Assignment 1

Mobile Programming

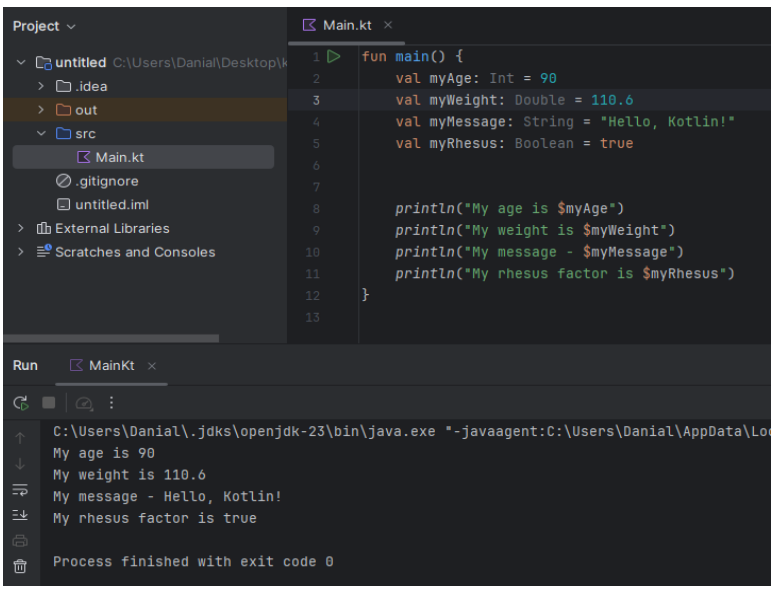
Student: Danial Serekov

Exercise 1: Kotlin Syntax Basics

1. Variables and Data Types:

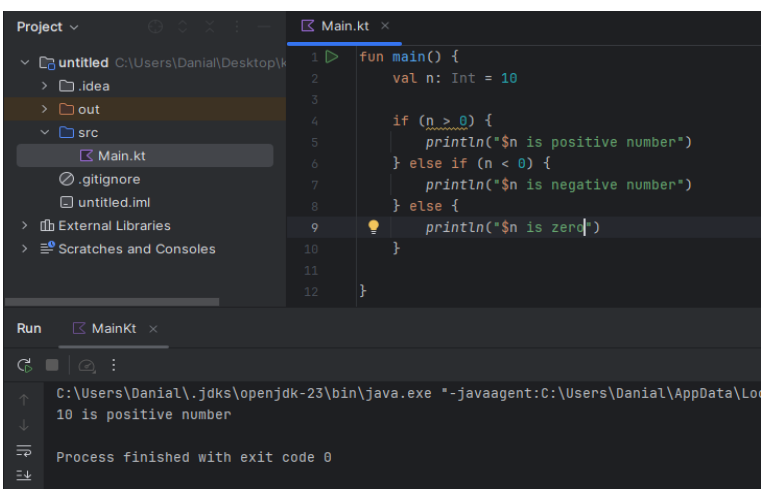
- Create variables of different data types: Int, Double, String, Boolean.

- Print the variables using println.



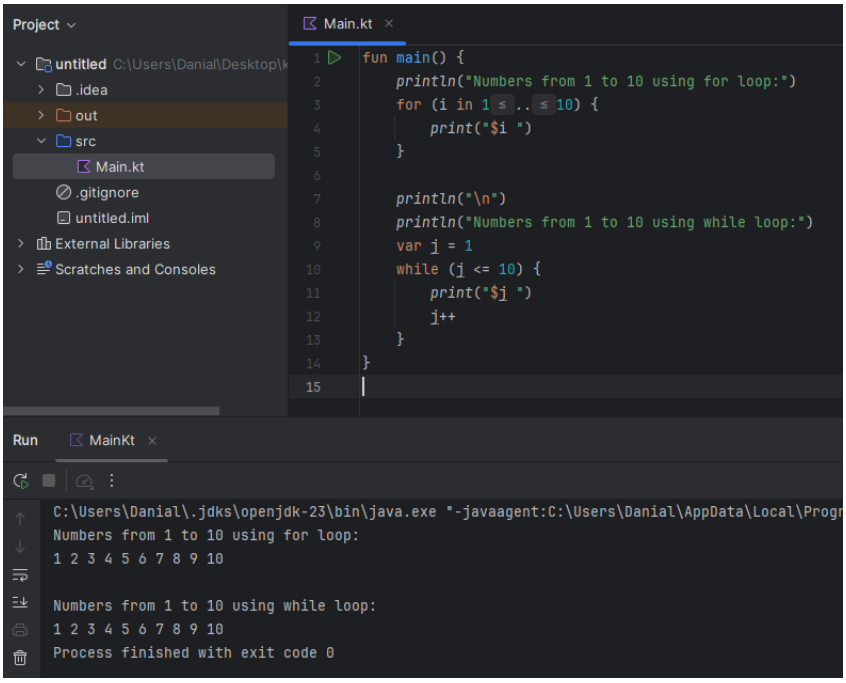
Conditional Statements:

- Create a simple program that checks if a number is positive, negative, or zero.



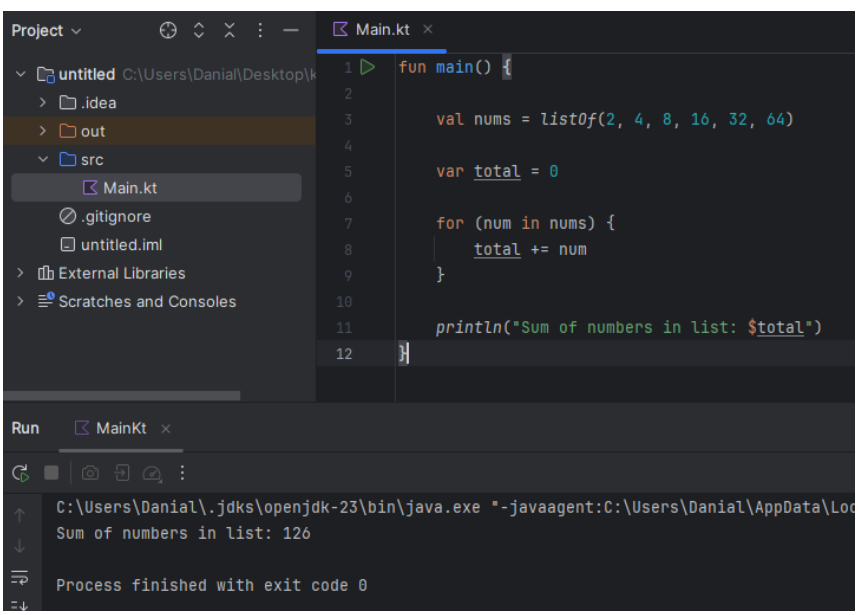
Loops:

- Write a program that prints numbers from 1 to 10 using for and while loops.



Collections:

- Create a list of numbers, iterate through the list, and print the sum of all numbers.



Exercise 2: Kotlin OOP (Object-Oriented Programming)

1. Create a Person class:

- Define properties for name, age, and email.

- Create a method to display the person's details.

```kotlin
class Person(val name: String, val age: Int, val email: String){

    fun displayDatails(){
        println("Name: $name, \nAge: $age, \nEmail: $email")
    }
}

fun main() {

    val person = Person( name: "Danial",  age: 21,  email: "danialserekov@gmail.com")

    person.displayDatails()
}
```

```
C:\Users\Danial\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Users\Danial\AppData\Local\Programs\Inte
Name: Danial,
Age: 21,
Email: danialserekov@gmail.com

Process finished with exit code 0
```

Inheritance:

- Create a class Employee that inherits from the Person class.

- Add a property for salary.

- Override the displayInfo method to include the salary.

```kotlin
open class Person(val name: String, val age: Int, val email: String){

    open fun displayDatails(){
        println("Name: $name, \nAge: $age, \nEmail: $email")
    }
}

class Employee(name: String, age: Int, email: String, val salary: Double): Person(name,age,email){

    override fun displayDatails(){
        super.displayDatails()
        println("Salary: $salary KZT")
    }
}

fun main() {

    val employee = Employee( name: "Danial",  age: 21,  email: "danialserekov@gmail.com",  salary: 999000.0)

    employee.displayDatails()
}
```

```
C:\Users\Danial\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Users\Danial\AppData\Local\Programs\IntelliJ IDEA Ultimate\lib\
Name: Danial,
Age: 21,
Email: danialserekov@gmail.com
Salary: 999000.0 KZT

Process finished with exit code 0
```

Encapsulation:

- Create a BankAccount class with a private property balance.

- Provide methods to deposit and withdraw money, ensuring the balance never goes negative.

```
1  open class Person(val name: String, val age: Int, val email: String){

3      open fun displayDatails(){
4          println("Name: $name \nAge: $age \nEmail: $email")
5      }
6  }
7
8  class Employee(name: String, age: Int, email: String, val salary: Double): Person(name,age,email){
9
10     override fun displayDatails(){
11         super.displayDatails()
12         println("Salary: $salary KZT \n")
13     }
14 }
15
16 class BankAccount(private var balance: Double){
17
18     fun deposit(money: Double){
19         if (money > 0) {
20             balance += money
21             println("Deposited $money KZT")
22         } else {
23             println("Your new deposit must be greater than 0 KZT")
24         }
25         println("Balance: $balance KZT \n")
26     }
27
28     fun withdraw(money: Double){
29         if (money > 0 && balance >= money) {
30             balance -= money
31             println("Withdrawn $money KZT")
32         } else if (money > balance && balance < 0) {
33             println("Not enough money on your deposit")
34         } else {
35             println("Your deposit withdrawal must be greater than 0 KZT")
36         }
37         println("Balance: $balance KZT \n")
38     }
39
40 }
```

```
42 fun main() {
43
44     val employee = Employee( name: "Danial Serekov",  age: 21,  email: "danialserekov@gmail.com",  salary: 999000.0)
45     employee.displayDatails()
46
47     val wallet = BankAccount( balance: 1000.0)
48     wallet.deposit( money: 5000.0)
49     wallet.deposit( money: -100.0)
50     wallet.withdraw( money: 15000.0)
51     wallet.withdraw( money: 2000.0)
52     wallet.withdraw( money: -2000.0)
53 }
54
```

Inheritance: class **Employee** inherits class **Person** on line 8 under ": Person(...)".

Polymorphism: method **displayDetails** is overridden in class **Employee** with **override**.

Encapsulation: balance property in **BankAccount** is private, accessed via methods.

Abstraction: Details of implementation methods such as **deposit** and **withdraw** are hidden.

```
Run    MainKt
C:\Users\Danial\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Users\Danial\AppDa
Name: Danial Serekov
Age: 21
Email: danialserekov@gmail.com
Salary: 999000.0 KZT

Deposited 5000.0 KZT
Balance: 6000.0 KZT

Your new deposit must be greater than 0 KZT
Balance: 6000.0 KZT

Your deposit withdrawal must be greater than 0 KZT
Balance: 6000.0 KZT

Withdrawn 2000.0 KZT
Balance: 4000.0 KZT

Your deposit withdrawal must be greater than 0 KZT
Balance: 4000.0 KZT


Process finished with exit code 0
```

Exercise 3: Kotlin Functions

1. Basic Function:

- Write a function that takes two integers as arguments and returns their sum.

```
Project
untitled C:\Users\Danial\Desk
  .idea
  out
  src
    Main.kt
  .gitignore
  untitled.iml
External Libraries
Scratches and Consoles
```

```kotlin
fun total(a: Int, b: Int): Int {
    return a + b
}

fun main() {
    val result = total( a: 829,  b: 653)
    println("The result is $result")
}
```

```
Run    MainKt
C:\Users\Danial\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Users\Danial\
The result is 1482

Process finished with exit code 0
```

Lambda Functions:

- Create a lambda function that multiplies two numbers and returns the result.

```
fun main() {
    val multiply: (Int, Int) -> Int = { a, b -> a * b }

    val result = multiply(32, 57)
    println("The result is $result")
}
```

```
C:\Users\Danial\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Users\Danial\AppData\Local\P
The result is 1824

Process finished with exit code 0
```

Higher-Order Functions:

- Write a function that takes a lambda function as a parameter and applies it to two integers.

```
fun highordfun(a: Int, b: Int, proc : (Int, Int) -> Int): Int {
    return proc(a, b)
}

fun main() {
    val addResult = highordfun( a: 95,  b: 28) { a, b -> a + b }
    val subtractionResult = highordfun( a: 64,  b: 39) { a, b -> a - b }
    val multiplyResult = highordfun( a: 74,  b: 238) { a, b -> a * b }
    val divisionResult = highordfun( a: 56,  b: 4) { a, b -> a / b }

    println("The result of addition is $addResult " +
            "\nThe result of subtraction is $subtractionResult " +
            "\nThe result of multiplication is $multiplyResult " +
            "\nThe result of division is $divisionResult")
}
```

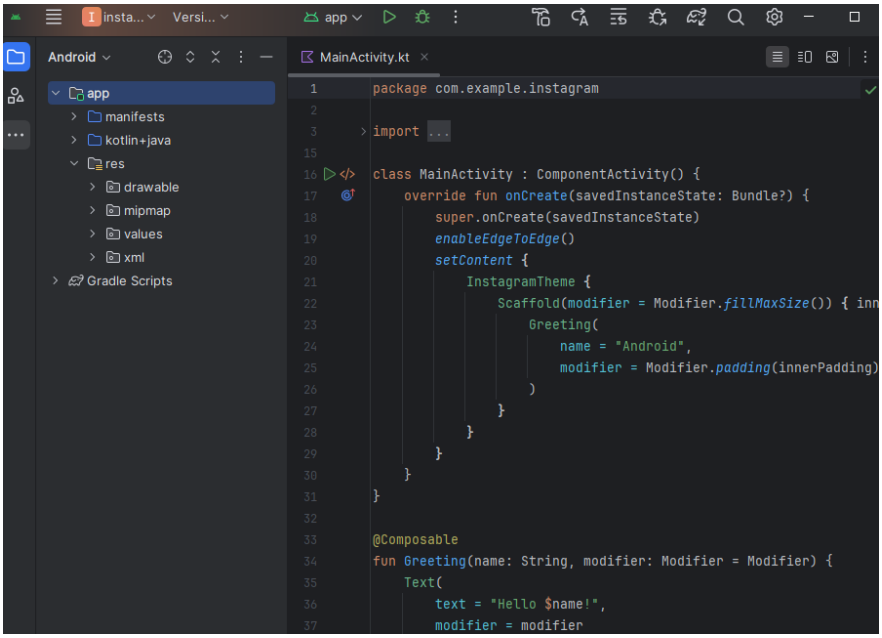
```
C:\Users\Danial\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Users\Danial\AppData\Local\Pr
The result of addition is 123
The result of subtraction is 25
The result of multiplication is 17612
The result of division is 14

Process finished with exit code 0
```

Exercise 4: Android Layout in Kotlin (Instagram-like Layout)

1. Set Up the Android Project:

- Create a new Android project in Android Studio.

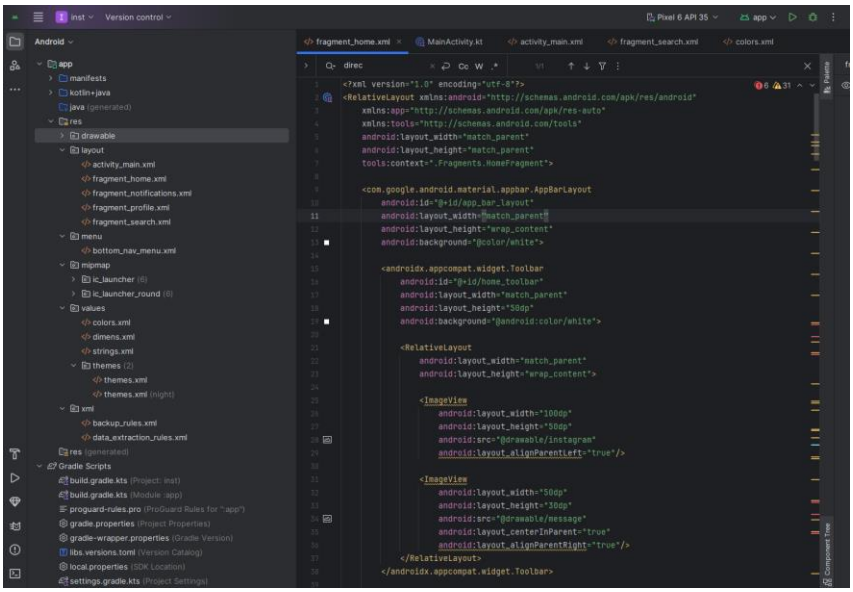
- Ensure you have a Kotlin-based project.



2. Design the Layout:

- Create a new XML layout file (**activity_main.xml**) for a simple Instagram-like user interface.

- Include elements like **ImageView**, **TextView**, and **RecyclerView** for the feed.



Create the RecyclerView Adapter:

- Set up the RecyclerView to display a feed of posts with **ImageView** for the picture and **TextView** for the caption.

MainActivity Setup:

- Initialize the **RecyclerView** in **MainActivity** and populate it with sample data.