Use Case: Digital Petty Cash Ledger System

1. Introduction

A Petty Cash system is used by businesses to manage small, incidental expenditures (like tea, stationery, or taxi fares) using a "float" or "imprest" fund.[1] This use case requires the development of a **Digital Petty Cash Ledger** that allows a cashier to record income (replenishments) and expenses. The system must be built using C# and must demonstrate type-safe data handling without any external database.

2. Objective

The objective of this test is to verify the student's ability to:

- **Encapsulation & Abstraction:** Define clear data models for financial transactions.
- **Inheritance:** Differentiate between types of transactions (Income vs. Expense).
- **Generics:** Create a reusable Ledger or Repository to handle any transaction type while maintaining type safety.
- **Collections:** Efficiently store, retrieve, and filter transaction records using List<T> or Dictionary<K,V>.[2]

3. Requirements

The student must implement the following logic in C#:

A. The Data Model (OOP)

- **Abstract Class:** Transaction (Properties: Id, Date, Amount, Description).
- **Derived Classes:**
  - ExpenseTransaction: Adds a Category property (e.g., Office, Travel, Food).
  - IncomeTransaction: Adds a Source property (e.g., Main Cash, Bank Transfer).
- **Interface:** IReportable with a method GetSummary() to be implemented by all transaction types.

B. The Ledger Logic (Generics & Collections)

- **Generic Class:** Ledger<T> where T : Transaction.
- **Storage:** Internal List<T> to hold the transaction history.
- **Methods:**
  - AddEntry(T entry): Adds a transaction.
  - GetTransactionsByDate(DateTime date): Returns a filtered list.
  - CalculateTotal(): Uses LINQ or a loop to sum up the Amount.

C. Technical Restrictions

- **Storage:** In-memory only (no SQL, no File I/O).
- **Type Safety:** The generic Ledger must ensure that only Transaction objects can be processed.

4. Use Case Definition

Use Case ID	**UC-FIN-01**
Use Case Name	Record and Balance Petty Cash
Actor	Petty Cash Custodian
Pre-condition	The application is running; a ledger for Expenses and a ledger for Income are initialized.
Description	The custodian adds a fund replenishment (Income) and then records multiple minor expenses. The system then calculates the remaining balance.

Main Success Scenario (Steps)

1. The user creates a Ledger<IncomeTransaction> to track funds received from the main office.
2. The user records a $500 replenishment from "Main Cash".
3. The user creates a Ledger<ExpenseTransaction> to track daily spends.
4. The user records an expense of $20 for "Stationery" and $15 for "Team Snacks".
5. The system uses a generic method to display the total from both ledgers.
6. The user calculates the **Net Balance** (Total Income - Total Expenses).

5. Expected Results

The student's solution should produce the following output:

- **Compile-time Check:** If a student tries to add an IncomeTransaction into a Ledger<ExpenseTransaction>, the code should fail to compile (proving Generic constraints).
- **Correct Calculations:** The console must show the total spent ($35) and the total received ($500) accurately.
- **Polymorphic Output:** A loop iterating through a List<Transaction> should be able to call GetSummary() and display unique details for both Income and Expenses.

6. Conclusion

By completing this use case, the student demonstrates a professional understanding of **Software Architecture**. Moving beyond simple variables to a Generic-based Ledger system shows that the developer can build scalable financial tools that are less prone to runtime errors and easy to maintain as business requirements grow.