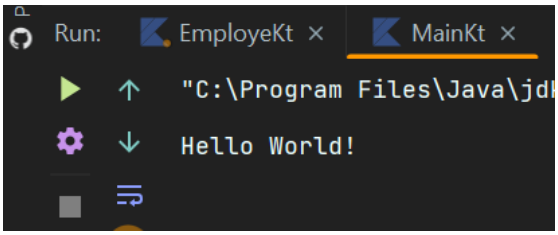
Sneha Walikar(Assignment1_Day1)

Task-1

Install Kotlin and configure IntelliJ IDEA. Verify the setup by running a "Hello, World!" program.

```
fun main() {
        println("Hello World!")
}
```

Output :

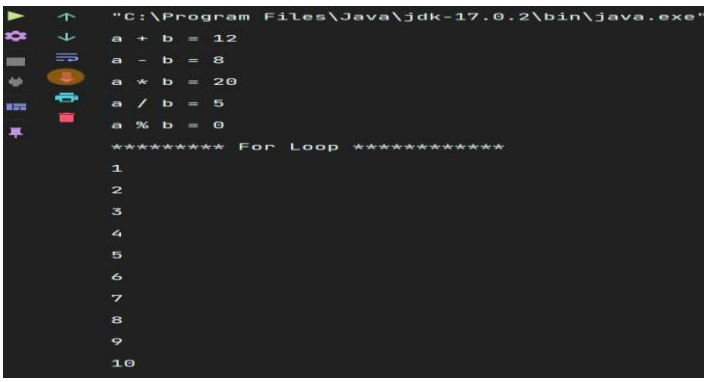


Task-2

Explore Kotlin REPL (Read-Eval-Print Loop) to familiarize with Kotlin syntax and basic operations.

```
fun main(){
    var a = 10
    var b = 2
    println("a + b = " + (a + b))
    println("a - b = " + (a - b))
    println("a * b = " + (a.times(b)))
    println("a / b = " + (a / b))
    println("a % b = " + (a.rem(b)))

    println("********* For Loop ************")

    for (i in 1..10){
        println(i)
    }
}
```

Output:

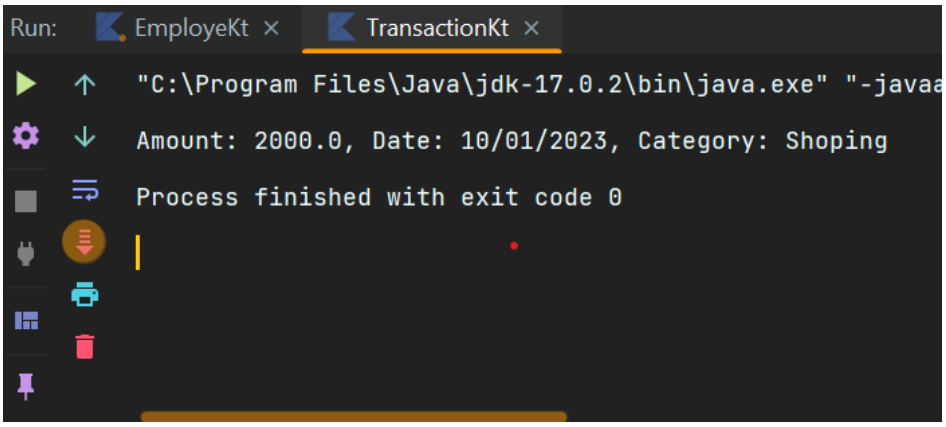


Task-3

Create a Transaction class with properties such as amount, date, and category.

```
class Transaction {
    var amount: Double = 0.0;
    var date: String = "";
    var category: String = "";

    fun info(): String{
        return "Amount: $amount, Date: $date, Category: $category"
    }
}

fun main(){
    var tran1 = Transaction()

    tran1.amount = 2000.0
    tran1.date = "10/01/2023"
    tran1.category = "Shoping"

    print(tran1.info())
}
```

Output:



Task-4

Implement control structures to categorize transactions (e.g., Food, Utilities, Entertainment) using when statements.

```
import java.time.LocalDate

// Define the Transaction class
class Task4(
    val amount: Double,
    val date: LocalDate,
    val description: String,
    var category: String = "Uncategorized" // Default value
) {
    // Function to categorize transactions
    fun categorizeTransaction() {
        category = when {
            description.contains("grocery", ignoreCase = true) -> "Food"
            description.contains("restaurant", ignoreCase = true) -> "Food"
```

```kotlin
            description.contains("cafe", ignoreCase = true) -> "Food"
            description.contains("utility", ignoreCase = true) -> "Utilities"
            description.contains("electricity", ignoreCase = true) -> "Utilities"
            description.contains("water bill", ignoreCase = true) -> "Utilities"
            description.contains("movie", ignoreCase = true) -> "Entertainment"
            description.contains("concert", ignoreCase = true) -> "Entertainment"
            description.contains("theater", ignoreCase = true) -> "Entertainment"
            else -> "Miscellaneous"
        }
    }

    // Override toString method for easy printing
    override fun toString(): String {
        return "Transaction(amount=$amount, date=$date, description='$description',
category='$category')"
    }
}

fun main() {
    // Create some transactions
    val transactions = listOf(
        Task4(50.0, LocalDate.of(2024, 5, 14), "grocery shopping"),
        Task4(100.0, LocalDate.of(2024, 5, 12), "electricity bill"),
        Task4(30.0, LocalDate.of(2024, 5, 13), "movie night"),
        Task4(20.0, LocalDate.of(2024, 5, 15), "restaurant lunch"),
        Task4(15.0, LocalDate.of(2024, 5, 16), "cafe coffee"),
        Task4(45.0, LocalDate.of(2024, 5, 17), "concert ticket"),
        Task4(60.0, LocalDate.of(2024, 5, 18), "water bill")
    )

    // Categorize each transaction
    transactions.forEach {
        it.categorizeTransaction()
        println(it)
    }
}
```

Output:

```
demo > src > main > kotlin > Task4.kt > f main
Transaction.kt × Task4.kt × car.kt × input.kt × Main.kt × Task2.kt ×
Run: EmployeKt × Task4Kt ×
"C:\Program Files\Java\jdk-17.0.2\bin\java.exe" "-javaagent:C:\Users\sohil\AppData\Local\JetBrains\I
Transaction(amount=50.0, date=2024-05-14, description='grocery shopping', category='Food')
Transaction(amount=100.0, date=2024-05-12, description='electricity bill', category='Utilities')
Transaction(amount=30.0, date=2024-05-13, description='movie night', category='Entertainment')
Transaction(amount=20.0, date=2024-05-15, description='restaurant lunch', category='Food')
Transaction(amount=15.0, date=2024-05-16, description='cafe coffee', category='Food')
Transaction(amount=45.0, date=2024-05-17, description='concert ticket', category='Entertainment')
Transaction(amount=60.0, date=2024-05-18, description='water bill', category='Utilities')

Process finished with exit code 0
```

Assignment-Employee Management System

```
import java.util.*

data class Employe(
    var id: Int,
    var name: String,
    var position: String,
    var salary: Double
)

class EmployeeManagementSystem {
    private val employees = mutableListOf<Employe>()
    private var nextId = 1

    fun manageEmployees() {
        val scanner = Scanner(System.`in`)
        while (true) {
            println("Choose an operation: add, delete, update, list, or exit")
            when (scanner.nextLine()) {
                "add" -> addEmployee(scanner)
                "delete" -> deleteEmployee(scanner)
                "update" -> updateEmployee(scanner)
                "list" -> listEmployees()
                "exit" -> return
                else -> println("Invalid operation. Please choose again.")
            }
        }
    }

    private fun addEmployee(scanner: Scanner) {
        println("Enter employee name:")
        val name = scanner.nextLine()
        println("Enter employee position:")
        val position = scanner.nextLine()
        println("Enter employee salary:")
        val salary = scanner.nextLine().toDouble()

        val employee = Employe(nextId++, name, position, salary)
        employees.add(employee)
        println("Employee added: $employee")
    }

    private fun deleteEmployee(scanner: Scanner) {
        println("Enter employee ID to delete:")
        val id = scanner.nextLine().toInt()
        val employee = employees.find { it.id == id }
        if (employee != null) {
            employees.remove(employee)
            println("Employee deleted: $employee")
        } else {
            println("Employee not found.")
        }
    }

    private fun updateEmployee(scanner: Scanner) {
        println("Enter employee ID to update:")
        val id = scanner.nextLine().toInt()
        val employee = employees.find { it.id == id }
        if (employee != null) {
            println("Enter new name (leave blank to keep current name):")
            val name = scanner.nextLine()
            if (name.isNotBlank()) employee.name = name

            println("Enter new position (leave blank to keep current position):")
```

```
            val position = scanner.nextLine()
            if (position.isNotBlank()) employee.position = position

            println("Enter new salary (leave blank to keep current salary):")
            val salaryInput = scanner.nextLine()
            if (salaryInput.isNotBlank()) employee.salary = salaryInput.toDouble()

            println("Employee updated: $employee")
        } else {
            println("Employee not found.")
        }
    }

    private fun listEmployees() {
        if (employees.isEmpty()) {
            println("No employees found.")
        } else {
            employees.forEach { println(it) }
        }
    }
}

fun main() {
    val employeeManagementSystem = EmployeeManagementSystem()
    employeeManagementSystem.manageEmployees()
}
```

Output : -

```
Run:  EmployeKt ×
"C:\Program Files\Java\jdk-17.0.2\bin\java.exe" "-javaagent:C:\Users\sohil\A
Choose an operation: add, delete, update, list, or exit
add
Enter employee name:
Sohil
Enter employee position:
Intern
Enter employee salary:
20000
Employee added: Employe(id=1, name=Sohil , position=Intern, salary=20000.0)
Choose an operation: add, delete, update, list, or exit
```