**Day 2: Functions and OOP Basics**

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

package task5to8  
data class Transaction(val id:Int, var description:String, var amount:Double)  
class TransactionList{  
 val transactions = *mutableListOf*<Transaction>()  
 // Add a transaction  
 fun addTransaction(transaction: Transaction){  
 transactions.add(transaction)  
 *println*("Transaction Added : $transaction")  
 }  
 // Delete a transaction by its ID  
 fun deleteTransaction(transactionId:Int):Boolean{  
 val transaction = transactions.*find* **{ it**.id == transactionId **}** return if(transaction != null){  
 transactions.remove(transaction)  
 *println*("Transaction Deleted: $transaction")  
 true  
 }else{  
 *println*("Transaction with Id $transactionId not found")  
 false  
 }  
 }  
  
 // Edit a transaction by its ID  
 fun editTransaction(transactionId: Int, newDescription: String, newAmount: Double): Boolean {  
 val transaction = transactions.*find* **{ it**.id == transactionId **}** return if (transaction != null) {  
 transaction.description = newDescription  
 transaction.amount = newAmount  
 *println*("Transaction updated: $transaction")  
 true  
 } else {  
 *println*("Transaction with ID $transactionId not found")  
 false  
 }  
 }  
  
 // List all transactions  
 fun listTransactions() {  
 if (transactions.isEmpty()) {  
 *println*("No transactions available")  
 } else {  
 *println*("Transactions:")  
 transactions.*forEach* **{** *println*(**it**) **}** }  
 }  
}  
  
fun main(){  
 val transactionList = TransactionList()  
 // Add transactions  
 transactionList.addTransaction(Transaction(1, "Entertainment", 200.0))  
 transactionList.addTransaction(Transaction(2, "Fuel", 40.0))  
  
 // List transactions  
 transactionList.listTransactions()  
  
 // Edit a transaction  
 transactionList.editTransaction(1, "Supermarket Shopping", 60.0)  
  
 // Delete a transaction  
 transactionList.deleteTransaction(2)  
  
 // List transactions after editing and deleting  
 transactionList.listTransactions()  
}

//Output

Transaction Added : Transaction(id=1, description=Entertainment, amount=200.0)

Transaction Added : Transaction(id=2, description=Fuel, amount=40.0)

Transactions:

Transaction(id=1, description=Entertainment, amount=200.0)

Transaction(id=2, description=Fuel, amount=40.0)

Transaction updated: Transaction(id=1, description=Supermarket Shopping, amount=60.0)

Transaction Deleted: Transaction(id=2, description=Fuel, amount=40.0)

Transactions:

Transaction(id=1, description=Supermarket Shopping, amount=60.0)

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

package task5to8  
data class Expense(val id:Int, val description:String, val amount:Double)  
class User(private val username: String, private val password: String){  
 private var isLoggedIn = false  
 private val expenses = *mutableListOf*<Expense>()  
  
 // Method to log in  
 fun login(inputUsername: String, inputPassword: String): Boolean {  
 return if (inputUsername == username && inputPassword == password) {  
 isLoggedIn = true  
 *println*("Login successful. Welcome, $username!")  
 true  
 } else {  
 *println*("Login failed. Invalid username or password.")  
 false  
 }  
 }  
  
 // Add an expense (only if logged in)  
 fun addExpense(expense: Expense) {  
 if (isLoggedIn) {  
 expenses.add(expense)  
 *println*("Expense added: $expense")  
 } else {  
 *println*("You must be logged in to add expenses")  
 }  
 }  
  
 // Display a summary of expenses  
 fun displayExpensesSummary() {  
 if (isLoggedIn) {  
 if (expenses.isEmpty()) {  
 *println*("No expenses recorded.")  
 } else {  
 val totalExpenses = expenses.*sumOf* **{ it**.amount **}** *println*("Expenses Summary:")  
 expenses.*forEach* **{** *println*(**it**) **}** *println*("Total Expenses: $totalExpenses")  
 }  
 } else {  
 *println*("You must be logged in to view expenses")  
 }  
 }  
}  
  
fun main() {  
 val user = User("Basheer", "Ahamed@123")  
  
 // Attempt to log in  
 if (user.login("Basheer", "Ahamed@123")) {  
 // Add expenses  
 user.addExpense(Expense(1, "Shopping", 500.0))  
 user.addExpense(Expense(2, "Electricity Bill", 75.5))  
  
 // Display expenses summary  
 user.displayExpensesSummary()  
 } else {  
 *println*("Unable to proceed. Login required")  
 }  
}

//Output

Login successful. Welcome, Basheer!

Expense added: Expense(id=1, description=Shopping, amount=500.0)

Expense added: Expense(id=2, description=Electricity Bill, amount=75.5)

Expenses Summary:

Expense(id=1, description=Shopping, amount=500.0)

Expense(id=2, description=Electricity Bill, amount=75.5)

Total Expenses: 575.5

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

package task5to8  
import java.time.LocalDate  
data class Transactions(val id: Int, val description: String, val amount: Double, val date: LocalDate)  
class TransactionsList {  
 private val transactions = *mutableListOf*<Transactions>()  
  
 // Add a transaction  
 fun addTransaction(transaction: Transactions) {  
 transactions.add(transaction)  
 *println*("Transaction added: $transaction")  
 }  
  
 // Filter transactions by a predicate  
 fun filterTransactions(predicate: (Transactions) -> Boolean): List<Transactions> {  
 return transactions.*filter*(predicate)  
 }  
  
 // Sort transactions by a comparator  
 fun sortTransactions(comparator: Comparator<Transactions>): List<Transactions> {  
 return transactions.*sortedWith*(comparator)  
 }  
}  
  
fun main() {  
 val transactionList = TransactionsList()  
  
 transactionList.addTransaction(Transactions(1, "Entertainment", 200.0, LocalDate.of(2025, 1, 5)))  
 transactionList.addTransaction(Transactions(2, "Shopping", 100.0, LocalDate.of(2025, 1, 3)))  
 transactionList.addTransaction(Transactions(3, "Fuel", 50.0, LocalDate.of(2025, 1, 1)))  
  
 // Filter transactions by amount greater than 100  
 val highValueTransactions = transactionList.filterTransactions **{ it**.amount > 100 **}** *println*("Filtered Transactions (Amount > 100): $highValueTransactions")  
  
 // Sort transactions by date  
 val sortedByDate = transactionList.sortTransactions(*compareBy* **{ it**.date **}**)  
 *println*("Transactions Sorted by Date: $sortedByDate")  
  
 // Sort transactions by amount in descending order  
 val sortedByAmountDescending = transactionList.sortTransactions(*compareByDescending* **{ it**.amount **}**)  
 *println*("Transactions Sorted by Amount (Descending): $sortedByAmountDescending")  
}

//Output

Transaction added: Transactions(id=1, description=Entertainment, amount=200.0, date=2025-01-05)

Transaction added: Transactions(id=2, description=Shopping, amount=100.0, date=2025-01-03)

Transaction added: Transactions(id=3, description=Fuel, amount=50.0, date=2025-01-01)

Filtered Transactions (Amount > 100): [Transactions(id=1, description=Entertainment, amount=200.0, date=2025-01-05)]

Transactions Sorted by Date: [Transactions(id=3, description=Fuel, amount=50.0, date=2025-01-01), Transactions(id=2, description=Shopping, amount=100.0, date=2025-01-03), Transactions(id=1, description=Entertainment, amount=200.0, date=2025-01-05)]

Transactions Sorted by Amount (Descending): [Transactions(id=1, description=Entertainment, amount=200.0, date=2025-01-05), Transactions(id=2, description=Shopping, amount=100.0, date=2025-01-03), Transactions(id=3, description=Fuel, amount=50.0, date=2025-01-01)]

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

package task5to8  
import java.time.LocalDate  
open class TransactionIncome(val id: Int, val description: String, val amount: Double, val date: LocalDate){  
 open fun details(): String {  
 return "TransactionIncome(id=$id, description='$description', amount=$amount, date=$date)"  
 }  
}  
  
// Income class inheriting from Transaction  
class Income(id: Int, description: String, amount: Double, date: LocalDate, val source: String):TransactionIncome(id, description, amount, date){  
 override fun details(): String {  
 return "Income(id=$id, description='$description', amount=$amount, date=$date, source='$source')"  
 }  
}  
  
// Expense class inheriting from Transaction  
class Expenses(id: Int, description: String, amount: Double, date: LocalDate, val category: String) : TransactionIncome(id, description, amount, date) {  
 override fun details(): String {  
 return "Expense(id=$id, description='$description', amount=$amount, date=$date, category='$category')"  
 }  
}  
  
fun main() {  
 val transactions = *listOf*(  
 Income(1, "Salary", 3000.0, LocalDate.of(2025, 1, 10), "Company A"),  
 Income(2, "Freelancing", 500.0, LocalDate.of(2025, 1, 20), "Client B"),  
 Expenses(3, "Groceries", 150.0, LocalDate.of(2025, 1, 12), "Food"),  
 Expenses(4, "Electricity Bill", 75.5, LocalDate.of(2025, 1, 15), "Utilities")  
 )  
  
 *println*("Transaction Details:")  
 transactions.*forEach* **{** *println*(**it**.details()) **}**}

//Output

Transaction Details:

Income(id=1, description='Salary', amount=3000.0, date=2025-01-10, source='Company A')

Income(id=2, description='Freelancing', amount=500.0, date=2025-01-20, source='Client B')

Expense(id=3, description='Groceries', amount=150.0, date=2025-01-12, category='Food')

Expense(id=4, description='Electricity Bill', amount=75.5, date=2025-01-15, category='Utilities')