Lab 2: Inheritance and Polymorphism

Exercise1: Create Person and Account Class as shown below in class diagram. Ensure minimum balance of INR 500 in a bank account is available.

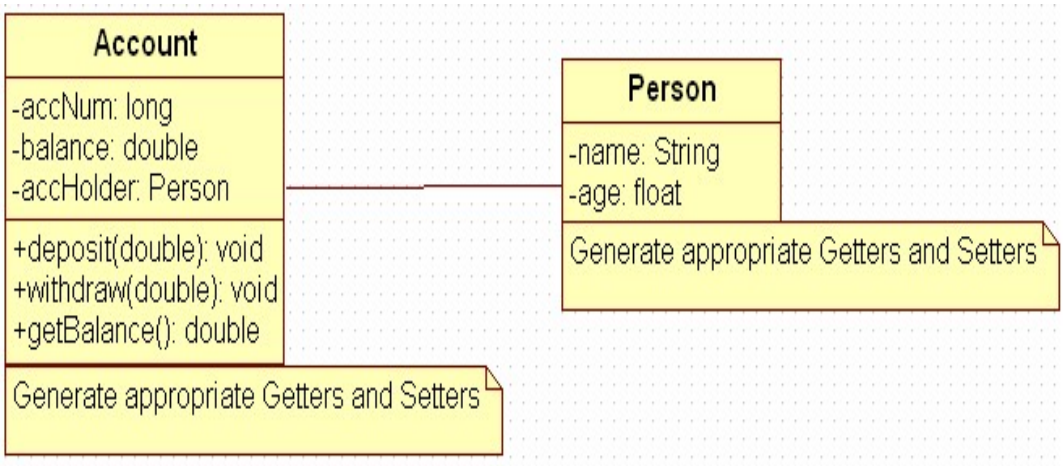

Figure 1: Association of person with account class

a) Create Account for smith with initial balance as INR 2000 and for Kathy with initial balance as 3000.(accNum should be auto generated).
b) Deposit 2000 INR to smith account.
c) Withdraw 2000 INR from Kathy account.
d) Display updated balances in both the account.
e) Extend the functionality through Inheritance and polymorphism. Inherit two classes Savings Account and Current Account from account class. And Implement the following in the respective classes.

- **Savings Account**

a. Add a variable called minimum Balance and assign final modifier.
b. Override method called withdraw (This method should check for minimum balance and allow withdraw to happen)

- **Current Account**

a. Add a variable called overdraft Limit
b. Override method called withdraw (checks whether overdraft limit is reached and returns a Boolean value accordingly)

Exercise 2: create packages and classes as given below:

a) com.cg.eis.bean

In this package, create “Employee” class with different attributes such as id, name, salary, designation, insuranceScheme.

b) com.cg.eis.service

This package will contain code for services offered in Employee Insurance System. The service class will have one EmployeeService Interface and its corresponding implementation class.

The services offered by this class are:

i) Get employee details from user.

ii) Find the insurance scheme for an employee based on salary and designation.

iii) Display all the details of an employee.

c) com.cg.eis.pl

This package will contain code for getting input from user, produce expected output to the user and invoke services offered by the system.

Exercise 3:

Using an inheritance hierarchy, design a Java program to model items at a library (books, journal articles, videos and CDs.) Have an abstract superclass called Item and include common information that the library must have for every item (such as unique identification number, title, and number of copies). No actual objects of type Item will be created - each actual item will be an object of a (non-abstract) subclass. Place item-type-specific behavior in subclasses (such as a video's year of release, a CD's musical genre, or a book's author).
More in detail:

1. Implement an abstract superclass called Item and define all common operations on this class (constructors, getters, setters, equals, toString, print, checkIn, checkOut, addItem, etc). Have private data for: identification number, title, and number of copies.

2. Implement an abstract subclass of Item named WrittenItem and define all common operations on this class. Added private data for author.

3. Implement 2 subclasses of WrittenItem: Book and JournalPaper.

    3.1. Class Book: no new private data. When needed, override/overload methods from the superclass.
    3.2. Class JournalPaper: added private data for year published. When needed, override/overload methods from the superclass.

4. Implement another abstract subclass of Item named MediaItem and define all common operations on this class. Added private data for runtime (integer).

5. Implement 2 subclasses of MediaItem: Video and CD.

    5.1. Class Video: added private data for director, genre and year released. When needed, override/overload methods from the superclass.
    5.2. Class CD: added private data for artist and genre. When needed, override/overload methods from the superclass.

Write the definitions of these classes and a client program (your choice!) showing them in use.