Lab7: Analyze, Plan, Design, Implement

Objectives

- Analyzing a problem
- Planning
- Designing a solution
- Implementing the solution in a programming language

Part I: Select only one of the following questions and complete a **simple** design

Please create a package named as **lab7Part1QuestionX** where X will be one of the questions given below(1-5). Your solution will be under this package.

1. **COLLEGE:** Design a class named **Person** and its two subclasses named **Student** and **Employee**. Make **Faculty** and **Staff** subclasses of **Employee**. A person has a name, address, phone number, and email address. A student has a class status (freshman, sophomore, junior, or senior). An employee has an office, salary, and date hired. Define a class named **MyDate** that contains the fields year, month, and day. Use MyDate class to define date hired for employee. A faculty member has office hours and a rank. A staff member has a title. Override the toString method in each class to display the class name and the person's name.

   a. Implement the classes in Java.
   b. Include all constructors, accessors, mutators, generic methods such as toString, equals, etc.
   c. Use chaining wherever possible.
   d. Write a test program that creates a Person, Student, Employee, Faculty, and Staff with using **polymorphic referencing**, and invokes their toString() methods.

2. **BANK:** Design a class named **Person** and its two subclasses named **Customer** and **Employee**. Design another class named **Info** and its two subclasses named **CustomerInfo** and **EmployeeInfo**. The basic **Info** has a name, address, phone number, and email address. Additionally, **CustomerInfo** should have also account number, password, balance, blocked, and **EmployeeInfo** should have security number, salary, and department information. The **Customer** can apply for a new account, deposit, or withdraw money, display/modify his/her information. An **Employee** will help the **Customer** for these operations. In a **Bank** class simulate these operations. A **Bank** can have at most 100 **Employees** and 1000 **Customers.** Override the toString method in each class to display the class name and the related information.

   a. Implement the classes in Java.
   b. Include all constructors, accessors, mutators, generic methods such as toString, equals, etc.
   c. Use chaining wherever possible.
   d. Write a test program that creates Person, Customer, Employee, Info, CustomerInfo, EmployeeInfo, Bank objects with using **polymorphic referencing**, and invokes their toString() methods.

3. **SHAPE:** Design a class named **Shape** and its two subclasses named **TwoDimShape** and **ThreeDimShape**. Make **Rectangle** and **Circle** subclasses of **TwoDimShape**. Make **RectangularPrism** and **Sphere** subclasses of **ThreeDimShape.** A shape will have a draw and an area method. Furthermore, **ThreeDimShape** has volume calculation Override the toString method in each class to display the class name and the person's name.

   a. Implement the classes in Java.
   b. Include all constructors, accessors, mutators, generic methods such as toString, equals, etc.
   c. Use chaining wherever possible.
   d. Write a test program that creates Shape, TwoDimShape, ThreeDimShape, Rectangle, Circle, TwoDimShape, RectangularPrism and Sphere with using **polymorphic referencing**, and invokes their toString() methods.

4. **ANIMAL:** Design a class named **Animal** and its two subclasses named **Aquatic** and **LandBased**. Make **Dog, Cat** and **Lion** subclasses of **LandBased**. Animal has a name and can move. **Dog** and **Cat** also implements **Pet** interface. A pet needs feeding and and health care, write them in pet interface as abstract and implement these methods in dog and cat. Override the toString method in each class to display the class name and the related information.

   a. Implement the classes in Java.
   b. Include all constructors, accessors, mutators, generic methods such as toString, equals, etc.
   c. Use chaining wherever possible.
   d. Write a test program that creates a Animal, Aquatic, LnadBased, Dog, Cat, Person, Student, Employee, Faculty, and Staff with using **polymorphic referencing**, and invokes their toString() methods.

5. **TRANSPORTATION:** Design a class named **Person** and its two subclasses named **Passenger** and **Employee**. Design another class named **Info** and its two subclasses named **PassengerInfo** and **EmployeeInfo**. The basic **Info** has a name, address, phone number and email address. Additionally, **PassengerInfo** should also have credit card number and **EmployeeInfo** should have social security number, salary. There will be an interface **Travel** which has methods like booking, buying ticket. The **Passenger** will implement **Travel.** An **Employee** will help the **Passenger** for these operations. In a **TravelAgency** class simulate these operations. A **TravelAgency** can have at most 100 **Employees** and 1000 **Passengers.** Override the toString method in each class to display the class name and the related information.

   a. Implement the classes in Java.
   b. Include all constructors, accessors, mutators, generic methods such as toString, equals, etc.
   c. Use chaining wherever possible.
   d. Write a test program that creates a Person, Passenger, Employee, Info, PassengerInfo, EmployeeInfo, Travel and TravelAgency with using **polymorphic referencing**, and invokes their toString() methods.

Part II: Problem Specification: Banking

Please create a package named as **lab7Part2**. Your solution will be under this package.

Write a program for banking operations. You will have many customers and your operations will be

- changing password,
- depositing,
- withdrawal, and
- printing the current account information.

Your program should

- prompt the user with the menu,
- read in the user's choice,
- execute the choice,
- and then re-prompt the user with the menu and continue until the user chooses to quit.

If a customer wants to login, you will ask

- the account number (test until a valid account number is given or exit is selected by -1) and
- the password (test the validity of password if it is given wrong three times block the account).
- Then ask for the operation and
  - if it is deposit/withdrawal ask for the amount,
  - if it is printing show the account information in a detailed way.
  - Continue these operations until the user wants to quit.

Use arrays and classes. You may use Scanner/System.out.print OR JOptionPane dialog boxes for input/output.

You need to write a class for

- **Customer** that has
  - the required attributes and methods for a customer.
  - Include **constructors, modifiers, and accessors**.
  - All attributes should be **private**.
- **Bank** that has
  - an **array of customers** and
  - required attributes and methods for the bank.
  - Include constructors, modifiers, and accessors.
  - Provide operations for new and existing customers.

<u>SEE SAMPLE EXECUTIONS BELOW!</u>

Sample execution:

```
Welcome
1. New account
2. Operations
3. Exit
Select (1-3):  1


Name: Zeynep Orhan
Address: MY ADDRESS
Phone: 1234567890
Initial Amount: 50
Password: xxxxxx
Re-type password: xxxxxx

Your account has been successfully completed as:

Name: Zeynep Orhan
Address: MY ADDRESS
Phone: 1234567890
Account number: 100
Balance: $50

Do not forget your account number and password
Press -1: Main menu 2: Operations
-1

Welcome
1. New account
2. Operations
3. Exit
Select (1-3):  2

Enter your account number: 100
Enter your password: xxxxxx
Operations:
        1.  Change password
        2.  Deposit
        3.  Withdraw
        4.  Print
        5.  Exit
        Select (1-5):  1
        New Password: 333
        Retype new password: 332
        They do not match. Retype again: 333
        Your password has been changed successfully.
Operations:
        1.  Change password
        2.  Deposit
        3.  Withdraw
        4.  Print
        5.  Exit
        Select (1-5):  2
        Amount: 100
        Operation successful
Operations:
        1.  Change password
        2.  Deposit
        3.  Withdraw
        4.  Print
        5.  Exit
        Select (1-5):  3
        Amount: 50
        Operation successful
Operations:
        1.  Change password
        2.  Deposit
        3.  Withdraw
        4.  Print
        5.  Exit
        Select (1-5):  4

Name: Zeynep Orhan
Address: MY ADDRESS
Phone: 1234567890
Account number: 100
Balance: $50

Operations:
        1.  Change password
        2.  Deposit
        3.  Withdraw
        4.  Print
        5.  Exit to main menu
        Select (1-5):  5
        Good Bye!


Welcome
1. New account
2. Operations
3. Exit
Select (1-3):   2

Enter your account number: 150
This is an invalid account please try again: 100
Enter your password: 123
Wrong password. Try again: 124
Wrong password. Try again: 125
Your account is blocked since your password is wrong!!! Call your bank immediately.

Welcome
1. New account
```

```
2. Operations
3. Exit
Select (1-3):  2
Enter your account number: 100
Your account has been blocked.

Welcome
1. New account
2. Operations
3. Exit
Select (1-3):  3
Operations done!
```

How to turn in this lab

Before turning in any program in this class, remember this mantra:

Just because it works does not mean it's good.

Your grade will also come from the following aspects of your code:

- Submission
- Accuracy/correctness
- Readability
- Neatness
- Presentation
- Style
- Testing
- Commenting

For all labs, turn in only an **electronic** version.
Please submit the followings after all labs:

- zip file of your project (the project folder, not just the .java file(s)): zip file name will be your **YourFirstNameLastNameLab7.zip**
- a single pdf file of all your codes (.java), screenshots of your output for each file: pdf file name will be **YourFirstNameLastNameLab7.pdf**

Submit

- **the zip and pdf files**
- after Monday Lab session until **SUNDAY 11 PM EST**
- from the Nexus Lab submission link that will be accessible in Week 2-10

Ask for help if you are having problems!