывод: null
    println(length)
}
```

Вывод кода

```
Hello!
null
null
```

Ненулевая переменная: nonNullString объявлена как ненулевая переменная, и попытка присвоить ей значение null приведет к ошибке компиляции.

Нулевая переменная: nullString может принимать значение null, что демонстрирует возможность работы с переменными, которые могут быть пустыми.

Безопасный вызов: Использование оператора ?. позволяет безопасно обращаться к свойствам или методам переменной, которая может быть null, предотвращая возникновение исключений.

Также Kotlin полностью совместим с Java, что позволяет разработчикам использовать существующие библиотеки и фреймворки без необходимости переписывать код. Это делает переход на Kotlin для команд, уже работающих с Java, более плавным и менее затратным. Язык поддерживает объектно-ориентированное программирование, а также функциональные концепции, такие как высшие функции и лямбда-выражения, что позволяет писать более выразительный и компактный код.

```
fun main() {
    // Пример высшей функции, которая принимает другую функцию в качестве параметра
    fun performOperation(a: Int, b: Int, operation: (Int, Int) -> Int): Int {
        return operation(a, b)
    }

    // Определение лямбда-выражения
    val add: (Int, Int) -> Int = { x, y -> x + y }
    val subtract: (Int, Int) -> Int = { x, y -> x - y }

    // Использование высшей функции с лямбда-выражениями
    val sum = performOperation(5, 3, add) // 5 + 3
    val difference = performOperation(5, 3, subtract)

    println("Сумма: $sum") // Вывод: Сумма: 8
    println("Разность: $difference") // Разность: 2

    // Пример использования лямбда-выражения непосредственно в вызове функции
    val multiplication = performOperation(5, 3) { x, y -> x * y }
    println("Произведение: $multiplication")
}
```

Вывод кода

```
Сумма: 8
Разность: 2
Произведение: 15
```

Высшая функция: performOperation — это функция, которая принимает два целых числа и другую функцию (операцию) в качестве параметров. Эта функция выполняет переданную операцию над двумя числами.

Лямбда-выражения: add и subtract — это лямбда-выражения, которые определяют операции сложения и вычитания соответственно.

Вызов высшей функции: performOperation вызывается с различными лямбда-выражениями, что демонстрирует возможность передачи функций как параметров.

Лямбда-выражение в вызове функции: В последнем вызове performOperation лямбда-выражение для умножения передается непосредственно в качестве аргумента, что делает код более компактным и удобным.

Kotlin поощряет использование неизменяемых объектов. Язык предлагает множество встроенных функций для работы с коллекциями, что упрощает манипуляции с данными. Например, функции расширения позволяют добавлять новые функции к существующим классам без необходимости наследования.

```
// Функция расширения для класса String
fun String.reverse(): String {
    return this.reversed() // Используем встроенную функцию reversed()
}

fun main() {
    val originalString = "Kotlin"
    // Вызываем функцию расширения
    val reversedString = originalString.reverse()

    println("Оригинальная строка: $originalString")
    println("Строка в обратном порядке: $reversedString")
}
```

Вывод кода

```
Оригинальная строка: Kotlin
Строка в обратном порядке: niltoK
```

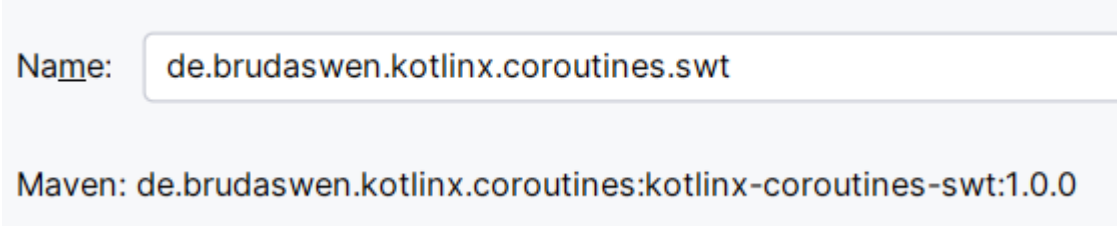
Функция расширения: Мы объявляем функцию reverse как расширение для класса String. Это делается с помощью синтаксиса fun String.reverse().

Тело функции: Внутри функции мы используем метод reversed(), который возвращает строку в обратном порядке.

Использование функции: В функции main мы создаем строку originalString и вызываем нашу функцию расширения reverse(), чтобы получить строку в обратном порядке.

Кроме того, Kotlin поддерживает корутины, которые упрощают работу с асинхронным программированием, позволяя писать код, который выглядит как последовательный, но выполняется асинхронно.

Для этого ставим библиотеку `kotlinx.coroutines`




```
import kotlinx.coroutines.*

fun main() = runBlocking {

    // Запускаем корутину
    println("Запуск асинхронной задачи...")
    launch {
        val result = thread() // Асинхронная операция
        println(result)
    }
    println("Асинхронная задача запущена")

    // Здесь можно выполнять другие операции
    delay(2000)
    println("Основной поток завершен")
}

// Функция, имитирующая асинхронную работу
suspend fun thread(): String {
    delay(1000)
    return "Дополнительный поток завершен"
}
```

Вывод кода

```
Запуск асинхронной задачи
Асинхронная задача запущена
Дополнительный поток завершен
Основной поток завершен
```

Основная корутина: Мы используем runBlocking, чтобы создать корутину, которая блокирует текущий поток до завершения всех вложенных корутин.

Запуск корутины: Внутри runBlocking мы используем launch, чтобы запустить новую корутину. Это позволяет нам выполнять асинхронную задачу.

Асинхронная функция: Функция thread помечена как suspend, что означает, что ее можно вызывать только из другой корутины. В этой функции мы используем delay, чтобы имитировать асинхронную операцию.

В заключение, Kotlin — современный мультиплатформенный язык программирования, который сочетает в себе черты объектно-ориентированного и функционального программирования.