Day-2, Assignment-1

Task 5: Write functions to add, delete, and edit transactions in a TransactionList class.

import java.time.LocalDate

data class Transaction(

val id: Int,

var amount: Double,

var date: LocalDate,

var category: String

)

class TransactionList {

private val transactions = mutableListOf<Transaction>()

private var nextId = 1

// Add a new transaction to the list

fun addTransaction(amount: Double, date: LocalDate, category: String) {

transactions.add(Transaction(nextId++, amount, date, category))

}

// Delete a transaction by its ID

fun deleteTransaction(id: Int) {

val transaction = transactions.find { it.id == id }

if (transaction != null) {

transactions.remove(transaction)

println("Transaction with ID $id deleted.")

} else {

println("Transaction with ID $id not found.")

}

}

// Edit a transaction by its ID

fun editTransaction(id: Int, newAmount: Double, newDate: LocalDate, newCategory: String) {

val transaction = transactions.find { it.id == id }

if (transaction != null) {

transaction.amount = newAmount

transaction.date = newDate

transaction.category = newCategory

println("Transaction with ID $id edited.")

} else {

println("Transaction with ID $id not found.")

}

}

// Display all transactions in the list

fun displayTransactions() {

println("Transaction List:")

for (transaction in transactions) {

println("${transaction.id} - ${transaction.amount} - ${transaction.date} - ${transaction.category}")

}

}

}

fun main() {

val transactionList = TransactionList()

// Adding transactions

transactionList.addTransaction(100.0, LocalDate.of(2024, 5, 15), "Food")

transactionList.addTransaction(50.0, LocalDate.of(2024, 5, 16), "Utilities")

// Displaying transactions

transactionList.displayTransactions()

// Editing a transaction

transactionList.editTransaction(1, 150.0, LocalDate.of(2024, 5, 20), "Entertainment")

transactionList.displayTransactions()

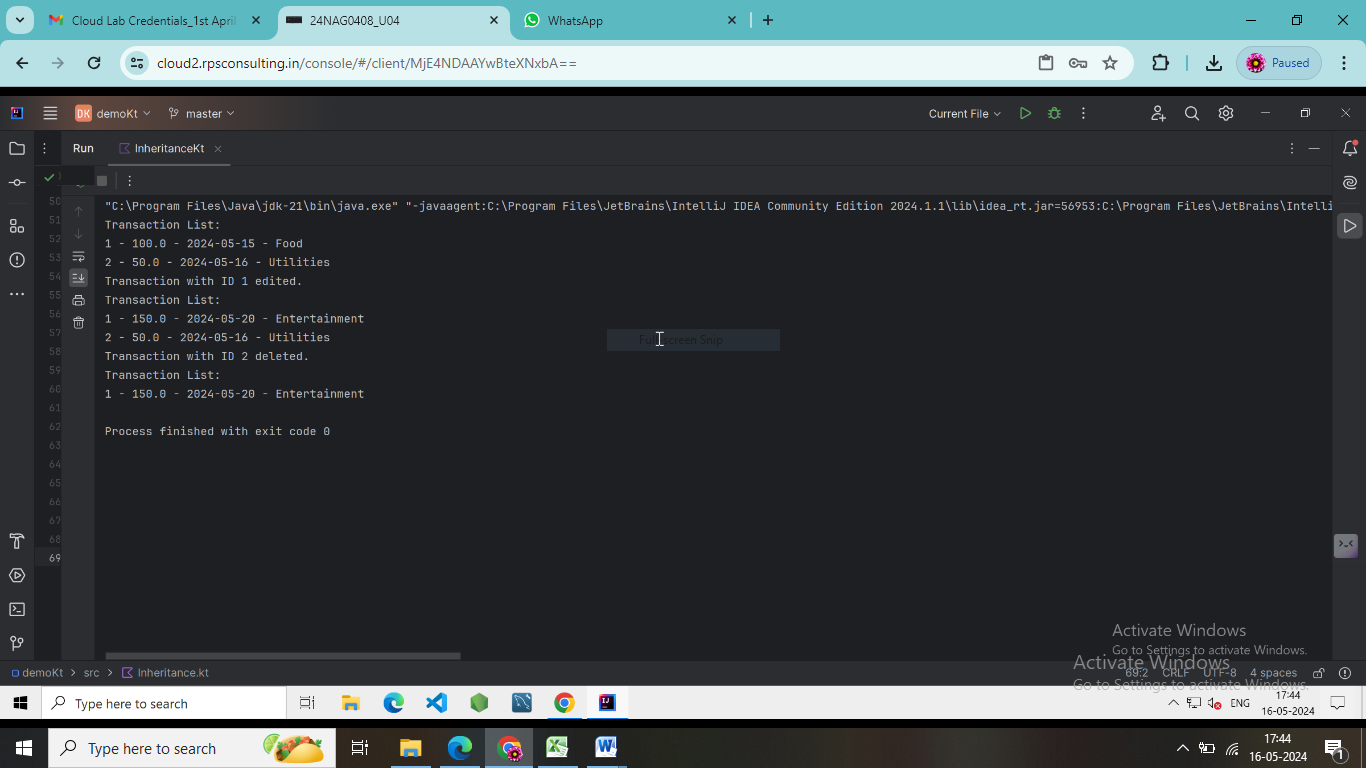
// Deleting a transaction

transactionList.deleteTransaction(2)

transactionList.displayTransactions()

}

## Output:



Task 6: Develop a simple User class with methods to login and display a summary of expenses.

class User(private val username: String, private val password: String) {

private var isLoggedIn: Boolean = false

// Dummy expenses data

private val expenses: Map<String, Double> = mapOf(

"Groceries" to 100.0,

"Utilities" to 50.0,

"Entertainment" to 80.0,

)

fun login(username: String, password: String): Boolean {

if (this.username == username && this.password == password) {

isLoggedIn = true

return true

}

return false

}

fun displayExpenseSummary() {

if (!isLoggedIn) {

println("Please login first to view expenses.")

return

}

println("Expense Summary:")

expenses.forEach { (category, amount) ->

println("$category: $${"%.2f".format(amount)}")

}

}

}

fun main() {

val user = User("username", "password")

// Logging in

if (user.login("username", "password")) {

println("Login successful!")

// Displaying expense summary

user.displayExpenseSummary()

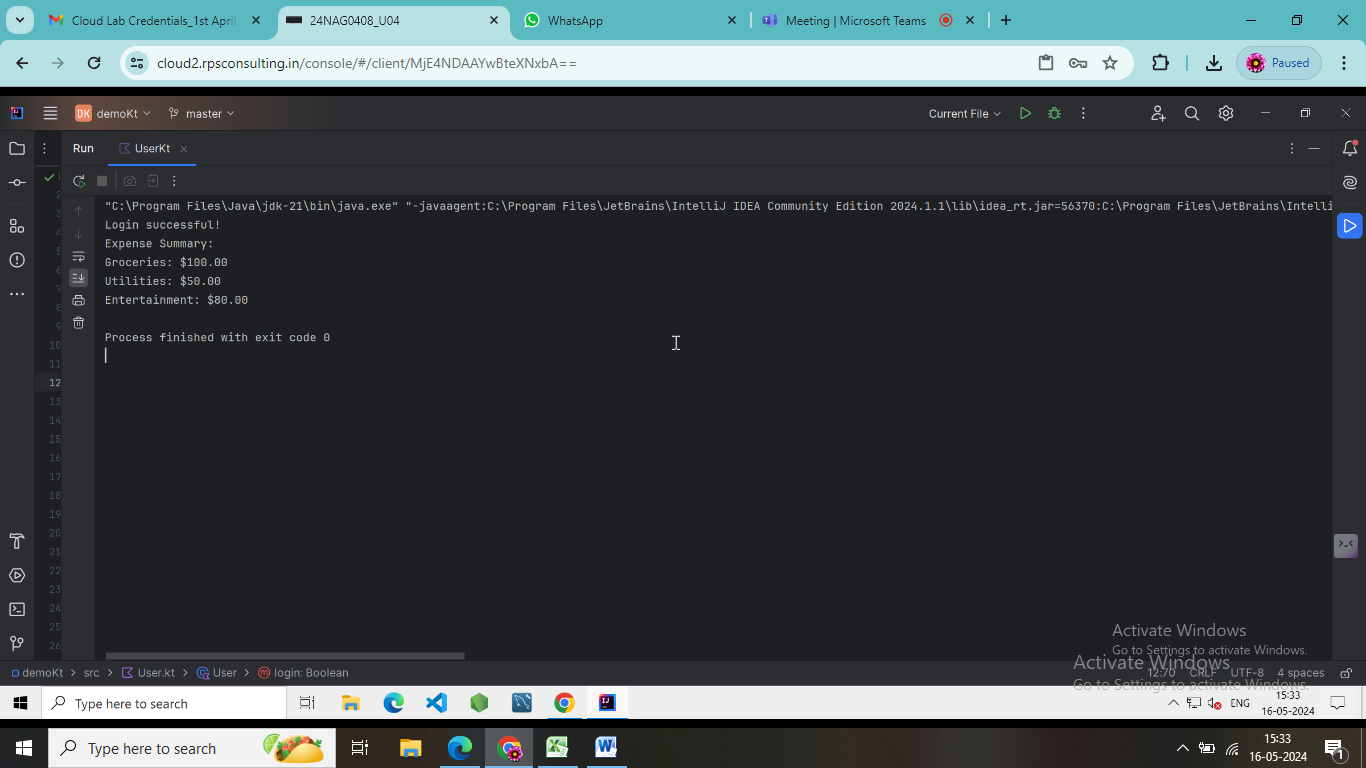
} else {

println("Invalid username or password.")

}

}

## Output:



Task 7: Use lambdas and higher-order functions to filter and sort transactions by date or amount.

data class Transaction(val date: String, val amount: Double)

fun main() {

val transactions = listOf(

Transaction("2024-05-10", 100.0),

Transaction("2024-05-12", 75.0),

Transaction("2024-05-15", 150.0),

Transaction("2024-05-08", 50.0),

Transaction("2024-05-20", 200.0)

)

println("Original Transactions:")

transactions.forEach { println(it) }

val sortedByDate = sortTransactions(transactions) { it.date }

println("\nTransactions Sorted by Date:")

sortedByDate.forEach { println(it) }

val sortedByAmount = sortTransactions(transactions) { it.amount }

println("\nTransactions Sorted by Amount:")

sortedByAmount.forEach { println(it) }

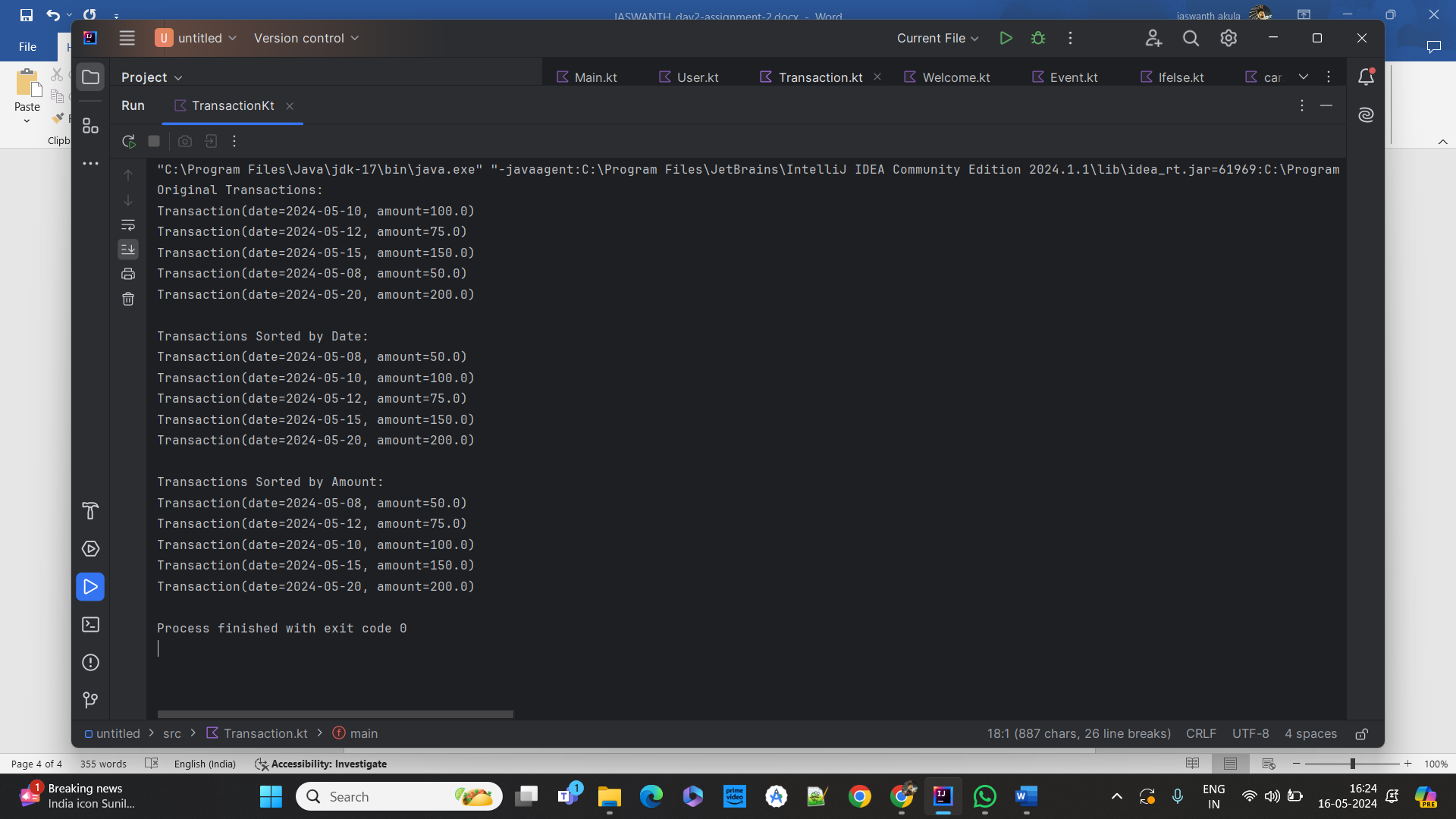
}

fun <T: Comparable<T>> sortTransactions(transactions: List<Transaction>, selector: (Transaction) -> T): List<Transaction> {

return transactions.sortedBy(selector)

}

## Output:



Task 8: Implement inheritance by creating specific transaction classes like Income and Expense that inherit from Transaction.

// Base class

open class Transaction(

val amount: Double,

val description: String

) {

open fun displayDetails() {

println("Amount: $amount, Description: $description")

}

}

// Derived class for Income

class Income(

amount: Double,

description: String,

val source: String

) : Transaction(amount, description) {

override fun displayDetails() {

super.displayDetails()

println("Source: $source")

}

}

// Derived class for Expense

class Expense(

amount: Double,

description: String,

val category: String

) : Transaction(amount, description) {

override fun displayDetails() {

super.displayDetails()

println("Category: $category")

}

}

fun main() {

val income = Income(1000.0, "Salary", "Job")

val expense = Expense(50.0, "Groceries", "Food")

income.displayDetails()

expense.displayDetails()

}

## Output:

