Labsheet 6: Classes and Objects

Question 1

Define a class called Pizza which uses the default constructor. The pizza class should contain three private attributes:

topping: a String describing the pizza topping
diameter: an integer describing the diameter in inches
price: a double that describes the price in dollars and cents

The pizza class should contain 6 methods:

setTopping: takes a String argument and sets the topping to the argument
setDiameter: takes an integer argument and sets the diameter to the argument
setPrice: takes a double argument and sets the price to the argument
getTopping: returns a String argument with the value of the topping
getDiameter: returns an integer argument with the value of the diameter
getPrice: returns a double argument with the value of the price

Write a program called TestPizza to construct two instances of Pizza. The instances are pizza1 and pizza2. The algorithm is as follows:

For each of the two instances of the Pizza class:

- Create an instance of the Pizza class
- Set the topping
- Set the diameter
- Set the price
- Output the result using the selector methods

Question 2

Define a class called Circle and includes a constructor. The circle class should contain three private attributes:

radius: an double that holds the radius of the circle
diameter: an double that represents the diameter of the circle
area: a double calculated by the pi * radius squared

The circle class should contain 5 methods:
Circle: The constructor that creates a circle with radius = 1
setRadius: takes an double argument and sets the radius to the argument
getRadius: returns an double argument with the value of the radius
computeDiameter: calculates the diameter and returns the value of the diameter
computeArea: calculates the area and returns the value of the area
The following calculations are used:
diameter = 2 * radius
area = 3.14 * radius * radius (pi*(r)squared)

Write a program called TestCircle that create three instances of Circle. The instances are circle1, circle2, and circle3. The algorithm is as follows:

For each of the three instances of the Circle class:

- Create an instance of the Circle class
- For one, set the radius < 1. For another, set the radius > 1. For another, accept the default radius of 1.
- Calculate the diameter and area
- Output the results

Question 3

Create a class named Checkup with fields that hold a patient number, two blood pressure figures (systolic and diastolic), and two cholesterol figures (LDL, and HDL). Include methods to get and set each of the fields. Include a method named computeRatio() that divides LDL cholesterol by HDL cholesterol and displays the result. Include an additional method named ExplainRatio() that explains that LDL is known as "good cholesterol" and that a ratio of 3.5 or lower is considered optimum. Save this class as Checkup.java.

Create a class named TestCheckup whose main() method declares four Checkup objects. Provide values for each field for each patient. Then display the values. Blood pressure numbers are usually displayed with a slash between the systolic and diastolic values. (Typical numbers are values like 110/78 or 130/90.) With the cholesterol figures, display the explanation of the cholesterol ratio calculation.

Question 4

Create a class called Invoice that a hardware store might use to represent an invoice for an item sold at the store. An Invoice should include four pieces of information as instance variables - a part number(type String),a part description(type String),a quantity of the item being purchased (type int) and a price per item (double). Your class should have a constructor that initializes the four instance variables. Provide a set and a get method for each instance variable. In addition, provide a method named getInvoiceAmount that calculates the invoice amount (i.e., multiplies the quantity by the price per item), then returns the amount as a double value. If the quantity is not positive, it should be set to 0. If the price per item is not positive, it should be set to 0.0. Write a test application named InvoiceTest that demonstrates class Invoice's capabilities.

Question 5

Create a class called Employee that includes three pieces of information as instance variables—a first name (type String), a last name (type String) and a monthly salary (double). Your class should have a constructor that initializes the three instance variables. Provide a set and a get method for each instance variable. If the monthly salary is not positive, set it to 0.0. Write a test application named EmployeeTest that demonstrates class Employee's capabilities. Create two Employee objects and display each object's yearly salary. Then give each Employee a 10% raise and display each Employee's yearly salary again.

Question 6

Create a class called Book to represent a book. A Book should include four pieces of information as instance variables-a book name, an ISBN number, an author name and a publisher. Your class should have a constructor that initializes the four instance variables. Provide a mutator method and accessor method (query method) for each instance variable. Inaddition, provide a method named getBookInfo that returns the description of the book as a String (the description should include all the information about the book). You should use this keyword in member methods and constructor.

Write a test application named BookTest to create an array of object for 30 elements for class Book to demonstrate the class Book's capabilities.

Question 7

Modify class **Account** (in the example below) to provide a method called debit that withdraws money from an Account. Ensure that the debit amount does not exceed the Account's balance. If it does, the balance should be left unchanged and the method should print a message indicating — Debit amount exceeded account balance. Modify class **AccountTest** (in the example below) to test method debit.

```
//filename: Account.java
// Account class
public class Account {
private double balance;
public Account(double initialBalance) {
if (initialBalance > 0.0) balance=initialBalance;
}
public void credit(double amount){
balance=balance+amount;
}
public void debit(double amount){
balance=balance-amount;
}
public double getBalance(){
return balance;
}
}
```

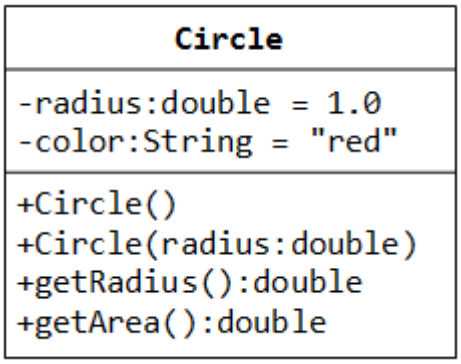
```
//filename: AccountTest.java
// Account testing class with the main() method
import java.util.Scanner;
public class AccountTest {
public static void main (String args[]){
Account account1 = new Account (50.00);
Account account2 = new Account (-7.53);
System.out.printf("Account1 Balance: $%.2f\n", account1.getBalance());
System.out.printf("Account2 Balance: $%.2f\n\n", account2.getBalance());
Scanner input = new Scanner( System.in );
double depositAmount;
double debitAmount;
System.out.print( "Enter deposit amount for account1: " ); // prompt
```

```
depositAmount = input.nextDouble(); // obtain user input
System.out.printf( "\nadding %.2f to account1 balance\n\n", depositAmount
);
account1.credit( depositAmount ); // add to account1 balance
// display balances
System.out.printf( "Account1 balance: $%.2f\n", account1.getBalance() );
System.out.printf( "Account2 balance: $%.2f\n\n", account2.getBalance() );
System.out.print( "Enter deposit amount for account2: " ); // prompt
depositAmount = input.nextDouble(); // obtain user input
System.out.printf( "\nAdding %.2f to account2 balance\n\n", depositAmount
);
account2.credit( depositAmount ); // add to account2 balance
// display balances
System.out.printf( "Account1 balance: $%.2f\n", account1.getBalance() );
System.out.printf( "Account2 balance: $%.2f\n", account2.getBalance() );
System.out.print( "Enter debit amount for account1: " );
debitAmount = input.nextDouble();
System.out.printf( "\nSubtracting %.2f from account1 balance\n\n",
debitAmount );
if (account1.getBalance()>=debitAmount) {
account1.debit( debitAmount );
System.out.printf( "Account1 balance: $%.2f\n", account1.getBalance() );
System.out.printf( "Account2 balance: $%.2f\n\n", account2.getBalance() );
}
else {
System.out.printf("!!! Debit amount exceeded account balance!!!\n\n");
}
// display balances
System.out.print( "Enter debit amount for account2: " );
debitAmount = input.nextDouble();
System.out.printf( "\nSubtracting %.2f from account2 balance\n\n",
debitAmount );
if (account1.getBalance()>=debitAmount) {
account1.debit( debitAmount );
System.out.printf( "Account1 balance: $%.2f\n", account1.getBalance() );
System.out.printf( "Account2 balance: $%.2f\n\n", account2.getBalance() );
}
else {
System.out.printf("!!!Debit amount exceeded account balance!!!\n\n");
}
}
}
```

Question 8

A class called **`circle`** is designed as shown in the following class diagram. It contains:

- Two `private` instance variables: `radius` (of type `double`) and `color` (of type `String`), with default value of 1.0 and "red", respectively.
- Two *overloaded* constructors;
- Two `public` methods: `getRadius()` and `getArea()`.



The source codes for `Circle` is as follows:

```
public class Circle {           // save as "Circle.java"
   // private instance variable, not accessible from outside this class
   private double radius;
   private String color;

   // 1st constructor, which sets both radius and color to default
   public Circle() {
      radius = 1.0;
      color = "red";
   }

   // 2nd constructor with given radius, but color default
   public Circle(double r) {
      radius = r;
      color = "red";
   }

   // A public method for retrieving the radius
   public double getRadius() {
     return radius;
   }

   // A public method for computing the area of circle
   public double getArea() {
      return radius*radius*Math.PI;
   }
}
```

Compile "`Circle.java`". Can you run the `Circle` class? Why? This `Circle` class does not have a `main()` method. Hence, it cannot be run directly. This `Circle` class is a "building block" and is meant to be used in another program.

Let us write a *test program* called `TestCircle` which uses the `Circle` class, as follows:

```
public class TestCircle {        // save as "TestCircle.java"
   public static void main(String[] args) {
      // Declare and allocate an instance of class Circle called c1
```

```
      //  with default radius and color
      Circle c1 = new Circle();
      // Use the dot operator to invoke methods of instance c1.
      System.out.println("The circle has radius of "
         + c1.getRadius() + " and area of " + c1.getArea());

      // Declare and allocate an instance of class circle called c2
      //  with the given radius and default color
      Circle c2 = new Circle(2.0);
      // Use the dot operator to invoke methods of instance c2.
      System.out.println("The circle has radius of "
         + c2.getRadius() + " and area of " + c2.getArea());
   }
}
```

Now, run the `TestCircle` and study the results.

TRY:

1. Constructor: Modify the class `Circle` to include a third constructor for constructing a `Circle` instance with the given `radius` and `color`.

   ```
   // Construtor to construct a new instance of Circle with the given
   radius and color
   public Circle (double r, String c) {......}
   ```

   Modify the test program `TestCircle` to construct an instance of `Circle` using this constructor.

2. Getter: Add a getter for variable `color` for retrieving the `color` of a `Circle` instance.

   ```
   // Getter for instance variable color
   public String getColor() {......}
   ```

   Modify the test program to test this method.

3. public vs. private: In `TestCircle`, can you access the instance variable `radius` directly (e.g., `System.out.println(c1.radius)`); or assign a new value to `radius` (e.g., `c1.radius=5.0`)? Try it out and explain the error messages.
4. Setter: Is there a need to change the values of `radius` and `color` of a `Circle` instance after it is constructed? If so, add two `public` methods called *setters* for changing the `radius` and `color` of a `Circle` instance as follows:

   ```
   // Setter for instance variable radius
   public void setRadius(double r) {
      radius = r;
   }

   // Setter for instance variable color
   public void setColor(String c) { ..... }
   ```

Modify the `TestCircle` to test these methods, e.g.,

```
Circle c3 = new Circle();   // construct an instance of Circle
c3.setRadius(5.0);          // change radius
c3.setColor(...);           // change color
```

5. Keyword "this": Instead of using variable names such as `r` (for `radius`) and `c` (for `color`) in the methods' arguments, it is better to use variable names `radius` (for `radius`) and `color` (for `color`) and use the special keyword "`this`" to resolve the conflict between instance variables and methods' arguments. For example,

```
// Instance variable
private double radius;

// Setter of radius
public void setRadius(double radius) {
   this.radius = radius; // "this.radius" refers to the instance
variable
                         // "radius" refers to the method's argument
}
```

   Modify ALL the constructors and setters in the `Circle` class to use the keyword "`this`".

6. Method toString(): Every well-designed Java class should contain a `public` method called `toString()` that returns a short description of the instance (in a return type of `String`). The `toString()` method can be called explicitly (via *`instanceName`*`.toString()`) just like any other method; or implicitly through `println()`. If an instance is passed to the `println(`*`anInstance`*`)` method, the `toString()` method of that instance will be invoked implicitly. For example, include the following `toString()` methods to the `Circle` class:

```
public String toString() {
   return "Circle: radius=" + radius + " color=" + color;
}
```

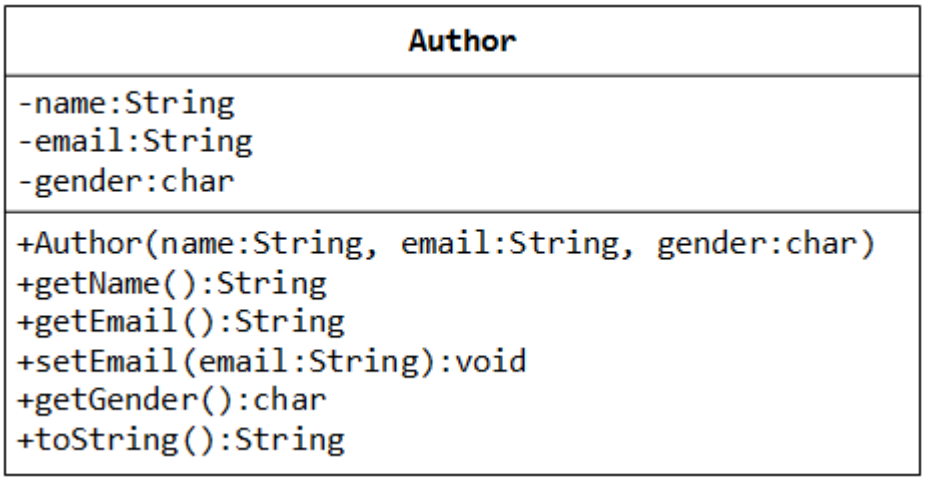
   Try calling `toString()` method explicitly, just like any other method:

```
Circle c1 = new Circle(5.0);
System.out.println(c1.toString());   // explicit call
```

   `toString()` is called implicitly when an instance is passed to `println()` method, for example,

```
Circle c2 = new Circle(1.2);
System.out.println(c2.toString());  // explicit call
System.out.println(c2);             // println() calls toString()
implicitly, same as above
System.out.println("Operator '+' invokes toString() too: " + c2);  //
'+' invokes toString() too
```

Question 9



A class called `Author` is designed as shown in the class diagram. It contains:

- Three `private` instance variables: `name` (`String`), `email` (`String`), and `gender` (`char` of either `'m'` or `'f'`);
- One constructor to initialize the `name`, `email` and `gender` with the given values;

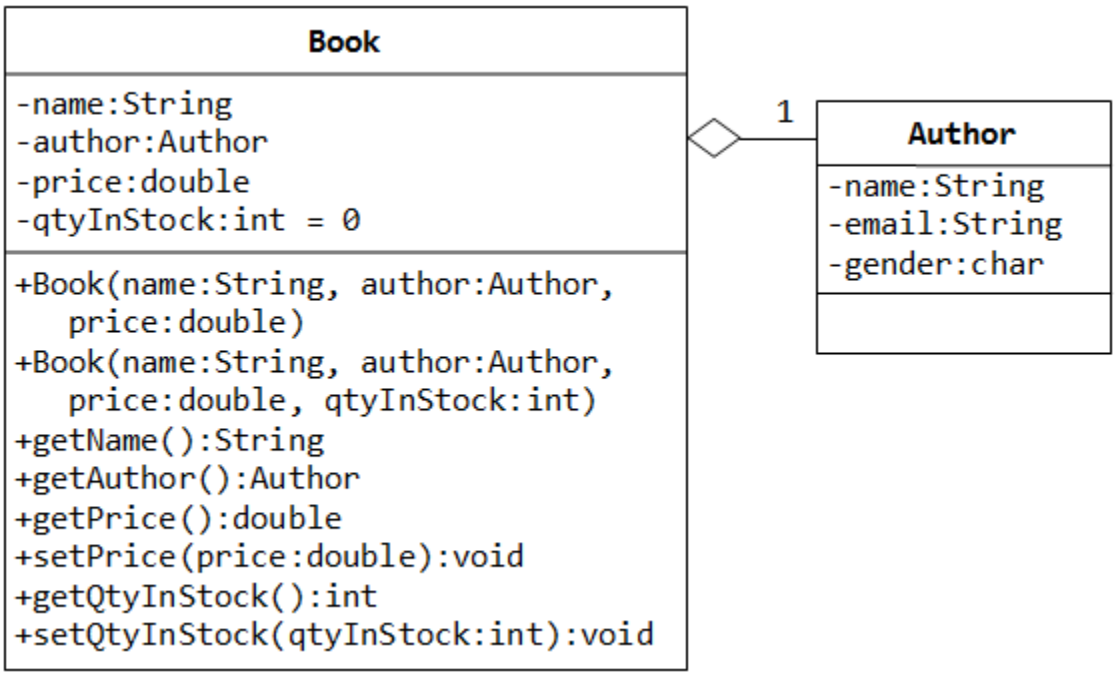
  ```
  public Author (String name, String email, char gender) {......}
  ```

  (There is no default constructor for Author, as there are no defaults for name, email and gender.)

- `public` getters/setters: `getName()`, `getEmail()`, `setEmail()`, and `getGender()`; (There are no setters for `name` and `gender`, as these attributes cannot be changed.)
- A `toString()` method that returns "*author-name (gender) at email*", e.g., "Tan Ah Teck (m) at ahTeck@somewhere.com".

Write the `Author` class. Also write a *test program* called `TestAuthor` to test the constructor and `public` methods. Try changing the `email` of an author, e.g.,

```
Author anAuthor = new Author("Tan Ah Teck", "ahteck@somewhere.com", 'm');
System.out.println(anAuthor);   // call toString()
anAuthor.setEmail("paul@nowhere.com")
System.out.println(anAuthor);
```



A class called `Book` is designed as shown in the class diagram. It contains:

- Four `private` instance variables: `name` (`String`), `author` (of the class `Author` you have just created, assume that each book has one and only one author), `price` (`double`), and `qtyInStock` (`int`);
- Two constructors:
- `public Book (String name, Author author, double price) {...}`
- `public Book (String name, Author author, double price, int qtyInStock) {...}`

- public methods `getName()`, `getAuthor()`, `getPrice()`, `setPrice()`, `getQtyInStock()`, `setQtyInStock()`.
- `toString()` that returns "*'book-name' by author-name (gender) at email*". (Take note that the `Author`'s `toString()` method returns "*author-name (gender) at email*".)

Write the class `Book` (which uses the `Author` class written earlier). Also write a test program called `TestBook` to test the constructor and `public` methods in the class `Book`. Take Note that you have to construct an instance of `Author` before you can construct an instance of `Book`. E.g.,

```
Author anAuthor = new Author(......);
Book aBook = new Book("Java for dummy", anAuthor, 19.95, 1000);
// Use an anonymous instance of Author
Book anotherBook = new Book("more Java for dummy", new Author(......),
29.95, 888);
```

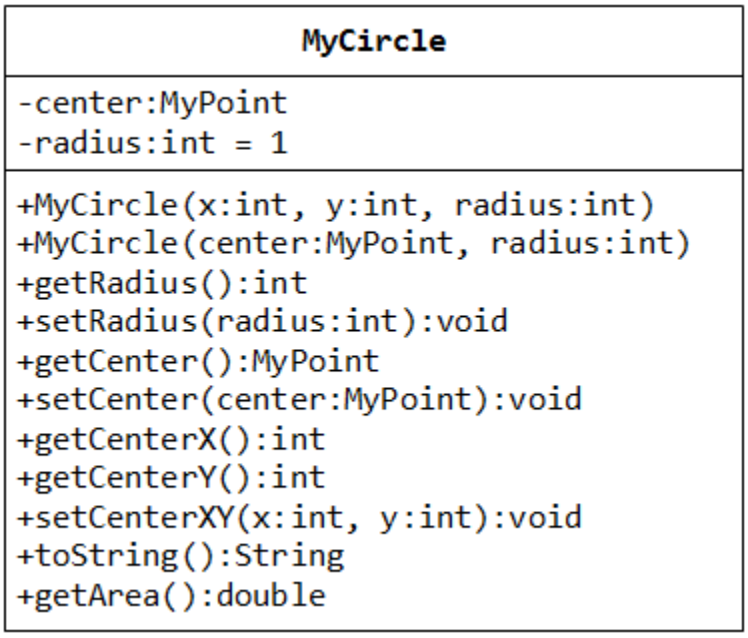
Take note that both `Book` and `Author` classes have a variable called `name`. However, it can be differentiated via the referencing instance. For a `Book` instance says `aBook`, `aBook.name` refers to the `name` of the book; whereas for an `Author`'s instance say `auAuthor`, `anAuthor.name` refers to the `name` of the author. There is no need (and not recommended) to call the variables `bookName` and `authorName`.

TRY:

1. Printing the `name` and `email` of the author from a `Book` instance. (Hint: `aBook.getAuthor().getName()`, `aBook.getAuthor().getEmail()`).
2. Introduce new methods called `getAuthorName()`, `getAuthorEmail()`, `getAuthorGender()` in the `Book` class to return the `name`, `email` and `gender` of the author of the book. For example,

   ```
   public String getAuthorName() { ..... }
   ```

Question 10



A class called `MyCircle`, which models a circle with a `center` (x, y) and a `radius`, is designed as shown in the class diagram. The `MyCircle` class uses an instance of `MyPoint` class (created in the previous exercise) as its `center`.

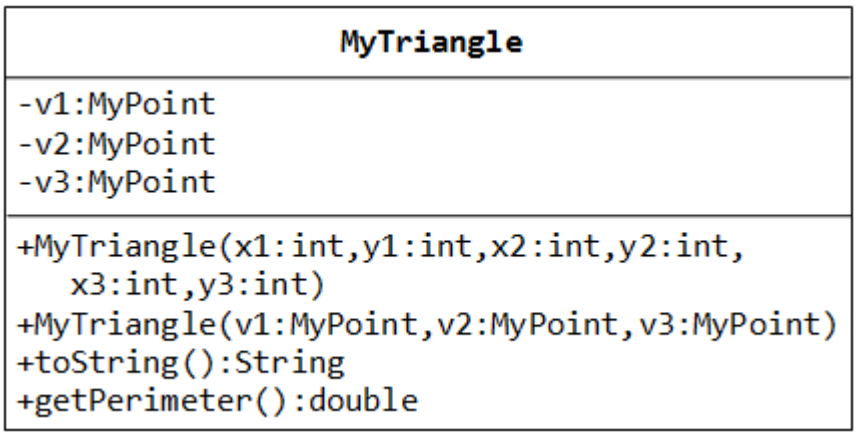
The class contains:

- Two `private` instance variables: `center` (an instance of `MyPoint`) and `radius` (`int`).
- A constructor that constructs a circle with the given center's (`x`, `y`) and `radius`.

- An overloaded constructor that constructs a `MyCircle` given a `MyPoint` instance as `center`, and `radius`.
- Various getters and setters.
- A `toString()` method that returns a string description of this instance in the format "Circle @ (*x*, *y*) radius=*r*".
- A `getArea()` method that returns the area of the circle in `double`.

Write the `MyCircle` class. Also write a test program (called `TestMyCircle`) to test all the methods defined in the class.

Question 11



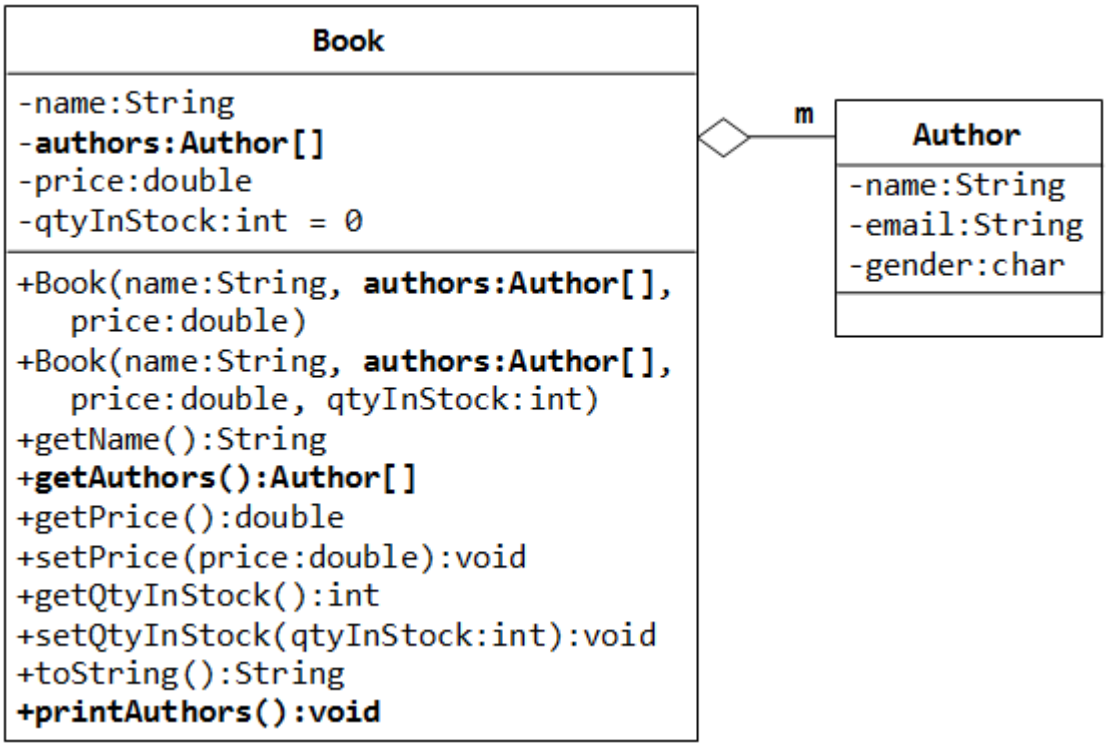
A class called `MyTriangle`, which models a triangle with 3 vertices, is designed as follows. The `MyTriangle` class uses three `MyPoint` instances (created in the earlier exercise) as the three vertices.

The class contains:

- Three `private` instance variables `v1`, `v2`, `v3` (instances of `MyPoint`), for the three vertices.
- A constructor that constructs a `MyTriangle` with three points `v1=(x1, y1)`, `v2=(x2, y2)`, `v3=(x3, y3)`.
- An overloaded constructor that constructs a `MyTriangle` given three instances of `MyPoint`.
- A `toString()` method that returns a string description of the instance in the format "Triangle @ (x1, y1), (x2, y2), (x3, y3)".
- A `getPerimeter()` method that returns the length of the perimeter in double. You should use the `distance()` method of `MyPoint` to compute the perimeter.
- A method `printType()`, which prints "equilateral" if all the three sides are equal, "isosceles" if any two of the three sides are equal, or "scalene" if the three sides are different.

Write the `MyTriangle` class. Also write a test program (called `TestMyTriangle`) to test all the methods defined in the class.

Question 12



In question 9, a book is written by one and only one author. In reality, a book can be written by one or more author. Modify the `Book` class to support one or more authors by changing the instance variable `authors` to an `Author` array. Reuse the Author class written earlier.

Notes:

- The constructors take an array of `Author` (i.e., `Author[]`), instead of an `Author` instance.
- The `toString()` method shall return "book-name by *n* authors", where *n* is the number of authors.
- A new method `printAuthors()` to print the names of all the authors.

You are required to:

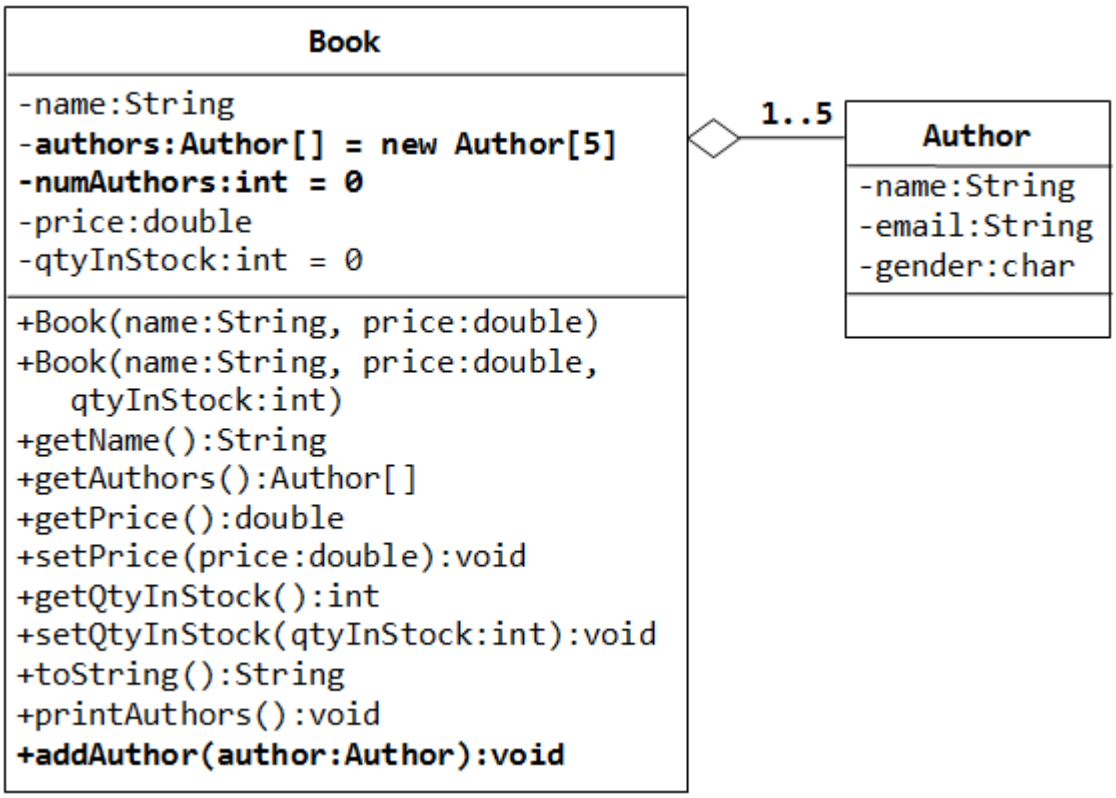
1. Write the code for the `Book` class. You shall re-use the `Author` class written earlier.
2. Write a test program (called `TestBook`) to test the `Book` class.

Hints:

```
// Declare and allocate an array of Authors
Author[] authors = new Author[2];
authors[0] = new Author("Tan Ah Teck", "AhTeck@somewhere.com", 'm');
authors[1] = new Author("Paul Tan", "Paul@nowhere.com", 'm');

// Declare and allocate a Book instance
Book javaDummy = new Book("Java for Dummy", authors, 19.99, 99);
System.out.println(javaDummy);  // toString()
System.out.print("The authors are: ");
javaDummy.printAuthors();
```

Question 13



In the above exercise, the number of authors cannot be changed once a `Book` instance is constructed. Suppose that we wish to allow the user to add more authors (which is really unusual but presented here for academic purpose).

We shall remove the `authors` from the constructors, and add a new method called `addAuthor()` to add the given `Author` instance to this `Book`.

We also need to pre-allocate an `Author` array, with a fixed length (says 5 - a book is written by 1 to 5 authors), and use another instance variable `numAuthors` (`int`) to keep track of the actual number of authors.

You are required to:

1. Modify your `Book` class to support this new requirement.
   Hints:

```
public class Book {
   // private instance variable
   private Author[] authors = new Author[5]; // declare and allocate
the array
                                             // BUT not the element's
instance
   private int numAuthors = 0;

   ......
   ......

   public void addAuthor(Author author) {
      authors[numAuthors] = author;
      ++numAuthors;
   }
}

// Test program
Book javaDummy = new Book("Java for Dummy", 19.99, 99);
System.out.println(javaDummy);  // toString()
System.out.print("The authors are: ");
javaDummy.printAuthors();

javaDummy.addAuthor(new Author("Tan Ah Teck", "AhTeck@somewhere.com",
'm'));
javaDummy.addAuthor(new Author("Paul Tan", "Paul@nowhere.com", 'm'));
System.out.println(javaDummy);  // toString()
System.out.print("The authors are: ");
      javaDummy.printAuthors();
```

2. Try writing a method called `removeAuthorByName(authorName)`, that remove the author from this `Book` instance if `authorName` is present. The method shall return `true` if it succeeds.

```
boolean removeAuthorByName(String authorName)
```

Advanced Note: Instead of using a fixed-length array in this case, it is better to be a dynamically allocated array (e.g., `ArrayList`), which does not have a fixed length.

Question 14

Create a class called Book to represent a book. A Book should include four pieces of information as instance variables - a book name, an ISBN number, an author name and a price. Your class should have a constructor that initializes the four instance variables. Provide a mutator method and accessor method (query method) for each instance

variable. In addition, provide a method named getBookInfo that returns the description of the book as a String (the description should include all the information about the book). You should use this keyword in member methods and constructor. Write a test application named BookTest to create an array of object for 5 elements for class Book, display the details of each book and calculate the total price for all the 5 books.