Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана

(национальный исследовательский университет)»

(МГТУ им. Н.Э. Баумана)

Факультет «Радиотехнический»

Кафедра ИУ5 «Системы обработки информации и управления»

Отчет по ДЗ: “Основы языка Kotlin”

«Парадигмы и конструкции языков программирования»

Выполнил:

студент группы РТ5-31Б

Иванченко Д.А.

Проверил:

Гапанюк Ю. Е

2023 г.

Задание

1. Выберите язык программирования (который Вы ранее не изучали) и (1) напишите по нему реферат с примерами кода или (2) реализуйте на нем небольшой проект (с детальным текстовым описанием).

2. Реферат (проект) может быть посвящен отдельному аспекту (аспектам) языка или содержать решение какой-либо задачи на этом языке.

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

4. В случае написания реферата необходимо разработать и откомпилировать примеры кода (или модифицировать стандартные примеры).

5. В случае создания проекта необходимо детально комментировать код.

6. При написании реферата (создании проекта) необходимо изучить и корректно использовать особенности парадигмы языка и основных конструкций данного языка.

7. Приветствуется написание черновика статьи по результатам выполнения ДЗ. Черновик статьи может быть подготовлен группой студентов, которые исследовали один и тот же аспект в нескольких языках или решили одинаковую задачу на нескольких языках.

Основы языка Kotlin

Kotlin - современный, статически типизированный и один из самых быстроразвивающихся языков программирования, созданный и развиваемый компанией JetBrains. Наиболее часто Kotlin применяется в разработке приложений для Android, однако Kotlin позволяет также писать кроссплатформенный код, который будет применяться на всех платформах. Язык используется для создания веб-приложений, причем как для бекэнда, так и для фронтенда. Kotlin также можно применять для создания десктопных приложений, для Data Science и так далее.

Первая программа “hello world”

```
    fun main(){
    println("Hello world")
}
```

fun - ключевое слово определения функции

main - функция, являющаяся точкой входа в программу на Kotlin. Эта функция должна быть в любой программе на языке Kotlin

println - функция вывода в консоль

Комментарии

Для многострочных комментариев используются символы: /* *многострочный комментарий* */

Для однострочных: // *комментарий*

```
    /*
Многострочный
комментарий
 */

// Однострочный
```

Переменные

Для объявления переменных в Kotlin используются ключевые слова **val** и **var**

val определяет неизменяемый тип переменной

var соответственно – изменяемый

В общем случае, объявление переменной выглядит так:

val|var {имя_переменной}: {тип_переменной}

```
    fun main() {
    val a: Int
    var b: Int

    a = 33
    b = 21
    b = 2 // можно переопределить
    // a = 3 - выдаст ошибку, т.к. a - val
}
```

Также для определения постоянных значений используется **const val.** Константы на стадии компиляции должны иметь некоторое значение и объявляться на самом верхнем уровне.

```
    const val pi = 3.14 // константа
```

Типы данных

Целочисленные: **Byte** (1 байт), **Short** (2 байта), **Int** (4 байта), **Long** (8 байт)

Целочисленные без знака: **UByte** (1 байт), **UShort** (2 байта), **UInt** (4 байта), **ULong** (8 байт)

```
    fun main() {

    val a: Byte = -10
    val b: UInt = 250U

        val address: Int = 0x0A1    // 16-ричная запись 161
}
```

Числа с плавающей точкой: **Float** (4 байта), **Double** (8 байт)

Логический тип: **Boolean**

```
    val a: Boolean = true
val b: Boolean = false
```

Символы: **Char** (этот тип включает спец. символы, например \n – перевод строки)

Строки: **String**

Также в Kotlin есть тип **Any**, позволяющий присвоить переменной любое значение

```
    fun main() {

    var x: Any = "Str"
    x = 22
    x = 'a'
}
```

Операции над числами

Помимо базовых арифметических операций +, -, *. / в Kotlin есть операции остатка %, инкремента ++, декремента --. Также поддерживаются операции присвоения: +=. -=, *=, /=, %=.

Поразрядные операции (применяются только к данным типов Int и Long):

- **shl**: сдвиг битов числа со знаком влево
- **shr**: сдвиг битов числа со знаком вправо
- **ushr**: сдвиг битов беззнакового числа вправо
- **and**: побитовая операция AND (логическое умножение или конъюнкция). Эта операция сравнивает соответствующие разряды двух чисел и возвращает единицу, если эти разряды обоих чисел равны 1. Иначе возвращает 0.
- **or**: побитовая операция OR (логическое сложение или дизъюнкция). Эта операция сравнивают два соответствующих разряда обоих чисел и возвращает 1, если хотя бы один разряд равен 1. Если оба разряда равны 0, то возвращается 0.
- **xor**: побитовая операция XOR. Сравнивает два разряда и возвращает 1, если один из разрядов равен 1, а другой равен 0. Если оба разряда равны, то возвращается 0.
- **inv**: логическое отрицание или инверсия - инвертирует биты числа

Условные конструкции

Конструкция if…else

```
fun main() {
    val y = 3
    if (y > 5) {
        println("y больше 5")
    } else {
        println("y меньше или равен 5")
    }
}
```

if.. else if… else

```
fun main() {
    val z = 7
    if (z > 10) {
        println("z больше 10")
    } else if (z == 10) {
        println("z равен 10")
    } else {
        println("z меньше 10")
    }
}
```

Также if может использоваться как часть выражения

```
fun main() {
    val a = 15
    val result = if (a > 10) "a больше 10" else "a меньше или равен 10"
    println(result)
}
```

Конструкция when. Используется в основном для условий с определённой конечной и известной выборкой значений

```
fun main() {
    val b = 3
    when (b) {
        1 -> println("b равен 1")
        2 -> println("b равен 2")
        else -> println("b не равен ни 1, ни 2") // else использовать не
обязательно
    }
}
```

Также when можно использовать для диапазонов с помощью оператора in.

```
val a = 10
when(a){
    in 10..19 -> println("a в диапазоне от 10 до 19")
    in 20..29 -> println("a в диапазоне от 20 до 29")
    !in 10..20 -> println("a вне диапазона от 10 до 20")
    else -> println("неопределенное значение")
}
```

Аналогично if, when может использоваться как часть выражения

```
val sum = 100

val rate = when(sum){
    in 10..99 -> 10
    in 100..999 -> 15
    else -> 20
}
```

Циклы

В Kotlin, циклы представлены ключевыми словами for, while и do…while.

Цикл for для перебора элементов в коллекции:

```
val numbers = listOf(1, 2, 3, 4, 5)
for (number in numbers) {
    println(number)
}
```

Цикл for для перебора элементов в диапазоне:

```
for (i in 1..5) {
    println(i)
}
```

Цикл while для выполнения блока кода до тех пор, пока условие истинно:

```
var count = 0
while (count < 5) {
    println(count)
    count++
}
```

Цикл do…while для выполнения блока кода хотя бы один раз, а затем продолжения выполнения до тех пор, пока условие истинно:

```
var x = 0
do {
    println(x)
    x++
} while (x < 3)
```

Чтобы перейти к следующей итерации при определённом условии используется оператор **continue**, чтобы прервать цикл при определённом условии используется оператор **break**

```
fun main() {
    for(n in 1..10){
        if(n == 5) continue // при n = 5 код ниже не выполнится
        if(n == 7) break // при n = 7 цикл завершится
        println(n)
    }
}
```

Массивы

Для хранения набора значений в Kotlin, как и в других языках программирования, можно использовать массивы. При этом массив может хранить данные только одного того же типа. В Kotlin массивы представлены типом Array.

val numbers: Array<Int>

Задать массив можно следующим образом:

```
val numbers: Array<Int> = arrayOf(1, 2, 3, 4, 5)
```

С помощью [] можно обращаться к элементам массива и переопределять их.

```
val numbers: Array<Int> = arrayOf(1, 2, 3, 4, 5)
val n = numbers[1]  // получаем второй элемент  n=2
```

```
numbers[2] = 7       // переустанавливаем третий элемент
println("numbers[2] = ${numbers[2]}") // numbers[2] = 7
```

Для перебора массивов можно применять цикл for:

```
fun main() {

    val numbers = arrayOf(1, 2, 3, 4, 5)
    for(number in numbers){
        print("$number \t")
    }
}
```

Вывод:

```
1   2   3   4   5
```

Можно проверить наличие или отсутствие элементов в массиве с помощью операторов in и !in

```
val numbers: Array<Int> = arrayOf(1, 2, 3, 4, 5)

println(4 in numbers)       // true
println(2 !in numbers)      // false
```

Для упрощения создания массива в Kotlin определены дополнительные типы BooleanArray, ByteArray, ShortArray, IntArray, LongArray, CharArray, FloatArray и DoubleArray.

Также можно создавать двумерные массивы, например

```
val table = Array(3, { Array(3, {0}) })
table[0] = arrayOf(1, 2, 3)     // первая строка таблицы
table[1] = arrayOf(4, 5, 6)     // вторая строка таблицы
table[2] = arrayOf(7, 8, 9)     // третья строка таблицы
```

Осуществим перебор значений массива с помощью циклов

```
fun main() {

    val table: Array<Array<Int>> = Array(3, { Array(3, {0}) })
    table[0] = arrayOf(1, 2, 3)
    table[1] = arrayOf(4, 5, 6)
    table[2] = arrayOf(7, 8, 9)
    for(row in table){

        for(cell in row){
            print("$cell \t")
        }
        println()
    }
}
```

Вывод:

```
1   2   3
4   5   6
7   8   9
```

Функции

Структура функции

fun имя_функции(параметры): возвращаемый_тип{

выполняемые инструкции

}

При задании параметров функции нужно указывать типы параметров. Также возможно задание значений по умолчанию для параметров

```
fun displayUser(name: String, age: Int = 18, position: String="unemployed"){
    println("Name: $name   Age: $age  Position: $position")
}

fun main() {

    displayUser("Tom", 23, "Manager")
    displayUser("Alice", 21)
    displayUser("Kate")
}
```

Если необходимо, чтобы функция возвращала некоторое значение, используется **retutn**

```
fun sum(x:Int, y:Int): Int{

    return x + y
}
fun main() {

    val a = sum(4, 3)
    val b = sum(5, 6)
    val c = sum(6, 9)
    println("a=$a  b=$b  c=$c")
}
```

Заключение

Были рассмотрены основные конструкции языка Kotlin, необходимые для использования более продвинутых инструментов языка, включая ООП.

В чём преимущества Kotlin? Их много и вот некоторые из них:

1. Мультиплатформенность: Kotlin поддерживает мультиплатформенную разработку, что означает, что вы можете использовать Kotlin для создания приложений как на Android, так и на JVM, JavaScript и Native. Это позволяет сэкономить время и усилия разработчиков, так как можно использовать один язык для разработки под разные платформы.

2. Совместимость с Java: Kotlin полностью совместим с Java, что означает, что вы можете использовать библиотеки и инструменты Java в своем проекте на Kotlin.

3. Безопасность типов: Kotlin предлагает безопасную типизацию, что помогает избежать ошибок типов во время компиляции. Это уменьшает вероятность возникновения ошибок во время выполнения программы.

4. Краткость и выразительность: Kotlin предлагает более краткий и выразительный синтаксис по сравнению с Java. Это позволяет писать более читаемый и компактный код.

5. Активное сообщество и поддержка Google.

В целом, Kotlin - это мощный и перспективный язык программирования, который обладает рядом преимуществ и широким спектром применения. Его активное развитие и поддержка со стороны крупных компаний делают его одним из наиболее востребованных языков в современной разработке программного обеспечения.