DAY 4 TASKS

**Task 1:** Implement null safety features to handle the absence of transaction data.  
CODE:

package day4tasks1to4  
data class Transaction(val id: Int, val amount: Double, val description: String,val date:String)  
  
class TransactionManager {  
  
 private val transactions: MutableList<Transaction?> = *mutableListOf*()  
  
 fun addTransaction(transaction: Transaction?) {  
 if (transaction != null) {  
 transactions.add(transaction)  
 *println*("Transaction added: $transaction")  
 } else {  
 *println*("Cannot add because this is null transaction.")  
 }  
 }  
 fun displayTransactions() {  
 if (transactions.isEmpty()) {  
 *println*("No transactions available")  
 } else {  
 *println*("Transactions List:")  
 transactions.*forEach* **{** transaction **->** *println*(transaction?.*let* **{** "ID: ${**it**.id}, Amount: ${**it**.amount}, Description: ${**it**.description},date:${**it**.date}" **}** ?: "No transaction data")  
 **}** }  
 }  
  
 fun findTransactionById(transactionId: Int): Transaction? {  
 return transactions.*find* **{ it**?.id == transactionId **}**?.*also* **{** *println*("Found transaction: $**it**")  
 **}** ?: *run* **{** *println*("Transaction with ID $transactionId not found.")  
 null  
 **}** }  
}  
  
fun main() {  
 val manager = TransactionManager()  
  
 val transaction1 = Transaction(1, 300.0, "Groceries","2025-01-10")  
 val transaction2 = Transaction(2, 100.0, "Food","2025-01-11")  
 val transaction3=Transaction(3,500.0,"Entertainment","2025-01-12")  
 val transaction4: Transaction? = null  
  
 manager.addTransaction(transaction1)  
 manager.addTransaction(transaction2)  
 manager.addTransaction(transaction3)  
 manager.addTransaction(transaction4)  
  
  
 // display all transactions  
 manager.displayTransactions()  
  
 manager.findTransactionById(1)  
 manager.findTransactionById(6)  
}

OUTPUT:

Transaction added: Transaction(id=1, amount=300.0, description=Groceries, date=2025-01-10)

Transaction added: Transaction(id=2, amount=100.0, description=Food, date=2025-01-11)

Transaction added: Transaction(id=3, amount=500.0, description=Entertainment, date=2025-01-12)

Cannot add because this is null transaction.

Transactions List:

ID: 1, Amount: 300.0, Description: Groceries,date:2025-01-10

ID: 2, Amount: 100.0, Description: Food,date:2025-01-11

ID: 3, Amount: 500.0, Description: Entertainment,date:2025-01-12

Found transaction: Transaction(id=1, amount=300.0, description=Groceries, date=2025-01-10)

Transaction with ID 6 not found.

**Task 2:** Write custom exception classes to handle errors related to transaction processing.

CODE:

package day4tasks1to4  
//CustomExpensions  
open class TransactionException(message: String) : Exception(message)  
class InsufficientFundsException(message: String) : Exception(message)  
  
class InvalidTransactionException(message: String) : Exception(message)  
  
class TransactionAlreadyProcessedException(message: String) : Exception(message)  
  
  
class UnauthorizedTransactionException(message: String) : Exception(message)  
  
fun processTransaction(amount: Double, accountBalance: Double, transactionId: String, isAuthorized: Boolean) {  
 if (!isAuthorized) {  
 throw UnauthorizedTransactionException("Transaction $transactionId is not authorized.")  
 }  
  
 if (amount > accountBalance) {  
 throw InsufficientFundsException("Insufficient funds for transaction $transactionId. Requested: $amount, Available: $accountBalance")  
 }  
 val transactionProcessed = false  
 if (!transactionProcessed) {  
 throw TransactionException("Transaction $transactionId timed out. Please try again later.")  
 }  
  
 val alreadyProcessed = false  
 if (alreadyProcessed) {  
 throw TransactionAlreadyProcessedException("Transaction $transactionId has already been processed.")  
 }  
  
 // proceed with the transaction  
 *println*("Transaction $transactionId processed successfully with amount: $amount")  
}  
  
fun main() {  
 try {  
 val accountBalance = 100.0  
 val amount = 500.0  
 val transactionId = "TXN123"  
 val isAuthorized = true  
  
 *processTransaction*(amount, accountBalance, transactionId, isAuthorized)  
 } catch (e: InsufficientFundsException) {  
 *println*("Error: ${e.message}")  
 } catch (e: InvalidTransactionException) {  
 *println*("Error: ${e.message}")  
 } catch (e: TransactionException) {  
 *println*("Error: ${e.message}")  
 } catch (e: TransactionAlreadyProcessedException) {  
 *println*("Error: ${e.message}")  
 } catch (e: UnauthorizedTransactionException) {  
 *println*("Error: ${e.message}")  
 } catch (e: Exception) {  
 *println*("An unexpected error occurred: ${e.message}")  
 }  
}

OUTPUT:

Error: Insufficient funds for transaction TXN123. Requested: 500.0, Available: 100.0

**Task 3:** Create extension functions for the List<Transaction> class to calculate total expenses and incomes.  
CODE:

package day4tasks1to4  
data class Trans(val amount: Double, val type: TransactionType)  
  
// Enum class for transaction types  
enum class TransactionType {  
 *INCOME*,  
 *EXPENSE*}  
  
fun List<Trans>.totalExpenses(): Double {  
 return this.*filter* **{ it**.type == TransactionType.*EXPENSE* **}** .*sumOf* **{ it**.amount **}**}  
  
fun List<Trans>.totalIncomes(): Double {  
 return this.*filter* **{ it**.type == TransactionType.*INCOME* **}** .*sumOf* **{ it**.amount **}**}  
  
fun main() {  
 val transactions = *listOf*(  
 Trans(400.0, TransactionType.*INCOME*),  
 Trans(200.0, TransactionType.*EXPENSE*),  
 Trans(300.0, TransactionType.*INCOME*),  
 Trans(100.0, TransactionType.*EXPENSE*)  
 )  
  
 val totalExpenses = transactions.*totalExpenses*()  
 val totalIncomes = transactions.*totalIncomes*()  
  
 *println*("Total Expenses: $$totalExpenses")  
 *println*("Total Incomes: $$totalIncomes")  
}

OUTPUT:

Total Expenses: $300.0

Total Incomes: $700.0

**Task 4:** Use coroutines to handle simultaneous processing of importing and exporting transaction data without blocking the main thread.  
CODE:

package day4tasks1to4  
  
import kotlinx.coroutines.\*  
  
fun main() = *runBlocking* **{** *println*("Starting transaction processing")  
  
 val import = *async* **{** importTransactionData() **}** val export = *async* **{** exportTransactionData() **}** val importResult = import.await()  
 val exportResult = export.await()  
  
 *println*("Import Result: $importResult")  
 *println*("Export Result: $exportResult")  
  
 *println*("Transaction processing complete")  
**}**suspend fun importTransactionData(): Int {  
 *println*("Importing transaction data")  
 delay(2000) // long-running import task  
 *println*("Import complete.")  
 return 100  
}  
  
suspend fun exportTransactionData():Int {  
 *println*("Exporting transaction data")  
 delay(3000) // Simulating a long-running export task  
 *println*("Export complete.")  
 return 200  
}

OUTPUT:

Starting transaction processing

Importing transaction data

Exporting transaction data

Import complete.

Export complete.

Import Result: 100

Export Result: 200

Transaction processing complete