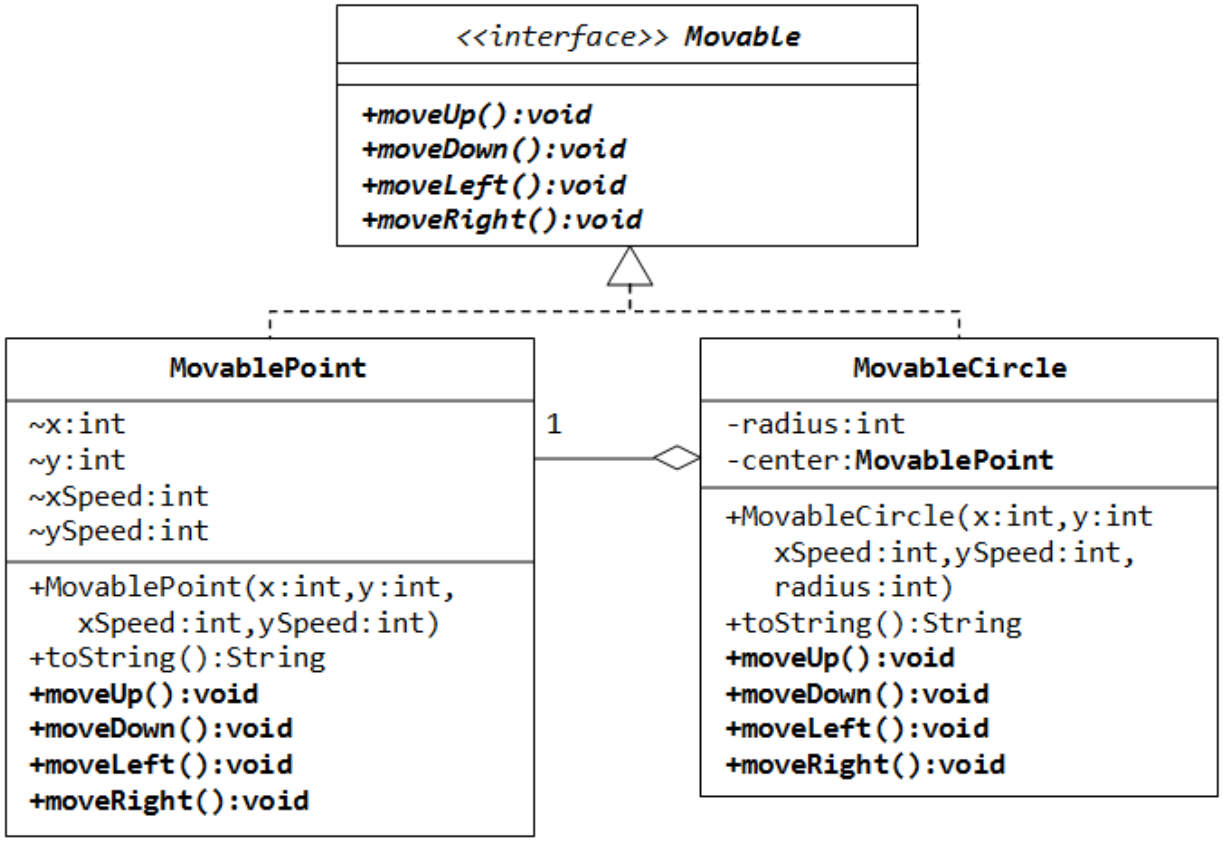
Evaluation Problems

Group 1

Suppose that we have a set of objects with some common behaviors: they could move up, down, left or right. The exact behaviors (such as how to move and how far to move) depend on the objects themselves. One common way to model these common behaviors is to define an interface called Movable, with abstract methods *moveUp()*, *moveDown()*, *moveLeft()* and *moveRight()*. The classes that implement the Movable interface will provide actual implementation to these abstract methods.



Write a test program and try out these statements:

```
Movable m1 = new MovablePoint(5, 6, 10, 15);      // upcast
System.out.println(m1);
m1.moveLeft();
System.out.println(m1);

Movable m2 = new MovableCircle(1, 2, 3, 4, 20);  // upcast
```

```
System.out.println(m2);
m2.moveRight();
System.out.println(m2);
```

Definition of Done:

a) The class definitions are defined as per the class diagram.

b) Each class definition is stored in its own .java file.

c) Base class constructors are invoked using super keyword

d) Function overriding is applied wherever applicable.

Group 2

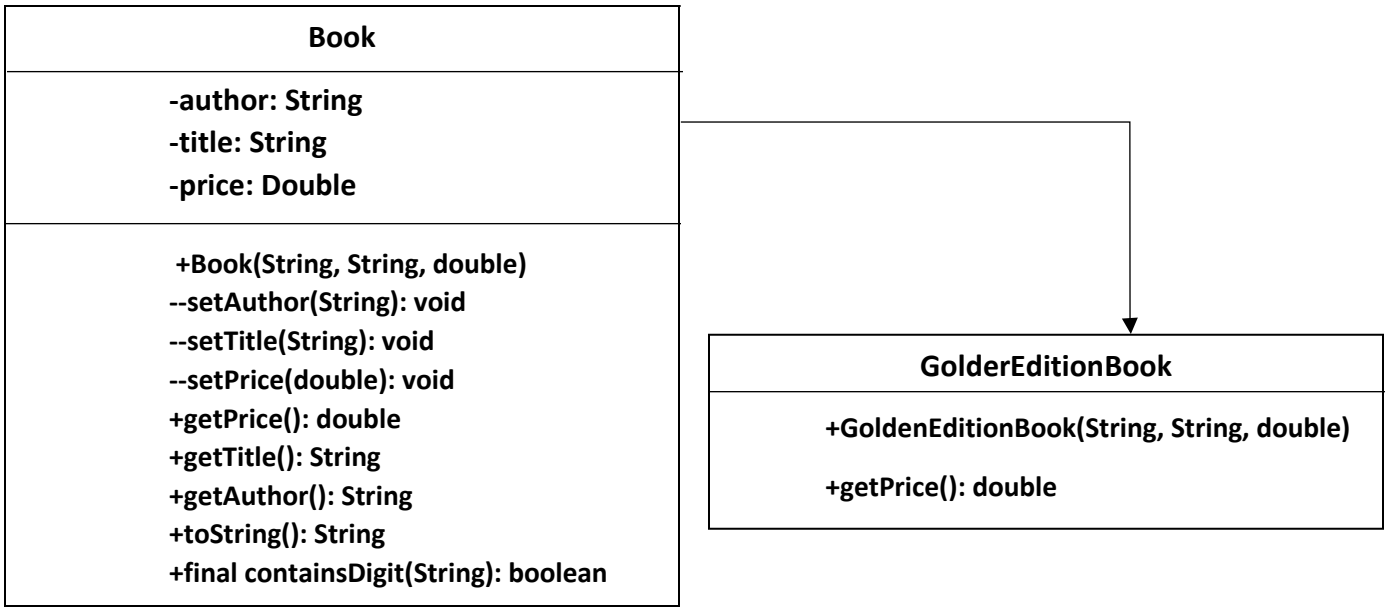
Your program should have two classes – one for the ordinary books - Book, and another for the special ones – GoldenEditionBook

Book - represents a book that holds title, author and price. A book should offer information about itself in the format shown in the output below

GoldenEditionBook - represents a special book holds the same properties as any Book, but its price is always 30% higher

Constraints:

- If the author has two names and the second name is starting with a digit- exception's message is: "Author not valid!"
- If the title's length is less than 3 symbols - exception's message is: "Title not valid!"
- If the price is zero or it is negative - exception's message is: "Price not valid!"



Definition of Done:

a) The class definitions are defined as per the class diagram.

b) Each class definition is stored in its own .java file.

c) Base class constructors are invoked using super keyword

d) Function overriding is applied wherever applicable.

Group 3:

Create a class called Rational for performing arithmetic with fractions. Write a program to test your class. Use integer variables to represent the private instance variables of the class—the numerator and the denominator. Provide a constructor that enables an object of this class to be initialized when it's declared. The constructor should store the fraction in reduced form. The fraction 2/4 is equivalent to 1/2 and would be stored in the object as 1 in the numerator and 2 in the denominator. Provide a no-argument constructor with default values in case no initializers are provided. Provide public methods that perform each of the following operations:

a) Add two Rational numbers: The result of the addition should be stored in reduced form. Implement this as a static method.

b) Subtract two Rational numbers: The result of the subtraction should be stored in reduced form. Implement this as a static method.

c) Multiply two Rational numbers: The result of the multiplication should be stored in reduced form. Implement this as a static method.

d) Divide two Rational numbers: The result of the division should be stored in reduced form. Implement this as a static method.

e) Return a String representation of a Rational number in the form a/b, where a is the numerator and b is the denominator.

f) Return a String representation of a Rational number in floating-point format. (Consider providing formatting capabilities that enable the user of the class to specify the number of digits of precision to the right of the decimal point.)

Definition of Done:

a) Two java files to be defined. One for class definition and another for the application.

b) Class definition includes a no-parameter constructor to initialize the array.

c) Class definition includes get and set methods.

d) Static methods – Add, Subtract, Multiply, Divide are defined with the desired functionalities

e) toString methods are defined with the desired functionalities

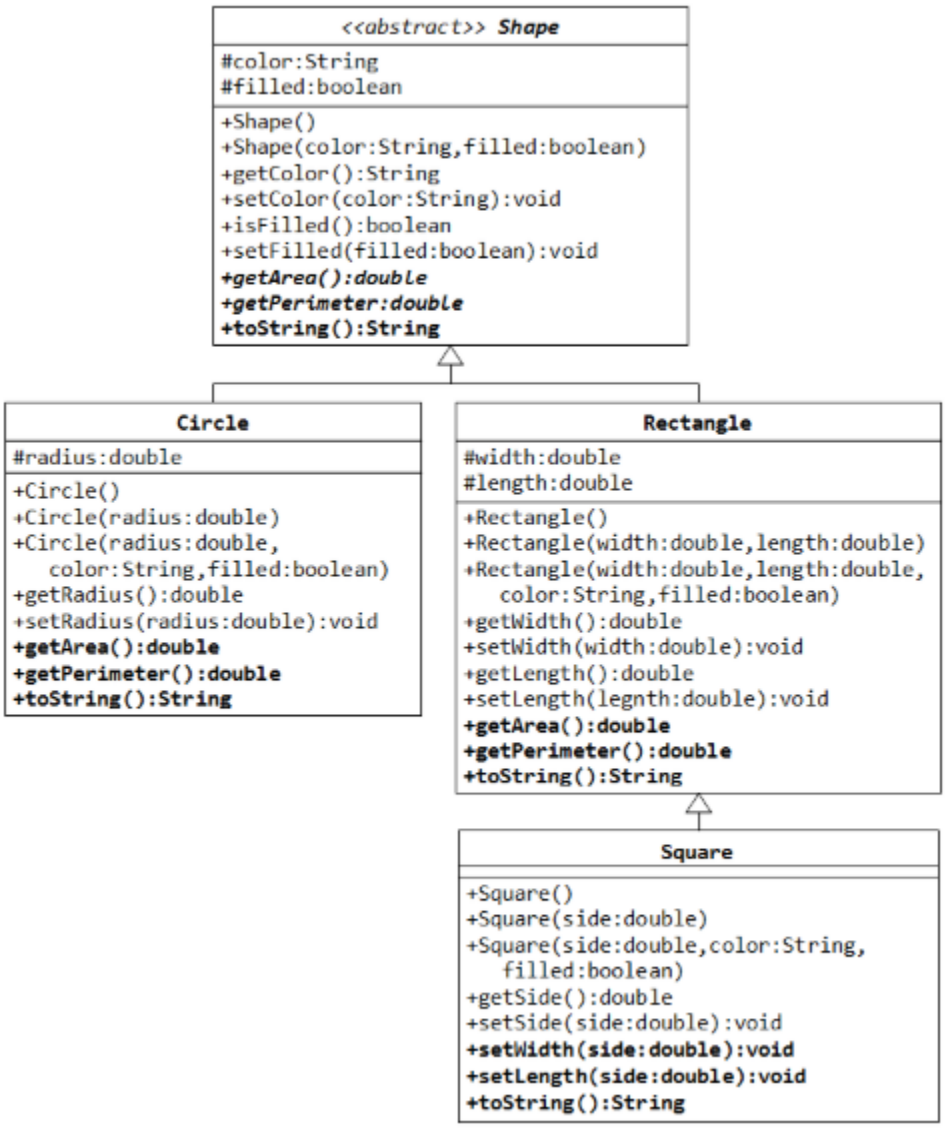
f) A menu driven application is defined that uses the above functionalities.

g) The program should display the result based on the choice.

After a successful run, the program should ask the user if he wants to try the program for some other array, and should exit only when the user wants to exit.

Group 4:

Define the class hierarchy as detailed in the following class diagram:



Definition of Done:

a) The class definitions are defined as per the class diagram.

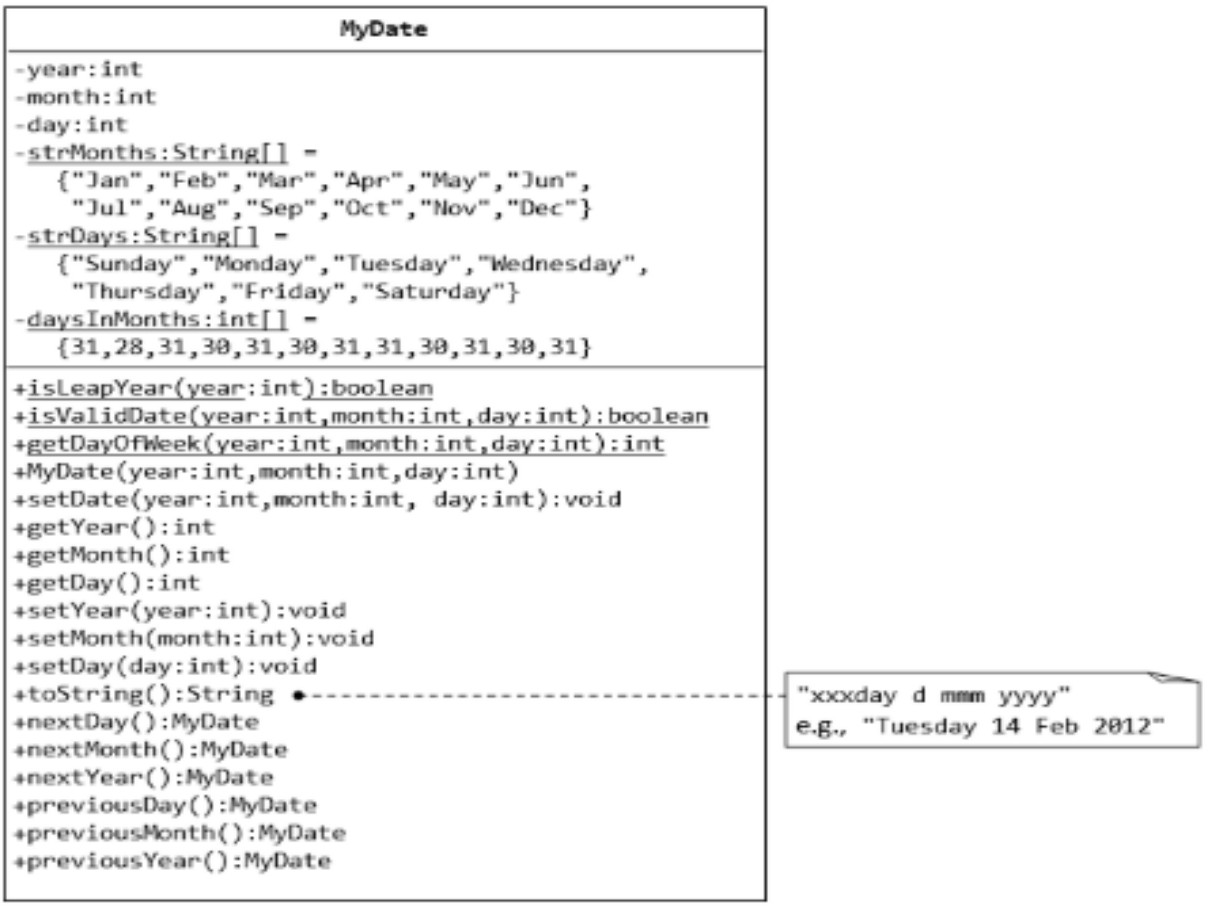
b) Each class definition is stored in its own .java file.

c) Base class constructors are invoked using super keyword

d) Function overriding is applied wherever applicable.

Group 5:

A class called MyDate, which models a date instance, is defined as shown in the class diagram.



Definition of Done:

a) Each class definition is defined in its own .java file. The permissible values of instance variables are:

year (int): Between 1 to 9999.

month (int): Between 1 (Jan) to 12 (Dec).

day (int): Between 1 to 28|29|30|31, where the last day depends on the month and whether it is a leap year for Feb (28|29).

The values are validated and proper exceptions are thrown if the value entered is not within the permissible values.

b) setDate(int year, int month, int day): It should invoke the static method isValidDate() to verify that the given year, month and day constitute a valid date. Otherwise, it shall throw an IllegalArgumentException with the message "Invalid year, month, or day!".

c) setYear(int year): It should verify that the given year is between 1 and 9999. Otherwise, it shall throw an IllegalArgumentException with the message "Invalid year!".

d) setMonth(int month): It should verify that the given month is between 1 and 12. Otherwise, it shall throw an IllegalArgumentException with the message "Invalid month!".

e) setDay(int day): It should verify that the given day is between 1 and dayMax, where dayMax depends on the month and whether it is a leap year for Feb. Otherwise, it should throw an IllegalArgumentException with the message "Invalid month!".)

f) nextDay(): update this instance to the next day and return this instance. Take note that nextDay() for 31 Dec 2000 shall be 1 Jan 2001.

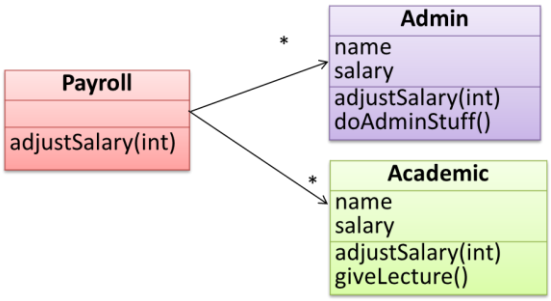
g) nextMonth(): update this instance to the next month and return this instance. Take note that nextMonth() for 31 Oct 2012 shall be 30 Nov 2012.

h) nextYear(): update this instance to the next year and return this instance. Take note that nextYear() for 29 Feb 2012 shall be 28 Feb 2013.Throw an IllegalStateException with the message "Year out of range!" if year > 9999.) previousDay(), previousMonth(), previousYear(): similar to the above.

i) Remaining functions have obvious meanings

Group 6:

Write a payroll application for a university to facilitate payroll processing of university staff. Suppose an *adjustSalary(int)* operation adjusts the salaries of all staff members. This operation will be executed whenever the university initiates a salary adjustment for its staff. However, the adjustment formula is different for different staff categories, say admin and academic. Here is one possible way of designing the classes in the Payroll system



Definition of Done:

a) The class definitions are defined as per the class diagram. Class Payroll is abstract

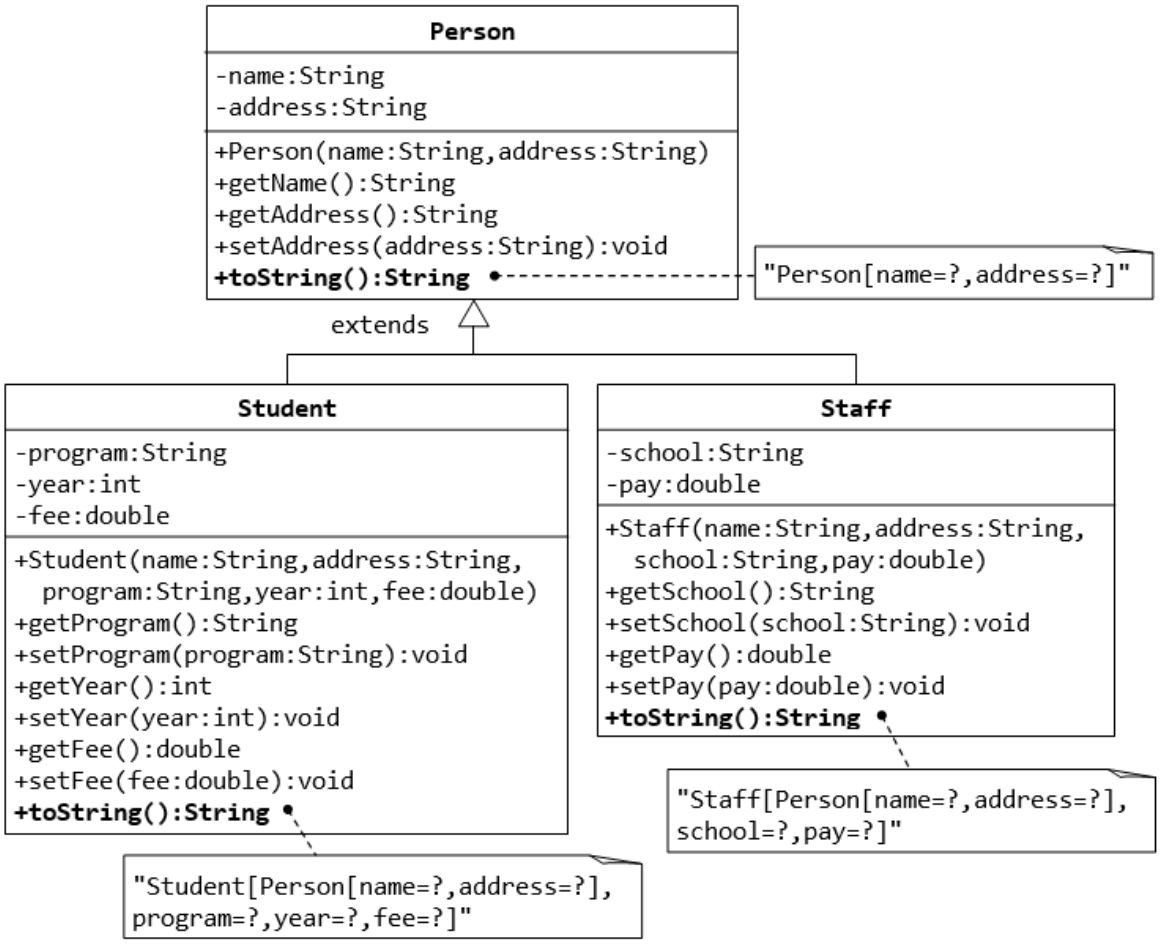
b) Each class definition is stored in its own .java file.

c) Base class constructors are invoked using super keyword

d) Function overriding is applied wherever applicable.

Group 7

Consider the classes defined below:



Definition of Done:

a. The class definitions are defined as per the class diagram
b. Each class definition is stored in its own .java file
c. Function overriding is applied wherever applicable