KINGDOM OF SAUDI ARABIA
Ministry of Higher Education
Al-Imam Mohammad University
College of Computer & Information Sciences

المملكة العربية السعودية
وزارة التعليم العالي
جامعة الإمام محمد بن سعود الإسلامية
كلية علوم الحاسب والمعلومات

1441/1442
Fall 2019

Anass Markhouss	438021838
Amr Aljmal	438021843
Abdulrahman Almesher	439023898

Group NO: 01

Section: 171

Submission date
30th Nov 2019

Supervisor
D. Abdullah Albarrak

Table of Contents

1. HISTORICAL BACKGROUND

Kotlin is a programing language developed by company called JetBrains creators of PhpStorm, PyCharm etc. Named after Russian island, but they didn't make **Kotlin** to sell. They made it to solve their own development problems.
In July 2011 JetBrains unveiled Project **Kotlin** as an open source, a new language for the JVM, the first officially stable version **Kotlin v1.0** released on 15 February 2016. Works with different IDE like IntelliJ IDEA, TryKotlin, and Android Studio. In 2019 Google announced **Kotlin** as its main language for writing codes for Android apps.

2. Design Process

2.1 Goals of the language

Kotlin is designed to interoperate fully with Java, and the JVM version of its standard library depends on the Java Class Library, but type inference allows its syntax to be more concise.

Kotlin is currently predominantly used for Android app development, spurred on by Google's official support. Companies using Kotlin to stay competitive include Google, Trello/Atlassian, Pinterest, Kickstarter and Uber to name just a few.

2.2 Advantages

1- **it can be learned in a few hours** by simply reading the language reference.
2- **Complies with existing Java code,** Kotlin is positioned as a 100% Java-interoperable programming language. It is consistent with Java and all related tools and frameworks, which makes it possible to switch to Kotlin step by step.
3- **Kotlin comes from industry, not academia**. It solves problems faced by working programmers today.
4- **Kotlin is more concise**, Kotlin is way more concise than other programing language like java for instance, in many cases, solving the same problems with fewer lines of code. This improves code maintainability and readability.
5- **Kotlin code is safer,** Kotlin code is inherently safer than other programing language codes because it prevents common programming mistakes by design, resulting in fewer system failures and application crashes. For instant when using Java, certain error causes are more likely to occur again.

2.3 Disadvantages

1- **It is Still Different from Java**, even though Kotlin and Java have a number of similarities to share, they have some prominent differences as well.
2- **Slower Compilation Speed**, in a few cases, Kotlin works faster than Java, especially while performing incremental builds. But it should be kept in mind that Java would still reign supreme when it is about clean building.
3- **Fewer Kotlin Professionals to Recruit,** Despite the sky-high popularity of Kotlin, there are still only a handful of programmers available in this field today.

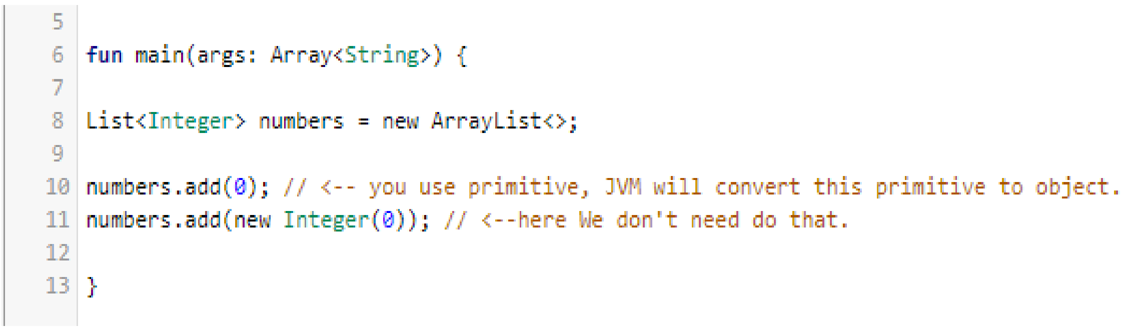
3. LANGUAGE OVERVIEW

3.1 Name of Variables

Variable names should use lower CamelCase instead of snake case, identifiers are case sensitive and may consist of letters, digits, and underscores, and cannot begin with a digit, whitespaces are not allowed also an identifier cannot contain symbols such as @, # etc.
In case of special words (Keywords, reserved words) **Kotlin** cannot use them as an identifier.

3.2 Data Types

Kotlin doesn't have primitive type (cannot declare primitive directly). It uses classes like Int, Float as an object wrapper for primitives. When the code is converted to JVM code, whenever possible, "primitive object" is converted to java primitive see Figure1. In some cases, this cannot be done. Those cases are, for example, collection of "primitives". For example, List<Int> cannot contains primitive. So, compiler knows when it can convert object to primitive.



```

fun main(args: Array<String>) {

List<Integer> numbers = new ArrayList<>;

numbers.add(0); // <-- you use primitive, JVM will convert this primitive to object.
numbers.add(new Integer(0)); // <--here We don't need do that.

}
```

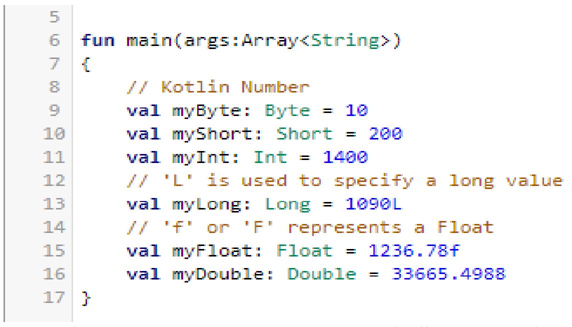
Figure 1 Example showing the primitive type

Numbers

Numeric types in Kotlin are similar to Java. They can be categorized into integer and floating-point types see Figure 2.

Integers

- Byte - 8 bit
- Short - 16 bit
- Int - 32 bit
- Long - 64 bit



```

fun main(args:Array<String>)
{
    // Kotlin Number
    val myByte: Byte = 10
    val myShort: Short = 200
    val myInt: Int = 1400
    // 'L' is used to specify a long value
    val myLong: Long = 1090L
    // 'f' or 'F' represents a Float
    val myFloat: Float = 1236.78f
    val myDouble: Double = 33665.4988
}
```

Figure 2 Example showing the declaration of different type of variables

Floating Point Numbers

- Float - 32 bit single-precision floating point value.
- Double - 64 bit double-precision floating point value.

Characters

Characters are represented using the type Char. Unlike Java, Char types cannot be treated as numbers. It means characters in **Kotlin** doesn't represent ASCII values.

Strings

Strings are represented using the String class. Elements of a string are characters that can be accessed by the indexing operation : s[i].

Arrays

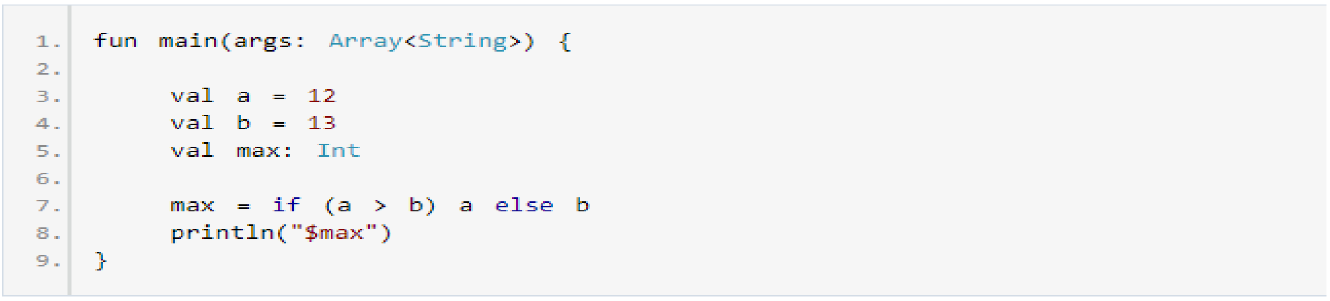
Arrays in **Kotlin** are represented using the Array class. You can create an array in **Kotlin** either using the library function arrayOf () or using the Array() constructor.

Variables in **Kotlin** have only two categories **var** and **val**
Declaring a variable type is dynamic using **var** keyword it's like a general variable and can be assigned multiple times and it is known as the mutable variable. And using **val** keyword it's like a constant variable and cannot be assigned multiple times and can be Initialized only single time and is known as the immutable variable in **Kotlin.**

3.3 Expressions and Assignment Statements

Expressions consist of variables, operators, etc. That evaluates to a single value for example, 90 + 20 is an expression that returns Int value. In **Kotlin**, if is an expression unlike Java (In Java, if is a statement) see Figure 3.

For example:



```
fun main(args: Array<String>) {

     val a = 12
     val b = 13
     val max: Int

     max = if (a > b) a else b
     println("$max")
}
```

Figure 3 Example showing the use of If statement as Expression and assign it to variable

Assignment statement is an expression that evaluates a value, which is assigned to the variable on the left side of the assignment operator. In Java, a statement always ends with a semicolon but, in **Kotlin** semicolon (;) is optional.

3.4 Control Statement

Kotlin's control statements which includes conditional expressions like if, if-else, when and looping statements like for, while, and do-while.

If, if-else

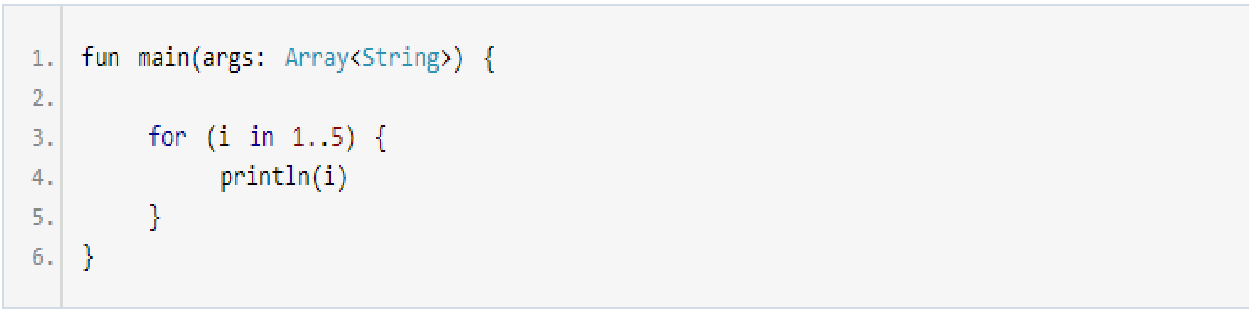
In **Kotlin**, you can use if as an expression instead of a statement. See figure1, you can assign the result of an if-else expression to a variable. Using if as an expression, required to have an else branch, otherwise, the compiler will throw an error. Unlike Java, Kotlin doesn't have a ternary operator because we can easily achieve what ternary operator does, using an if-else expression.

When

when expression is the replacement of switch statement from other languages like C, C++, and Java. It is concise and more powerful than switch statements. Just like if, when can be used as an expression and we can assign its result to a variable.

For, do-while, while

In **Kotlin** the loop statements are similar to the one in java and could be the same except the syntax of for loop is different from the one in Java see Figure 4, and similar to python which iterates through anything that provides an iterator, also the break and continue keywords used in **Kotlin**.



```kotlin
fun main(args: Array<String>) {

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

Figure 4 Example showing one of the for-loop shapes

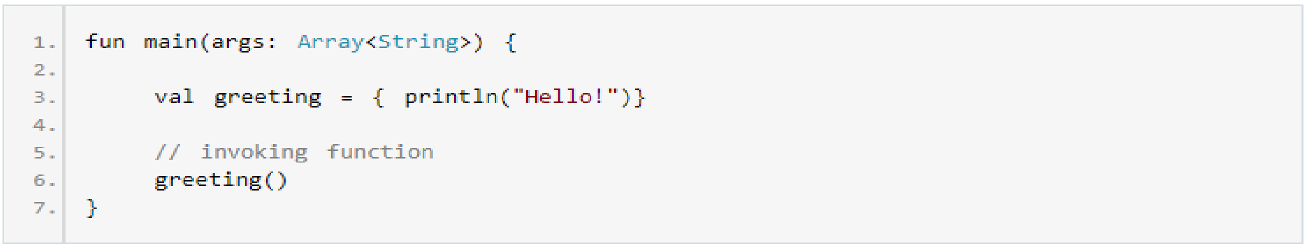
3.5 Subprograms

In **Kotlin** subprograms (also called functions) are defined using **fun** keyword with optional parameters and a return value, parameters are defined using Pascal notation, i.e. *name*: *type*, and separated using commas. Each parameter must be explicitly typed. Functions must always specify return types explicitly, unless it's intended for them to return Unit.

Example of Function return Type

- ()->Unit —the function type that returns nothing useful (Unit) and takes no arguments.
- (Int)->Int— the function type that returns Int and takes single argument of type Int.
- ()-> ()->Unit— the function type that returns another function that returns nothing useful (Unit). Both functions take no arguments.

Lambda expression is an anonymous function; a function without name see Figure 5. Lambdas are used to minimize the code and also to pass the block of code as parameter.



```
fun main(args: Array<String>) {

    val greeting = { println("Hello!")}

    // invoking function
    greeting()
}
```

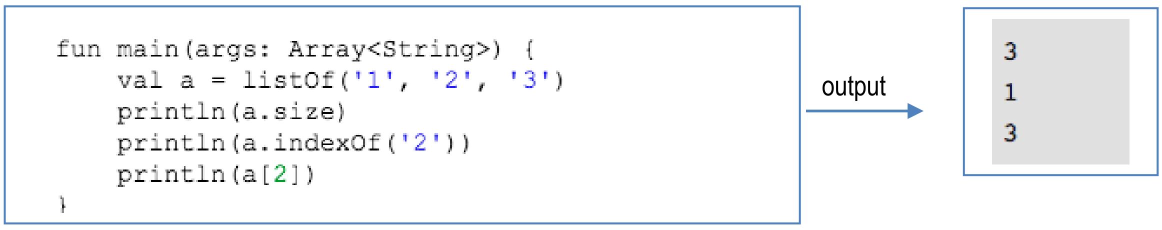
Figure 5 Example showing Lambda function

3.6 Abstract data types (ADT)

Kotlin supports various abstract data types (ADT) such as:

Lists: ListOF()

Kotlin has two types of lists, immutable lists (cannot be modified) and mutable lists (can be modified). Read-only lists are created with listOf() see Figure 6, whose elements cannot be modified and mutable lists created with mutableListOf() method where we alter or modify the elements of the list.



```
fun main(args: Array<String>) {
    val a = listOf('1', '2', '3')
    println(a.size)
    println(a.indexOf('2'))
    println(a[2])
}
```

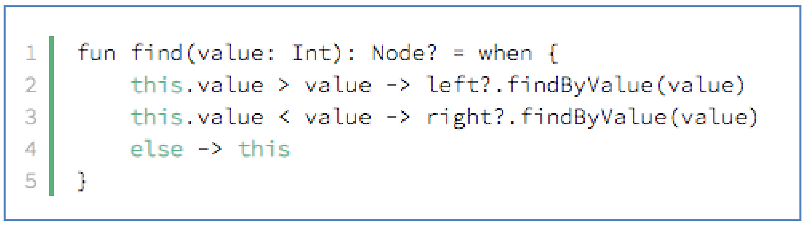
```
3
1
3
```

Figure 6 Example showing the list data type

Tree

Tree is a widely used abstract data type (ADT)—or data structure implementing this ADT—that simulates a hierarchical tree structure, with a root value and subtrees of children with a parent node, represented as a set of linked nodes. **Kotlin** supports a dissent amount of tree types like: Binary Tree, Binary Search Tree, N-ary tree, Heap Structure, Huffman Tree, etc.

Example for looking up (searching):



```
fun find(value: Int): Node? = when {
    this.value > value -> left?.findByValue(value)
    this.value < value -> right?.findByValue(value)
    else -> this
}
```

Queue

Queue is a special case in **Kotlin** which Queue is an interface. So, you can't instantiate an interface, you have to implement it or instantiate a class that implement it.
For example, you can do (var queue: Queue<Int> = ArrayDeque<Int> ()).
ArrayDeque implements Queue.

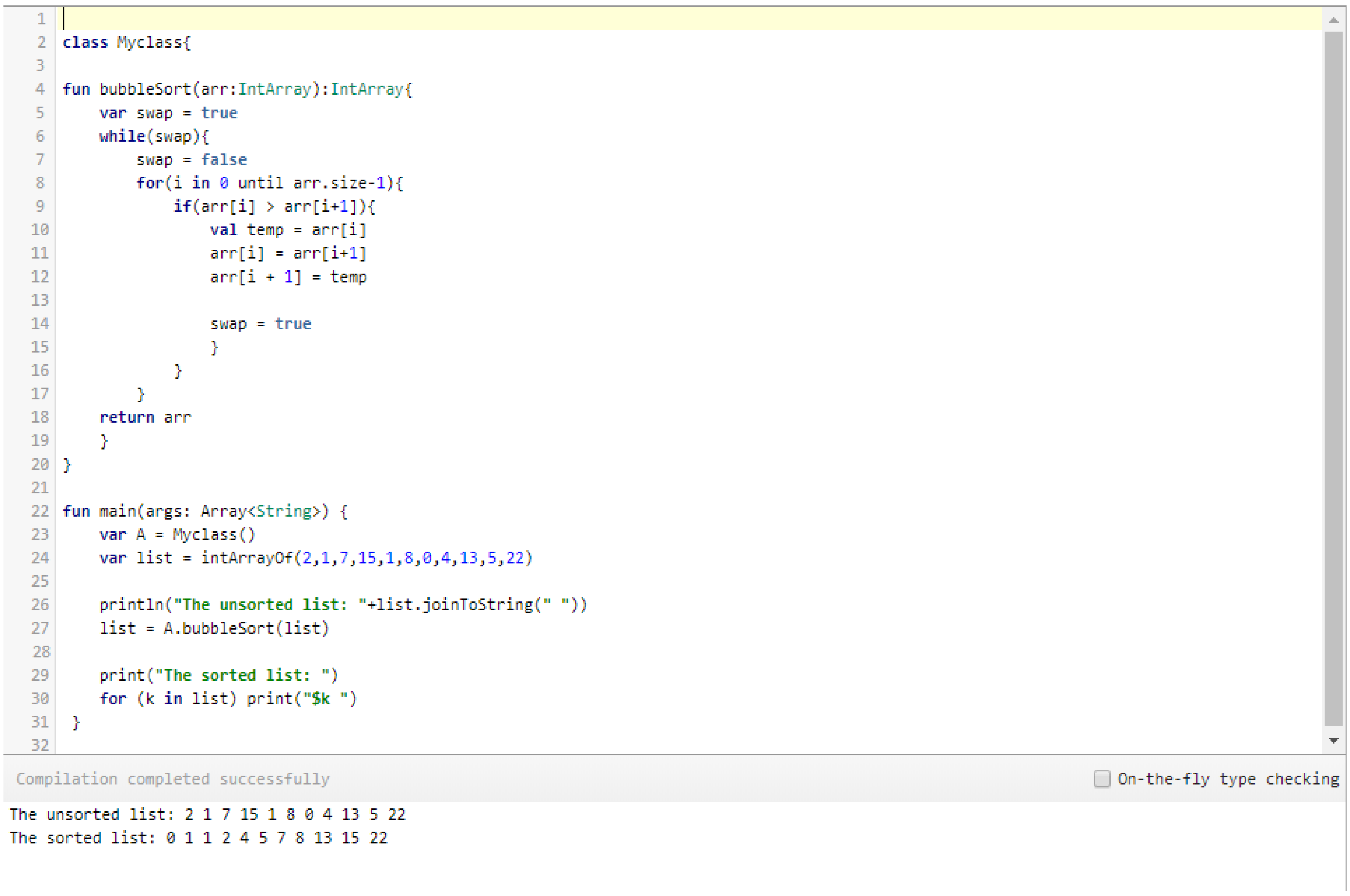
3.7 Support for Object-Oriented Programming

Basically, it supports object-oriented and functional programing. However, this should be obvious from the fact that is simplifies Java coding by removing or dealing with redundancies presents in Java coding syntax. Java in both a functional and object-oriented language.

3.8 Exception Handling and Event Handling

Kotlin has only one type of the exceptions which is unchecked exception and it can caught only at the run time using **Try** and **Catch** keywords. All the exception classes are descendants of throwable class. The event handler causes confusion especially in Kotlin.

3.9 Program in Kotlin



```

class Myclass{

fun bubbleSort(arr:IntArray):IntArray{
    var swap = true
    while(swap){
        swap = false
        for(i in 0 until arr.size-1){
            if(arr[i] > arr[i+1]){
                val temp = arr[i]
                arr[i] = arr[i+1]
                arr[i + 1] = temp

                swap = true
                }
            }
        }
    return arr
    }
}

fun main(args: Array<String>) {
    var A = Myclass()
    var list = intArrayOf(2,1,7,15,1,8,0,4,13,5,22)

    println("The unsorted list: "+list.joinToString(" "))
    list = A.bubbleSort(list)

    print("The sorted list: ")
    for (k in list) print("$k ")
 }

```

```
Compilation completed successfully                                On-the-fly type checking
The unsorted list: 2 1 7 15 1 8 0 4 13 5 22
The sorted list: 0 1 1 2 4 5 7 8 13 15 22
```

4. EVALUATION OF THE LANGUAGE

4.1 Readability

The ability to read and understand the code make the language more desired, **Kotlin** outperformed java in aspect of readability and by foxing on application development the number of unnecessary functions and features is reduce in the other hand the number of features and functions related to application development is increased.

4.1.1 Overall Simplicity

When comparing with Java, with all those useful attributes, **Kotlin** is known for its simplicity and precision. The similarity to java leads to common features in the context of modern object-oriented programming however, it offers many new syntactic constructs.

4.1.2 Orthogonality

Orthogonality is one of the most important properties that can help make even complex designs compact. In **Kotlin** feature of a program can be used freely without having effect to other features with consist ways for binding structure

4.1.3 Control Statement

The syntax of the Control Statement in **Kotlin** helps the programmers to understand what they read since the Kotlin used almost the same syntax in other object-oriented programming languages, but with extra features and constraint.

4.1.4 Data Types and Structures

Kotlin provide data types and structures like the other object-oriented programing languages, with simple constraint and flexible way to deal with them in.

4.1.5 Syntax Considerations

Kotlin retained most of the syntax of java which was helpful for java programmers trying to learn this new language with simple and familiar object-oriented structure, **Kotlin** syntax is concise and it offers a bunch of features which can shrink down codes to a great extent.

4.2 Writability

4.2.1 Simplicity and Orthogonality

Kotlin has a many constructs and structures with different techniques to combine them also the ability to translate the **Java code** into **Kotlin code** automatically by the compiler which are great feature.

4.2.2 Support for Abstraction

Abstraction is one of the core concepts of Objected Oriented Programming. When there is a knowledge of what functionalities a class should have, but not aware of how the functionality is implemented or if the functionality could be implemented in several ways. Abstraction in **Kotlin** could be achieved by following ways:

1- Kotlin Interfaces
2- Kotlin Abstract Classes

4.2.3 Expressivity

Kotlin's creative language features, such as its support for type-safe builders and delegated properties, help build powerful and easy-to-use abstractions in addition to several ways to define and manipulate structures like for loop , functions, which provide simple and varying ways to represent and express codes.

4.3 Reliability

4.3.1 Type checking

Kotlin uses **(is**) and (**! is**) keywords to check whether a property is and is not of a certain type respectively. Also, there is some other type checking properties such as: smart type casting and smart casts in class. Kotlin is powerful enough to determine the types at compile time itself, this gives an aid to the reliability which make it strongly reliable.

4.3.2 Exception handling

All exception classes in **Kotlin** are descendants of the class Throwable. Every exception has a message, stack trace and an optional cause. Kotlin does not have checked exceptions. There are many reasons for this. 'try' and 'catch' can also be used in order to handle the exception manually same as java. This gives a huge aid to reliability making Kotlin more reliable.

4.3.3 Aliasing

As same as java having two or more distinct names that can be used to access the same memory cell. Whish affect reliability negatively. Not to mix with aliasing type in **Kotlin** which provide alternative names for existing types. If the type name is too long you can introduce a different shorter name and use the new one instead. So, they are not the same thing.

4.3.4 Readability and Writability

Since **Kotlin** supports nature ways of expression. The codes are easy to read and to be understood also, codes can be written easily since they are smiler to java language. So, for that they give an aid to reliability.

5.COST OF THE LANGUAGE

Kotlin is free programing language but documentation, books, training, classes, videos, implementations (actual compilers, associated development tools, and IDEs) change the range from free to expensive. The cost is often dependent on the licensing terms **Kotlin** is an open source language developed under the Apache 2.0 license that allow to distribute, modify, or use the software for any purpose. For example, JetBrains provide two versions of IntelliJ IDEA for Kotlin, **Community** (Free edition) and **Ultimate** with different ability and features.