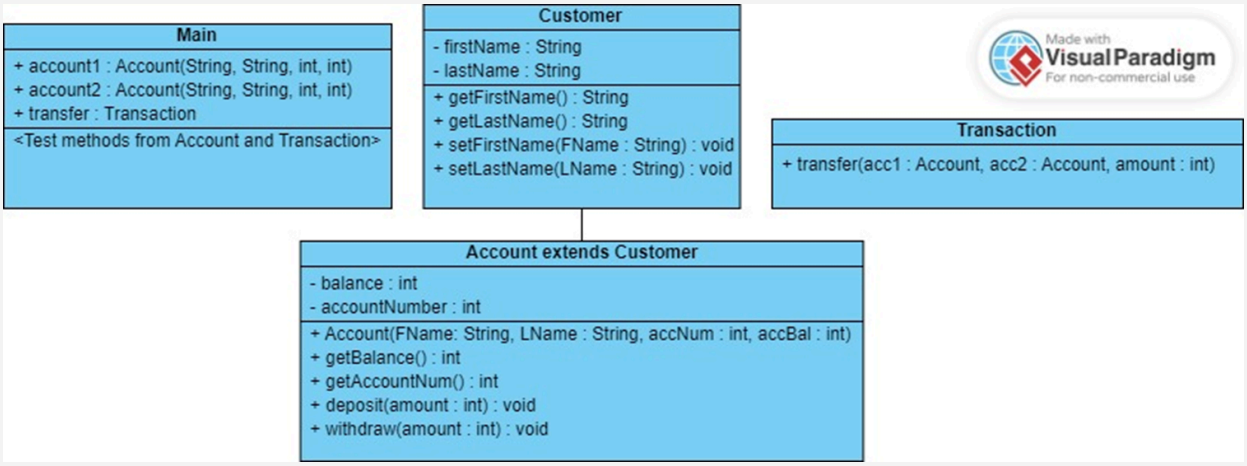
## Basic Banking System (OOP Foundation)

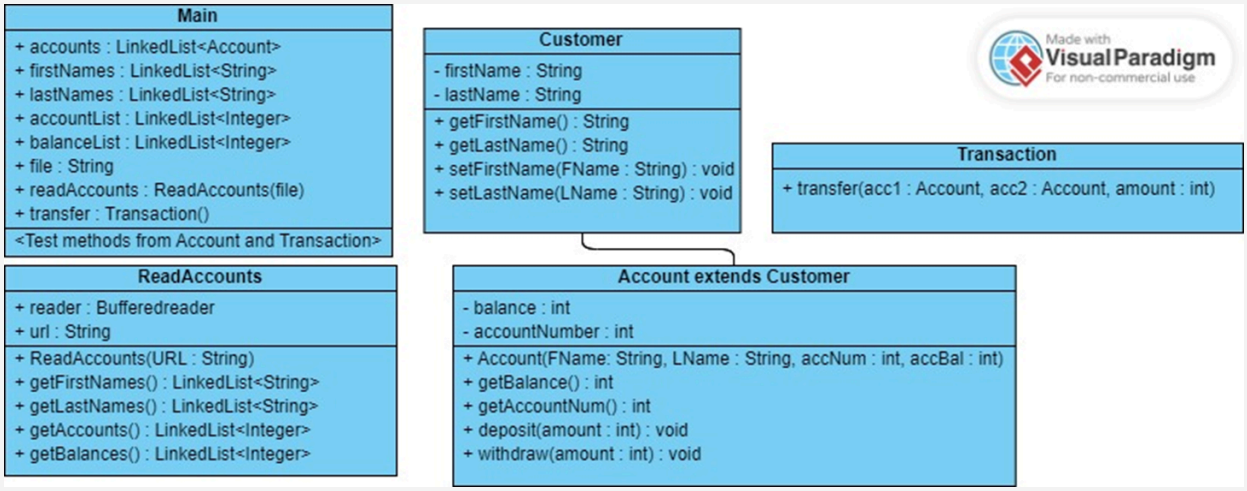



Part 1 aims to create a simple banking system on the basis of fundamental object-oriented programming (OOP) principles. The system has three primary classes namely Customer, Account and Transaction. The Customer class is a container of some minimal personal information such as first and last name with getter and setter functions to retrieve and update this information. The Account class is an extension of Customer, and shares these properties with the new ones such as balance and accountNumber. It also outlines the procedures like deposit, and withdrawal to manipulate the balance. The classes of transactions have one method transfer that helps to transfer money between two accounts. The Main class does the testing - it opens two sample accounts and makes deposit, withdrawal and transfer operations. In this section the concepts of OOP are presented like, inheritance, encapsulation, interaction of objects and testing of methods.

- `Customer`:
    - Attributes: `firstName`, `lastName`
    - Methods: Getters and Setters for names
- `Account extends Customer`:
    - Adds `accountNumber` and `balance`
    - Methods: `deposit()`, `withdraw()`, `getBalance()`
- `Transaction`:

- Method `transfer(acc1, acc2, amount)`

- `Main`:
  - Tests all methods by creating two accounts manually
  - Demonstrates OOP principles (inheritance, encapsulation)




## Banking System with CSV File Input

Part 2 expands the banking system by introducing file handling. Instead of hardcoding accounts, the system reads them from a CSV file named Accounts.csv. A new class, ReadAccounts, is introduced to handle file input. It reads the file line by line and extracts data into four separate LinkedLists: firstNames, lastNames, accountNumbers, and balances. These lists are then used in the Main class to create Account objects dynamically inside a loop. This change makes the system scalable and data-driven. The UML diagram reflects the addition of ReadAccounts and shows that the Main class now contains these four lists and calls the readAccounts object to load them.

Introduced `ReadAccounts` class:

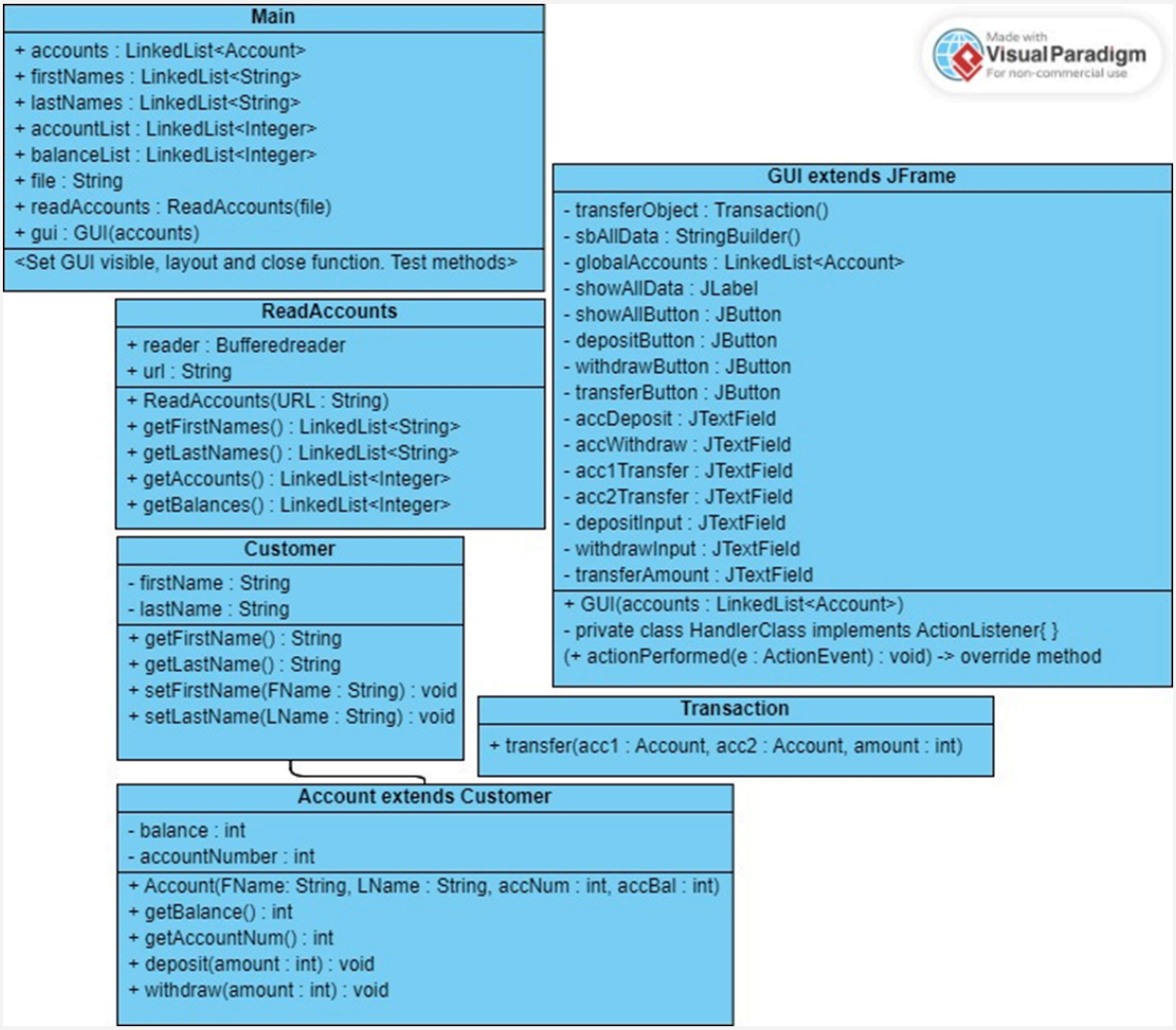
- Uses `BufferedReader` to read CSV

- Methods return: first names, last names, account numbers, balances

In `Main`:

- Replaces manual account creation with dynamic generation from CSV
- Uses four `LinkedLists` and a loop to populate `accounts`

## GUI-Integrated Banking System




A new class GUI extends JFrame and displays account data and control buttons on a window. It includes buttons for depositing, withdrawing, transferring funds, and viewing all accounts. Text fields are provided for users to input account numbers and amounts. The constructor of the GUI class sets up the layout, initializes all components, and uses a StringBuilder to display account information. Event handling is implemented through a private inner class HandlerClass that implements ActionListener. This class detects which button was clicked and performs the corresponding action, such as calling deposit() or transfer(). Updates are made live in the GUI and later saved to the CSV file via the WriteAccounts class. This part demonstrates event-driven programming, GUI design using Swing, and real-time file updates.

`GUI extends JFrame`:

- Contains text fields, buttons, and a label to show data
- Constructor sets layout and default content

`HandlerClass implements ActionListener`:

- Responds to button clicks (deposit, withdraw, transfer)
- Updates account data and saves to CSV

Real-time feedback:

- GUI loads account data at startup
- Placeholder text disappears on focus

Uses `WriteAccounts` to persist changes after each transaction