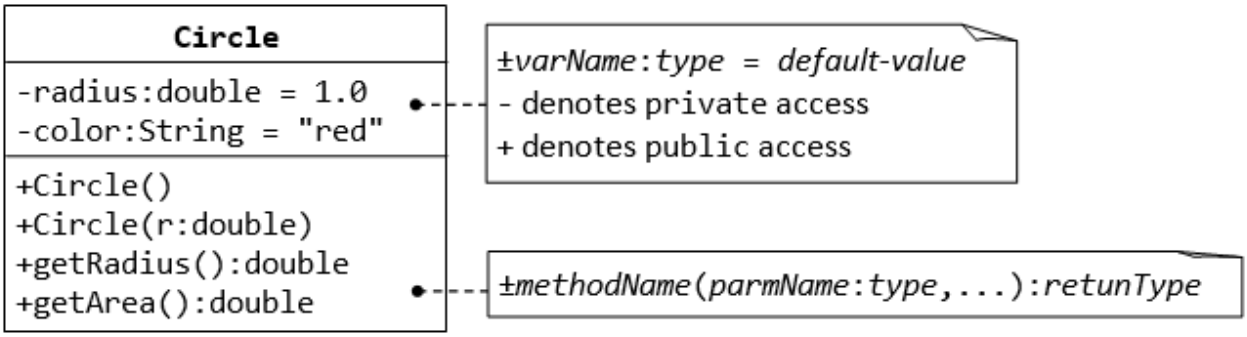
Day 2: Basic UML, classes, Object-Oriented programming

Q1: Implement following UML diagram, Write a program to test circle class.

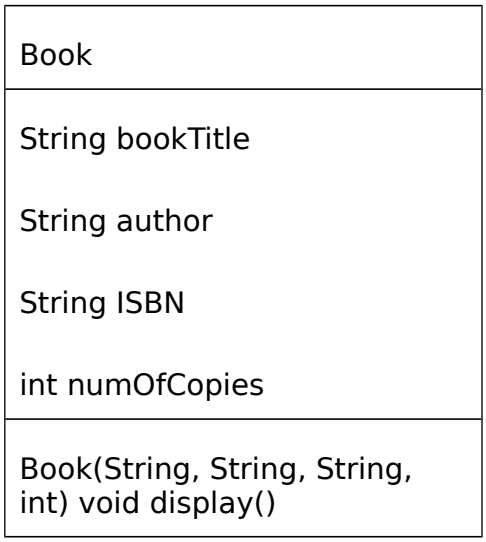


Q2. Create a book store application which will help a book store to keep the record of its books. For each book, the application will have the Book Title, Book Author, Book ISBN along with the number of copies for each book. The system will allow you to display all books, order new/existing books and sell books. With sell or order of existing books, number of copies will decrease/increase. With order of new book, a new book entry will be added to the system. The system will display a menu on the screen for the user to choose from. Here is the menu.

Enter “1”, to display the Books: Title – Author – ISBN - Quantity.
Enter “2”, to order new books.
Enter “3”, to sell books.
Enter “0”, to exit the system.

Here is what you need to do.

1) Create the following **Book** class.



display() method will display the book info in “Title – Author – ISBN – Quantity” format.

2) Create another class “**BookStore**” which should contain all the book objects. For now use an array of **Book** type and assume you can **have maximum 10 different books** but each could have **multiple copies**.

BookStore
Book[] books
void sell(String bookTitle, int noOfCopies) void order(String isbn, int nofOfCopies) void display()

a. ***sell(String, int)*** method will search for the book in "***books***" array using the bookTitle value. If the book is found in the list, number of copies will decrease. If the book is not found a message should display.

b. ***order(String, int)*** method will order book for the book store. You have to handle both new book and existing book scenario.
   i. First search for the book in "***books***" array using the isbn value.
   ii. If the book is found in the list (which means the book already exists in the system), number of copies will increase.
   iii. If the book is not found (which means the book does not exists in the system and you need to order new book), a new book entry will be added to the "***books***" array.

c. ***display()*** method will display info of all books in "books" array "Title – Author – ISBN – Quantity" format. Use ***Book*** class's ***display***() method to display each book's info.

3) Now create class "**BookStoreApp**" which should contain the **main** method. In main method create an object of **BookStore** class and then provide the **menu** as mentioned before. Once the user enters his/her option, you need to read the value and take appropriate action (See below) using the **BookStore** object.

- For option 1, **display** all the books in the format above, with each one on a separate line.
- For option 2, the system will allow you to **order** one or more books. For this option, you need to take *book title* and *no. of copies* as input from user. After you are done ordering the book, the system will ask whether you want to order another book.
- For option 3, the system will allow you to **sell** one or more books. It will ask user to enter the *ISBN* and *no. of copies* to sell book. After you are done selling book(s), the system will ask whether you want to sell another book.

For option 0, **exit** the application by breaking the loop or system exit

Assignment on Inheritance, Method overriding, Method overloading

Problem#1: A Banking System

Create a **Banking System**, where a user can **create new account**, **deposit** money, **withdraw** money and **check** the balance. For Simplicity, we will create the system for one customer. First the application will require the user to create an account. After creating the account, he should have options to **deposit**, **withdraw** money, and **check balance** as many times as he wants (until he exits the system).

When a user opens/creates an account, he has the option to open a "Saving Account" or a "Current Account". See below for the requirements of Current and Savings account.

4) **Saving account:** A saving account allows user to accumulate interest on funds he has saved for future needs. Interest rates can be compounded on a daily, weekly, monthly, or annual basis. Saving account required a minimum balance. For our purpose let's assume the **minimum balance** is 500 Tk and **interest rate** is 5% (ignore if it is daily, weekly or monthly). From savings account, user is only **allowed to withdraw a maximum amount** of money which will be set up during the account creation.

5) **Current account:** Current account offers easy access to your money for your daily transactional needs and helps keep your cash secure. You need a **trading license** to open a Current account. There is no restriction on how much money you can withdraw from Current account.

What you need to do:

- Create the **Account** class:

1. This will have ***memberName***, ***accountNumber***, and ***accountBalance*** instance variables;
2. One method name ***deposit***(double).

Note:

The ***accountNumber*** will be an auto-generated 5 digits number. So, you do not need to pass the ***accountNumber*** as a parameter in the constructor.

Code to generate 5 digit random number: (shown 3 different examples below)

i. The num variable in the examples below will store a 5 digit number in String format.

```
Random rand = new Random();
String num ="" + rand.nextInt(10) + rand.nextInt(10)+ rand.nextInt(10)+
rand.nextInt(10)+ rand.nextInt(10);
```

Or

```
Random rand = new Random();
String num = 10000 + rand.nextInt(89999) + "";
```

Or

```
String num = 10000 + (int)(Math.random()*89999) + "";
```

- Create a **SavingsAccount** class:

1. This class is a **subclass** of Account class.
2. This will have **two additional** instance variables
   1. One is "*interest*" and initialized to 5.
   2. Another instance variable for maximum withdraw amount limit.
3. Implement ***getBalance***() method. This method will add the total interest to the accountBalance value and return the value but it won't change the accountBalance value.
4. Implement ***withdraw***(double) method. This method will allow to withdraw money if the withdraw amount is less than the maximum amount limit and doesn't set the balance less than minimum balance after withdraw.

- Create a **CurrentAccount** class:

1. Should **extend** the *Account* class
2. Add an instance variable ***tradeLicenseNumber***.
3. Implement ***getBalance***() method. This method will return the accountBalance.
4. Implement ***withdraw***(double) method. This method will allow to withdraw money if the withdraw amount doesn't exceed the ***accountBalance*** value.

- Create the **application class** "**Bank**" which will have the **main** method.

First the application will require the user to create an account. So, you have to ask for user name, what type of account he wants to open and what would be the initial balance. The system will create the account (SavingsAccount or CurrentAccount object) with a randomly generated 5 digit account number.

After creating the account, you have to provide a menu on the console. It will take user input to decide what action to take.

Input '1' means deposit money. For this input, you have to ask user for the amount of money he wants to deposit. Also you need to prompt if he needs to know the balance. If yes, you need to display the balance before the deposit and after the deposit.

Input '2' means withdraw money. So, you have to ask user for the amount of money he wants to withdraw. Also you need to prompt if he needs to know the balance. If yes, you need to display the balance before the withdrawal and after the withdrawal.

Input '3' means display the balance of the account. In that case you have to display the balance.

Input '0' means exit the system.

Problem#2: Overloading

Create a **Student** Class with 3 instance variables for name, id and grade. Create 3 overloaded constructors and 2 overloaded methods.

Constructors:

1. Student(String, String, double) – this will take the name, id and grade as parameter and initialize.
2. Student(String, String) – this will take the name and id as parameter and initialize.
3. Student(String) – this will take the id as parameter and initialize.

Note: **For constructor, invoke the first constructor in the other 2. Similarly invoke/call display() in display(int) method.**

Methods:

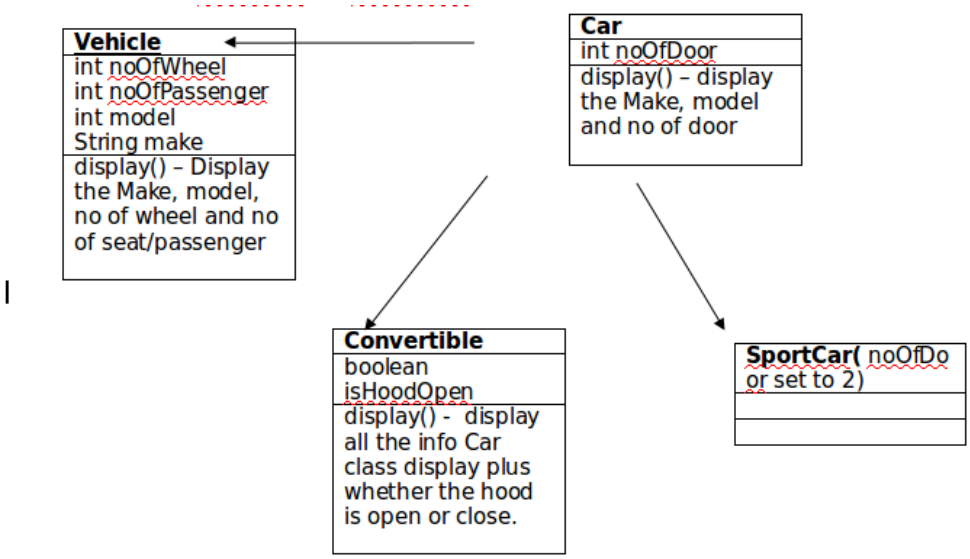
1. display() – will display the name, id, grade if they have a value.
2. display(int) – will take the current year of the student as parameter and display the name, id, grade and in which year the if they have a value.

Now implement the **application class** that will have the **main** method. From main method, you have to provide a menu on the console. It will take user input to create the object using one of the 3 constructors. **Which constructor you have to use depends on what info user has provided e.g. if user just provide id, you have to call Student(String) and so on**. Do similar logic for option 2/display.

'1' – to create a Student object.
'2' – to display the student info.

Problem#3: Overriding

Here are the 4 Java classes you need to implement. **Car** class inherits **Vehicle** class. **Convertible** and **SportCar** are descendent of **Car** class.



Now implement the **application class** that will have the **main** method. From main method, you have to provide a menu on the console. It will take user **input to create specific type [one of the 4 classes above] of object of and call the display method.**

‘1’ – to create a Vehicle object.
‘2’ – to create a Car object.
‘3’– to create a Convertible object.
‘4’– to create a SportCar object.