Title: Banking System Class Diagram

# Objective:

Design and implement a **Banking System** in Java using the given **UML Class Diagram**, applying **Object-Oriented Programming concepts** such as abstraction, inheritance, polymorphism, association, composition, interface, constructor overloading, and method overloading/overriding.

# Classes to Implement:

## BankService (interface)

Represents services provided by a bank.  
**Abstract Methods**:

* + openAccount()
  + closeAccount()
  + transferFunds(fromAccount, toAccount, amount)

## BankAccount (abstract class)

Superclass for all types of bank accounts.  
**Fields**:

* + accountNumber: int
  + balance: double
  + customer: Customer

**Constructor Overloading**:

* + One with just balance
  + One with accountNumber, balance, and customer

**Methods**:

* + deposit(double amount)
  + withdraw(double amount)
  + calculateInterest(): double (abstract)

## SavingsAccount (extends BankAccount)

* + Overrides calculateInterest() to provide fixed interest rate.

## CurrentAccount (extends BankAccount)

* + Overrides calculateInterest() with custom logic (or 0 if no interest).

## Customer

* + name: String
  + address: Address
  + account: BankAccount
  + Constructor with and without address (overloading)
  + viewBalance() – prints balance of their account.

## Address

* + street: String
  + city: String
  + zip: String

## Transaction

* + txnId: int
  + fromAccount: BankAccount
  + toAccount: BankAccount
  + makeTransfer(double amount) – deduct from fromAccount, credit to toAccount

## ATM

* + location: String
  + withdraw(BankAccount acc, double amount)
  + deposit(BankAccount acc, double amount)

# Relationships Between Classes:

|  |  |  |  |
| --- | --- | --- | --- |
| Class | Relationship Type | Related Class | Description |
| BankAccount | IS-A | SavingsAccount, CurrentAccount | Inheritance from abstract class |
| BankAccount | Implements | BankService | Interface with account operations |
| Customer | HAS-A | BankAccount | Each customer has an account |
| Customer | HAS-A | Address | Each customer has an address |
| Transaction | USES-A | BankAccount | Transfers use accounts |
| ATM | USES-A | BankAccount | ATMs perform operations on accounts |

# Requirements for Submission:

* Java class files implementing the diagram
* UML diagram in .jpg/.png or draw.io format
* A working main() method demonstrating:
  + Account creation
  + Deposit, withdrawal via ATM
  + Interest calculation
  + Fund transfer between two customers

# Key OOP Concepts to Implement:

|  |  |
| --- | --- |
| Concept | Example from Design |
| Class & Object | All classes like Customer, ATM, etc. |
| Constructor Overloading | Customer(String name) and Customer(String name, Address addr) |
| Method Overloading | withdraw(amount) and withdraw(amount, mode) in ATM |
| Method Overriding | calculateInterest() in SavingsAccount, CurrentAccount |
| IS-A | SavingsAccount IS-A BankAccount |
| HAS-A | Customer HAS-A BankAccount, Customer HAS-A Address |
| USED-A | ATM uses BankAccount |
| Abstract Class | BankAccount |
| Interface with Abstract Methods | BankService |

# Sample Main Flow (optional for reference):

public class Main {

public static void main(String[] args) {

Address addr = new Address("MG Road", "Bangalore", "560001");

Customer cust1 = new Customer("Alice", addr);

BankAccount acc1 = new SavingsAccount(1001, 5000.0, cust1);

ATM atm = new ATM("MG Road ATM");

atm.deposit(acc1, 1500.0);

atm.withdraw(acc1, 1000.0);

Customer cust2 = new Customer("Bob", addr);

BankAccount acc2 = new CurrentAccount(1002, 8000.0, cust2);

Transaction txn = new Transaction(1, acc1, acc2);

txn.makeTransfer(2000.0);

System.out.println("Interest on Alice's Account: " + acc1.calculateInterest());

}

}

# Notes:

* Validate balance before withdrawal or transfer.
* Use super() keyword when chaining constructors.
* Practice @Override annotation and understand dynamic dispatch.
* Try additional features like printing mini-statements, logging transactions, etc.