Lab 4. OOP Exercises

Writing Good Programs

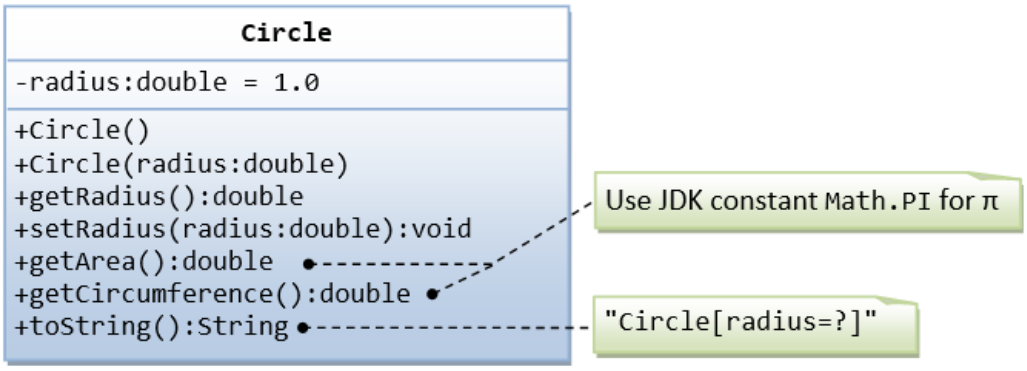
The only way to learn programming is program, program and program. Learning programming is like learning cycling, swimming or any other sports. You can't learn by watching or reading books. Start to program immediately. On the other hands, to improve your programming, you need to read many books and study how the masters program.
It is easy to write programs that work. It is much harder to write programs that not only work but also easy to maintain and understood by others – I call these good programs. In the real world, writing program is not meaningful. You have to write good programs, so that others can understand and maintain your programs.
Pay particular attention to:

1. **Coding style**:
   - Read Java code convention: "Google Java Style Guide" or "Java Code Conventions - Oracle".
   - Follow the Java Naming Conventions for variables, methods, and classes STRICTLY. Use CamelCase for names. Variable and method names begin with lowercase, while class names begin with uppercase. Use nouns for variables (e.g., radius) and class names (e.g., Circle). Use verbs for methods (e.g., getArea(), isEmpty()).
   - **Use Meaningful Names**: Do not use names like a, b, c, d, x, x1, x2, and x1688 - they are meaningless. Avoid single-alphabet names like i, j, k. They are easy to type, but usually meaningless. Use single-alphabet names only when their meaning is clear, e.g., x, y, z for co-ordinates and i for array index. Use meaningful names like row and col (instead of x and y, i and j, x1 and x2), numStudents (not n), maxGrade, size (not n), and upperbound (not n again). Differentiate between singular and plural nouns (e.g., use books for an array of books, and book for each item).
   - Use consistent indentation and coding style. Many IDEs (such as Eclipse / NetBeans) can re-format your source codes with a single click.
2. **Program Documentation**: Comment! Comment! and more Comment to explain your code to other people and to yourself three days later.
3. The problems in this tutorial are certainly NOT challenging. There are tens of thousands of challenging problems available – used in training for various programming contests (such as International Collegiate Programming Contest (ICPC), International Olympiad in Informatics (IOI)).

1 Exercises on Classes

1.1 An Introduction to Classes and Instances by Example - The Circle Class

This first exercise shall lead you through all the basic concepts in OOP.



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

- Two private instance variables: *radius* (of the type *double*) and *color* (of the type *String*), with default value of 1.0 and "red", respectively.
- Two *overloaded* constructors - a *default* constructor with no argument, and a constructor which takes a *double* argument for *radius*.
- Two public methods: *getRadius()* and *getArea()*, which return the radius and area of this instance, respectively.

The source codes for Circle.java is as follows:

```java
/**
 * The Circle class models a circle with a radius and color.
 */
public class Circle {  // Save as "Circle.java"

   // private instance variable, not accessible from outside this class
   private double radius;
   private String color;

   // Constructors (overloaded)
   /**
    * Constructs a Circle instance with default value for radius and color
    */
   public Circle() {  // 1st (default) constructor
```

```
15        radius = 1.0;
          color = "red";
17      }

19      /**
         * Constructs a Circle instance with the given radius and default color
21       */
        public Circle(double r) {  // 2nd constructor
23        radius = r;
          color = "red";
25      }

27      /**
         * Returns the radius
29       */
        public double getRadius() {
31        return radius;
        }
33
        /**
35       * Returns the area of this Circle instance
         */
37      public double getArea() {
          return radius*radius*Math.PI;
39      }
    }
```

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** (in another source file called TestCircle.java) which uses the Circle class, as follows:

```
 /**
2 *  A Test Driver for the Circle class
  */
4 public class TestCircle {  // Save as "TestCircle.java"
    public static void main(String[] args) {
6     // Declare an instance of Circle class called circle1.
      // Construct the instance circle1 by invoking the "default" constructor
8     // which sets its radius and color to their default value.
      Circle circle1 = new Circle();
10    // Invoke public methods on instance circle1, via dot operator.
      System.out.println("The circle has radius of "
12      + circle1.getRadius() + " and area of " + circle1.getArea());
      //The circle has radius of 1.0 and area of 3.141592653589793
```

```
14
      // Declare an instance of class circle called circle2.
16    // Construct the instance circle2 by invoking the second constructor
      // with the given radius and default color.
18    Circle circle2 = new Circle(2.0);
      // Invoke public methods on instance circle2, via dot operator.
20    System.out.println("The circle has radius of "
        + circle2.getRadius() + " and area of " + circle2.getArea());
22    // The circle has radius of 2.0 and area of 12.566370614359172
   }
24 }
```

Now, run the TestCircle and study the results.

More Basic OOP Concepts

1. **Constructor**: Modify the class Circle to include a third constructor for constructing a Circle instance with two arguments - a *double* for *radius* and a *String* for *color*.

   ```
   // 3rd constructor to construct a new instance of Circle with the given radius and color
2  public Circle(double r, String c) {
     ......
4  }
   ```

   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 this instance.

   ```
   // Getter for instance variable color
2  public String getColor() {
     ......
4  }
   ```

   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(circle1.radius)*); or assign a new value to *radius* (e.g., *circle1.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:

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

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

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

```java
Circle circle4 = new Circle();  // construct an instance of Circle
circle4.setRadius(5.5);         // change radius
System.out.println("radius is: " + circle4.getRadius());  // Print
    ↪ radius via getter

circle4.setColor("green");       // Change color
System.out.println("color is: " + circle4.getColor());    // Print
    ↪ color via getter

// You cannot do the following because setRadius() returns void,
// which cannot be printed
System.out.println(circle4.setRadius(4.4));
```

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,

```java
// Instance variable
private double radius;

/**
 * Constructs a Circle instance with the given radius and default color
 */
public Circle(double radius) {
   this.radius = radius;    // "this.radius" refers to the instance variable
   // "radius" refers to the method's parameter
   color = "red";
}

/**
 * Sets the radius to the given value
```

```
   */
16 public void setRadius(double radius) {
     this.radius = radius;    // "this.radius" refers to the instance variable
18   // "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 description of the instance (in the 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:

```
1 /**
   * Return a self-descriptive string of this instance in the
3  * form of Circle[radius = ?, color = ?]
   */
5 public String toString() {
     return "Circle[radius = " + radius + " color = " + color + "]";
7 }
```

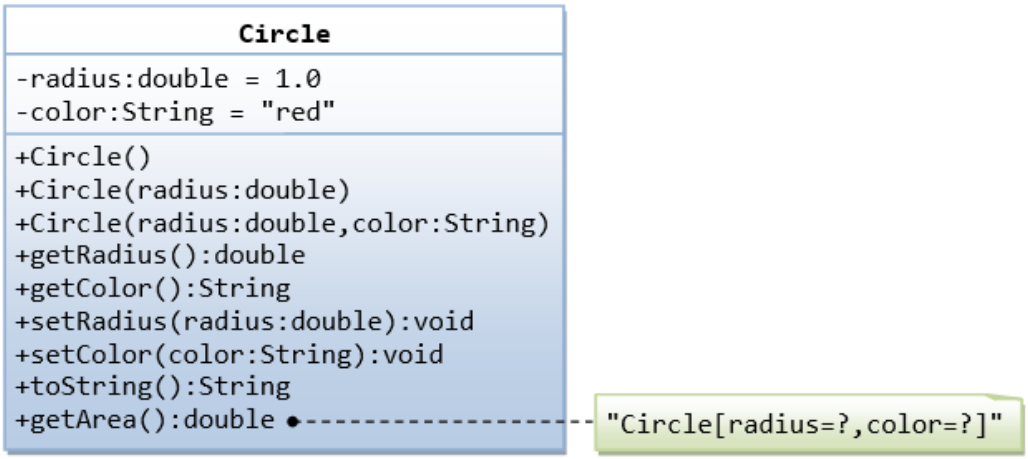
Try calling *toString()* method explicitly, just like any other method:

```
1 Circle circle5 = new Circle(5.5);
  System.out.println(circle5.toString());    // explicit call
```

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

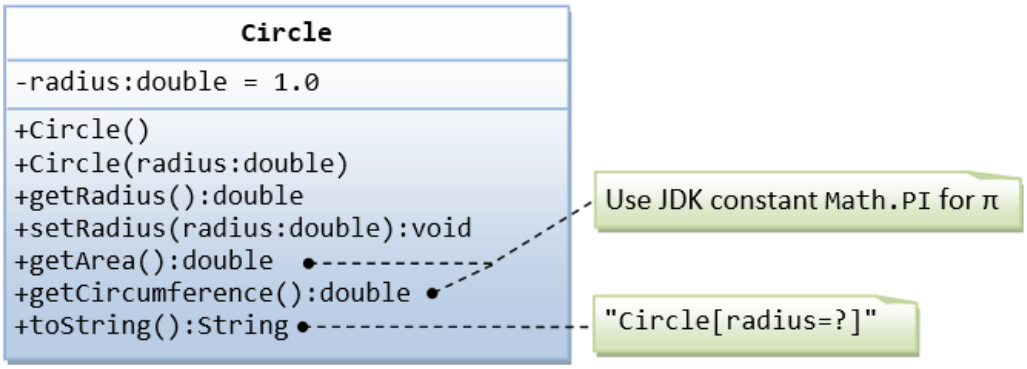
```
  Circle circle6 = new Circle(6.6);
2 System.out.println(circle6.toString());  // explicit call
  System.out.println(circle6);              // println() calls toString
     ↪ () implicitly, same as above
4 System.out.println("Operator '+' invokes toString() too: " + circle6
     ↪ );  // '+' invokes toString() too
```

The final class diagram for the Circle class is as follows:



1.2 Another Circle Class

A class called **Circle**, which models a circle with a radius, is designed as shown in the following class diagram. Write the **Circle** class.



Below is a test driver to test your **Circle** class.

```java
/**
 * Test Driver to test Circle class
 */
public class TestMain {

   public static void main(String[] args) {
      // Test Constructors and toString()
      Circle circle1 = new Circle(1.1);
      System.out.println(circle1);    // toString()
      Circle circle2 = new Circle(); // default constructor
      System.out.println(circle2);
```

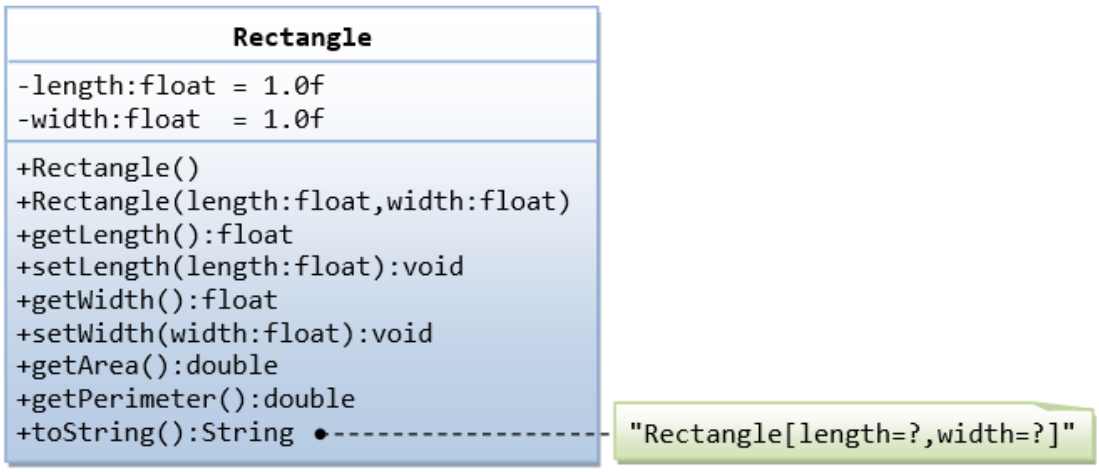
```
12
      // Test setter and getter
14    circle1.setRadius(2.2);
      System.out.println(circle1);       // toString()
16    System.out.println("radius is: " + circle1.getRadius());

18    // Test getArea() and getCircumference()
      System.out.printf("area is: %.2f%n", circle1.getArea());
20    System.out.printf("circumference is: %.2f%n", circle1.
         ↪ getCircumference());
   }
22 }
```

The expected output is:

```
Command window

Circle[radius = 1.1]
Circle[radius = 1.0]
Circle[radius = 2.2]
radius is: 2.2
area is: 15.21
circumference is: 13.82
```

1.3 The Rectangle Class

A class called Rectangle, which models a rectangle with a length and a width (in *float*), is designed as shown in the following class diagram. Write the Rectangle class.



Below is a test driver to test the Rectangle class:

```java
/**
 * Test Driver to test Rectangle class
 */
public class TestMain {

   public static void main(String[] args) {
      // Test constructors and toString()
      // You need to append a 'f' or 'F' to a float literal
      Rectangle rectangle1 = new Rectangle(1.2f, 3.4f);
      System.out.println(rectangle1);  // toString()
      Rectangle rectangle2 = new Rectangle();  // default constructor
      System.out.println(rectangle2);

      // Test setters and getters
      rectangle1.setLength(5.6f);
      rectangle1.setWidth(7.8f);
      System.out.println(rectangle1);  // toString()
      System.out.println("length is: " + rectangle1.getLength());
      System.out.println("width is: " + rectangle1.getWidth());

      // Test getArea() and getPerimeter()
      System.out.printf("area is: %.2f%n", rectangle1.getArea());
      System.out.printf("perimeter is: %.2f%n", rectangle1.getPerimeter());
   }
}
```

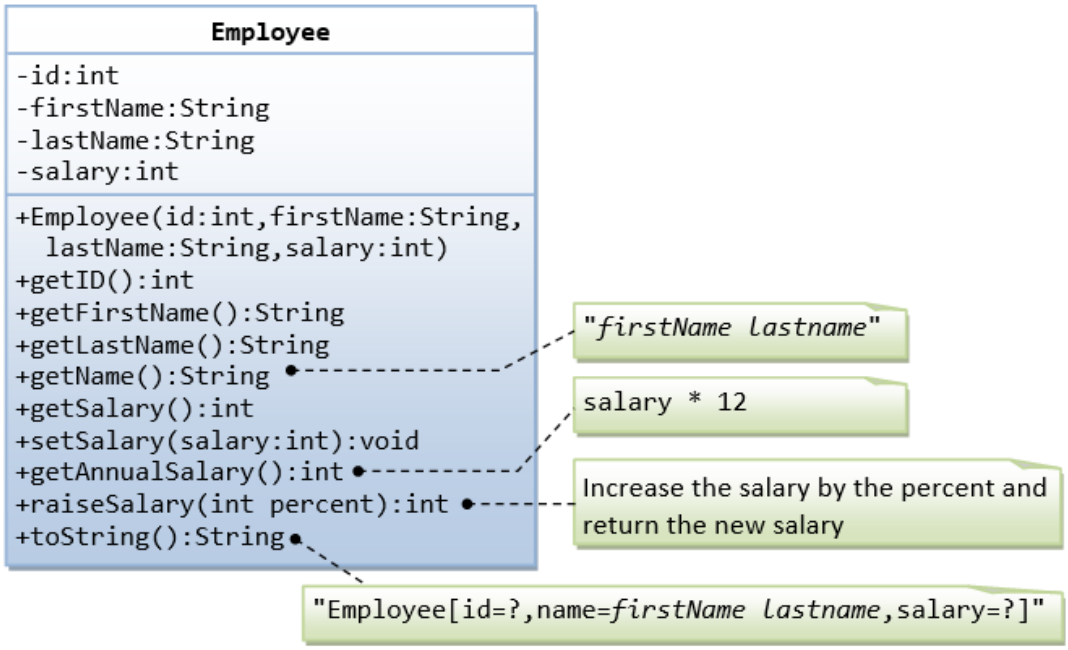
The expected output is:

Command window

```
Rectangle[length = 1.2, width = 3.4]
Rectangle[length = 1.0, width = 1.0]
Rectangle[length = 5.6, width = 7.8]
length is: 5.6
width is: 7.8
area is: 43.68
perimeter is: 26.80
```

1.4 The Employee Class

A class called Employee, which models an employee with an ID, name and salary, is designed as shown in the following class diagram. The method *raiseSalary(percent)* increases the salary by the given percentage. Write the Employee class.



Below is a test driver to test the Employee class:

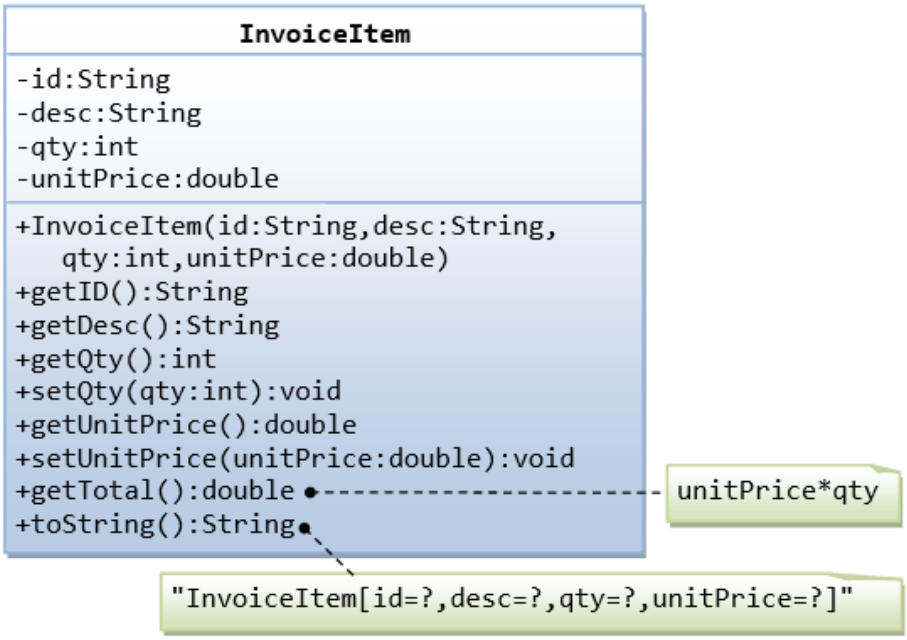
```java
/**
 * Test driver to test Employee class
 */
public class TestMain {

   public static void main(String[] args) {
      // Test constructor and toString()
      Employee employee1 = new Employee(8, "Peter", "Tan", 2500);
      System.out.println(employee1);  // toString();

      // Test Setters and Getters
      employee1.setSalary(999);
      System.out.println(e1);  // toString();
      System.out.println("id is: " + employee1.getID());
      System.out.println("firstname is: " + employee1.getFirstName());
      System.out.println("lastname is: " + employee1.getLastName());
      System.out.println("salary is: " + employee1.getSalary());

      System.out.println("name is: " + employee1.getName());
      System.out.println("annual salary is: " + employee1.getAnnualSalary()
         ↪ ); // Test method

      // Test raiseSalary()
      System.out.println(employee1.raiseSalary(10));
      System.out.println(employee1);
   }
```

The expected out is:

```
Command window

Employee[id = 8, name = Peter Tan, salary = 2500]
Employee[id = 8, name = Peter Tan, salary = 999]
id is: 8
firstname is: Peter
lastname is: Tan
salary is: 999
name is: Peter Tan
annual salary is: 11988
1098
Employee[id = 8, name = Peter Tan, salary = 1098]
```

1.5 The InvoiceItem Class

A class called InvoiceItem, which models an item of an invoice, with ID, description, quantity and unit price, is designed as shown in the following class diagram. Write the InvoiceItem class.



Below is a test driver to test the InvoiceItem class:

```java
/**
 * Test Driver to test InvoiceItem class
 */
public class TestMain {

   public static void main(String[] args) {
      // Test constructor and toString()
      InvoiceItem inv1 = new InvoiceItem("A101", "Pen Red", 888, 0.08);
      System.out.println(inv1);  // toString();

      // Test Setters and Getters
      inv1.setQty(999);
      inv1.setUnitPrice(0.99);
      System.out.println(inv1);  // toString();
      System.out.println("id is: " + inv1.getID());
      System.out.println("desc is: " + inv1.getDesc());
      System.out.println("qty is: " + inv1.getQty());
      System.out.println("unitPrice is: " + inv1.getUnitPrice());

      // Test getTotal()
      System.out.println("The total is: " + inv1.getTotal());
   }
}
```

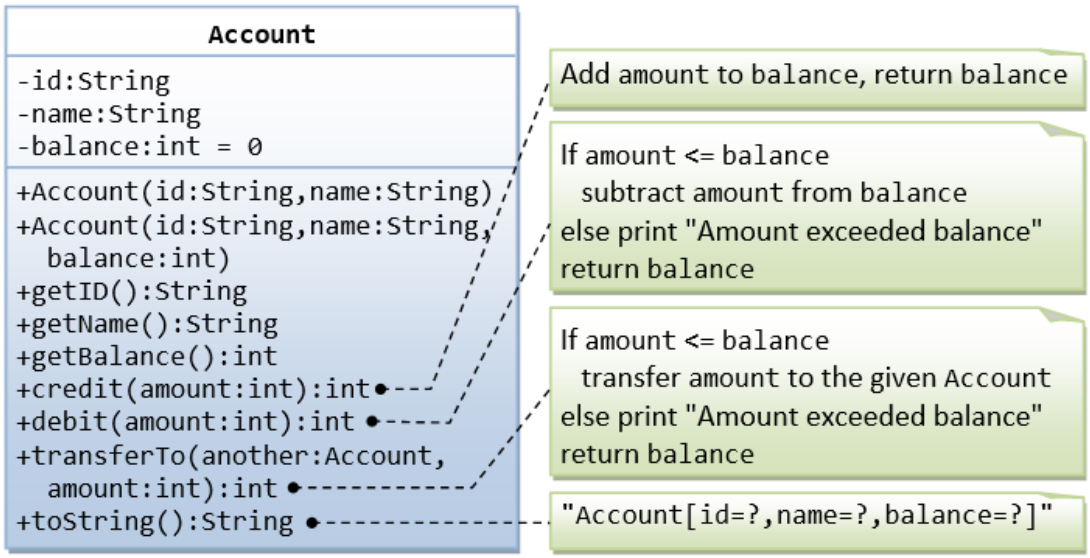
The expected output is:

Command window

```
InvoiceItem[id = A101, desc = Pen Red, qty = 888, unitPrice = 0.08]
InvoiceItem[id = A101, desc = Pen Red, qty = 999, unitPrice = 0.99]
id is: A101
desc is: Pen Red
qty is: 999
unitPrice is: 0.99
The total is: 989.01
```

1.6 The Account Class

A class called Account, which models a bank account of a customer, is designed as shown in the following class diagram. The methods credit(amount) and debit(amount) add or subtract the given amount to the balance. The method *transferTo(anotherAccount, amount)* transfers the given *amount* from this Account to the given *anotherAccount*. Write the Account class.



Below is a test driver to test the Account class:

```java
/**
 * Test driver to test Account class
 */
public class TestMain {

   public static void main(String[] args) {
      // Test constructor and toString()
      Account account1 = new Account("A101", "Tan Ah Teck", 88);
      System.out.println(account1);  // toString();
      Account account2 = new Account("A102", "Kumar"); // default balance
      System.out.println(account2);

      // Test Getters
      System.out.println("ID: " + account1.getID());
      System.out.println("Name: " + account1.getName());
      System.out.println("Balance: " + account1.getBalance());

      // Test credit() and debit()
      account1.credit(100);
      System.out.println(account1);
      account1.debit(50);
      System.out.println(account1);
      account1.debit(500);  // debit() error
      System.out.println(account1);

      // Test transfer()
      account1.transferTo(account2, 100);  // toString()
      System.out.println(account1);
      System.out.println(account2);
```

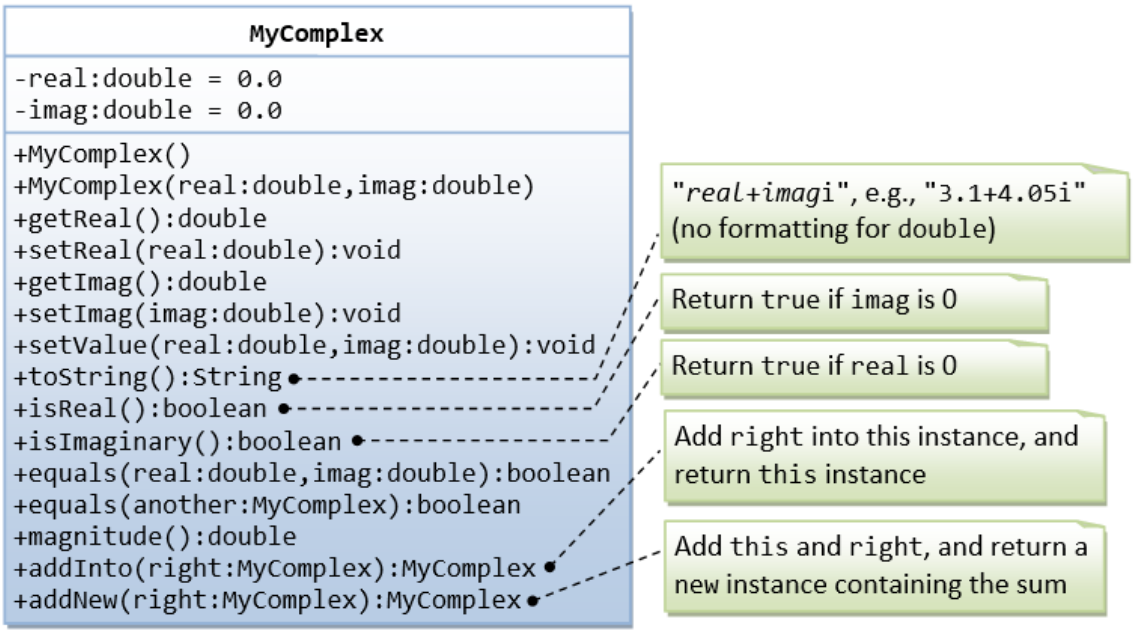
```
    }
}
```

The expected output is:

```
Command window

Account[id = A101,name = Tan Ah Teck,balance = 88]
Account[id = A102, name = Kumar, balance = 0]
ID: A101
Name: Tan Ah Teck
Balance: 88
Account[id = A101, name = Tan Ah Teck, balance = 188]
Account[id = A101, name = Tan Ah Teck, balance = 138]
Amount exceeded balance
Account[id = A101, name = Tan Ah Teck, balance = 138]
Account[id = A101, name = Tan Ah Teck, balance = 38]
Account[id = A102, name = Kumar, balance = 100]
```

1.7 The MyComplex class

A class called MyComplex, which models a complex number with real and imaginary parts, is designed as shown in the class diagram.



It contains:

- Two instance variable named *real* (*double*) and *imag* (*double*) which stores the real and imaginary parts of the complex number, respectively.
- A constructor that creates a MyComplex instance with the given real and imaginary values.
- A default constructor that create a MyComplex at $0.0 + 0.0i$.
- Getters and setters for instance variables *real* and *imag*.
- A method *setValue()* to set the value of the complex number.
- A *toString()* that returns "$(x + yi)$" where x and y are the real and imaginary parts, respectively.
- Methods *isReal()* and *isImaginary()* that returns *true* if this complex number is real or imaginary, respectively.

  Hints

  ```
  return (imag == 0);
  ```

- A method *equals(double real, double imag)* that returns *true* if this complex number is equal to the given complex number $(real, imag)$.

  Hints

  ```
  return (this.real == real && this.imag == imag);
  ```

- An overloaded *equals(MyComplex another)* that returns *true* if this complex number is equal to the given MyComplex instance another.

  Hints

  ```
  return (this.real == another.real && this.imag == another.imag);
  ```

- A method *magnitude()* that returns the magnitude of this complex number.

```
magnitude(x + yi) = Math.sqrt(x*x + y*y)
```

- Methods *addInto(MyComplex right)* that adds and subtract the given MyComplex instance (called right) into this instance and returns this instance.

```
(a + bi) + (c + di) = (a + c) + (b + d)i
```

Hints

```
return this;  // return "this" instance
```

- Methods *addNew(MyComplex right)* that adds this instance with the given MyComplex instance called right, and returns a new MyComplex instance containing the result.

Hints

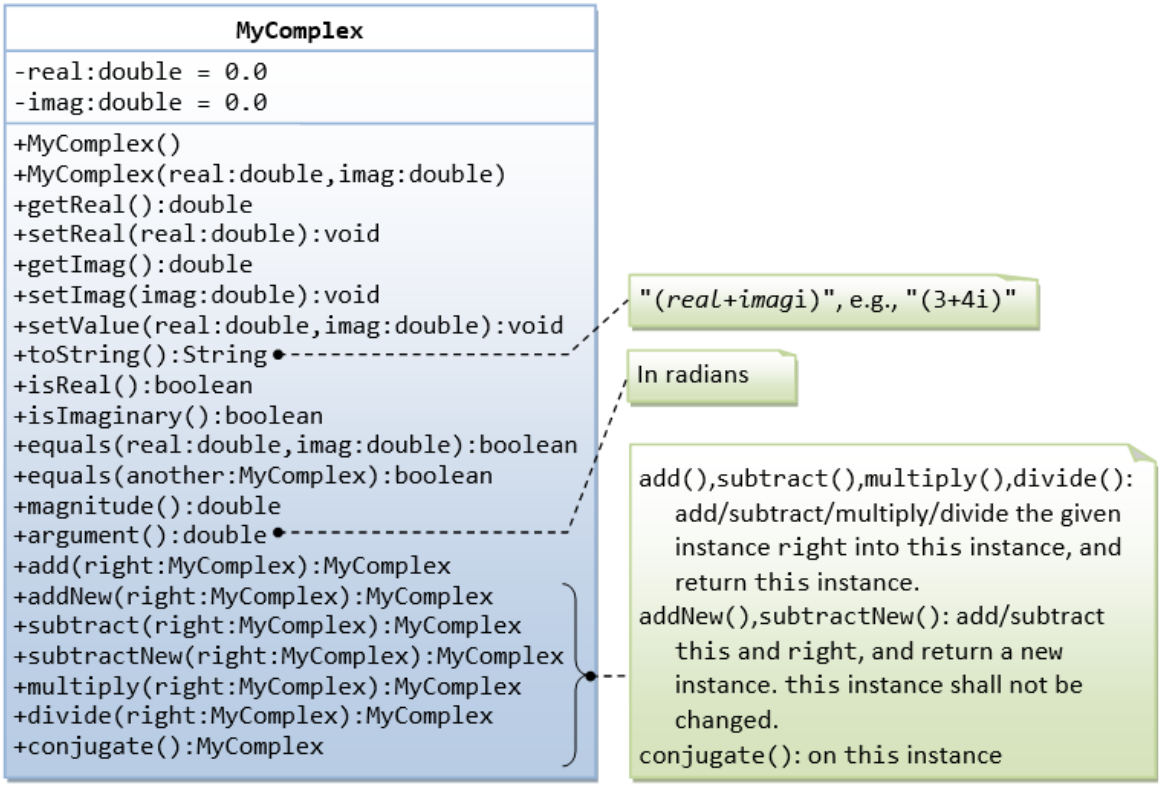
```
// construct a new instance and return the constructed instance
return new MyComplex(..., ...);
```

You are required to:

1. Write the MyComplex class.
2. Write a test driver to test all the public methods defined in the class.
3. Write an application called MyComplexApp that uses the MyComplex class. The application shall prompt the user for two complex numbers, print their values, check for real, imaginary and equality, and carry out all the arithmetic operations.

```
Command window

Enter complex number 1 (real and imaginary part): 1.1 2.2
Enter complex number 2 (real and imaginary part): 3.3 4.4

Number 1 is: (1.1 + 2.2i)
(1.1 + 2.2i) is NOT a pure real number
(1.1 + 2.2i) is NOT a pure imaginary number

Number 2 is: (3.3 + 4.4i)
(3.3 + 4.4i) is NOT a pure real number
(3.3 + 4.4i) is NOT a pure imaginary number

(1.1 + 2.2i) is NOT equal to (3.3 + 4.4i)
(1.1 + 2.2i) + (3.3 + 4.4i) = (4.4 + 6.6000000000000005i)
```

Try
A (more) complete design of MyComplex class is shown below:



- Methods *argument()* that returns the argument of this complex number in radians (*double*).

```
arg(x + yi) = Math.atan2(y, x) (in radians)
```

Note: The Math library has two arc-tangent methods, *Math.atan(double)* and *Math.atan2(double, double)*. We commonly use the *Math.atan2(y, x)* instead of *Math.atan(y/x)* to avoid division by zero. Read the documentation of Math class in package java.lang.

- The method *addInto()* is renamed *add()*. Also added *subtract()* and *subtractNew()*.
- Methods *multiply(MyComplex right)* and *divide(MyComplex right)* that multiplies and divides this instance with the given MyComplex instance right, and keeps the result in this instance, and returns this instance.

```
(a + bi) * (c + di) = (ac - bd) + (ad + bc)i
(a + bi) / (c + di) = [(a + bi) * (c - di)] / (c*c + d*d)
```

- A method conjugate() that operates on this instance and returns this instance containing the complex conjugate.

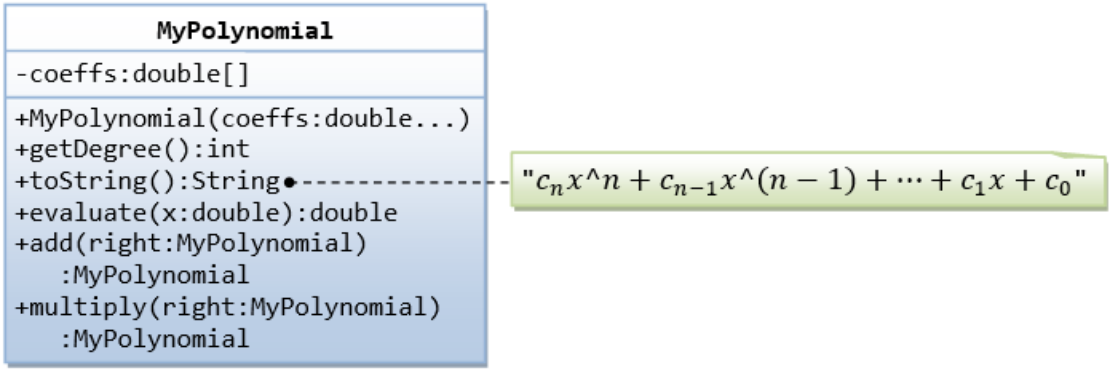
```
conjugate(x + yi) = x - yi
```

Take note that there are a few flaws in the design of this class, which was introduced solely for teaching purpose:

- Comparing *doubles* in *equal()* using "==" may produce unexpected outcome. For example, $(2.2 + 4.4) == 6.6$ returns false. It is common to define a small threshold called EPSILON (set to about 10^{-8}) for comparing floating point numbers.
- The method *addNew()*, *subtractNew()* produce new instances, whereas *add()*, *subtract()*, *multiply()*, *divide()* and *conjugate()* modify this instance. There is inconsistency in the design (introduced for teaching purpose).

Also take note that methods such as *add()* returns an instance of MyComplex. Hence, you can place the result inside a *System.out.println()* (which implicitly invoke the *toString()*). You can also chain the operations, e.g., *complex1.add(complex2).add(complex3)* (same as *(complex1.add(complex2)).add(complex3)*), or *complex1.add(complex2).subtract(complex3)*.

1.8 The MyPolynomial Class



A class called MyPolynomial, which models polynomials of degree-n (see equation), is designed as shown in the class diagram.

$$c_n x^n + c_{n-1}x^{n-1} + \cdots + c_1 x + c_0.$$

It contains:

- A constructor MyPolynomial(coeffs: *double*...) that takes a variable number of doubles to initialize the coeffs array, where the first argument corresponds to $c0$.
- The three dots is known as varargs (variable number of arguments), which is a new feature introduced in JDK 1.5. It accepts an array or a sequence of comma-separated arguments. The compiler automatically packs the comma-separated arguments in an array. The three dots can only be used for the last argument of the method.

Hints

```java
public class MyPolynomial {
   private double[] coeffs;

   public MyPolynomial(double... coeffs) {  // varargs
     this.coeffs = coeffs;  // varargs is treated as array
   }
   ......
}

// Test program
// Can invoke with a variable number of arguments
MyPolynomial polynomial1 = new MyPolynomial(1.1, 2.2, 3.3);
MyPolynomial polynomial1 = new MyPolynomial(1.1, 2.2, 3.3, 4.4, 5.5)
    ↪ ;

// Can also invoke with an array
```

```
double[] coeffs = {1.2, 3.4, 5.6, 7.8}
MyPolynomial polynomial2 = new MyPolynomial(coeffs);
```

- A method *getDegree()* that returns the degree of this polynomial.
- A method *toString()* that returns "$c_n x^n + c_{n-1} x^{n-1} + \cdots + c_1 x + c_0$.".
- A method *evaluate(double x)* that evaluate the polynomial for the given x, by substituting the given x into the polynomial expression.
- Methods *add()* and *multiply()* that adds and multiplies this polynomial with the given MyPolynomial instance another, and returns this instance that contains the result.

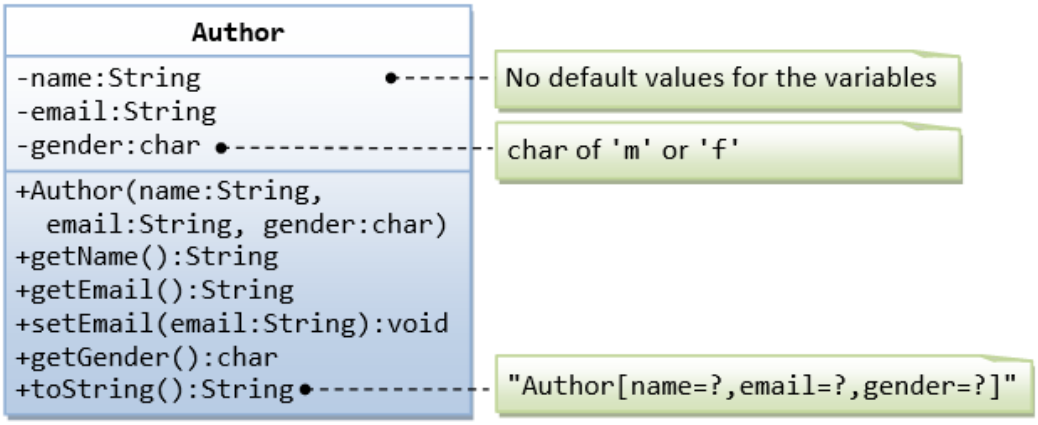
Write the MyPolynomial class. Also write a test driver (called TestMyPolynomial) to test all the public methods defined in the class.

Question: Do you need to keep the degree of the polynomial as an instance variable in the MyPolynomial class in Java? How about C/C++? Why?

2 Exercises on Composition

2.1 An Introduction to OOP Composition by Example - the Author and Book Classes

This first exercise shall lead you through all the concepts involved in OOP Composition.



A class called Author (as shown in the class diagram) is designed to model a book's author. 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;

```
1 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.)

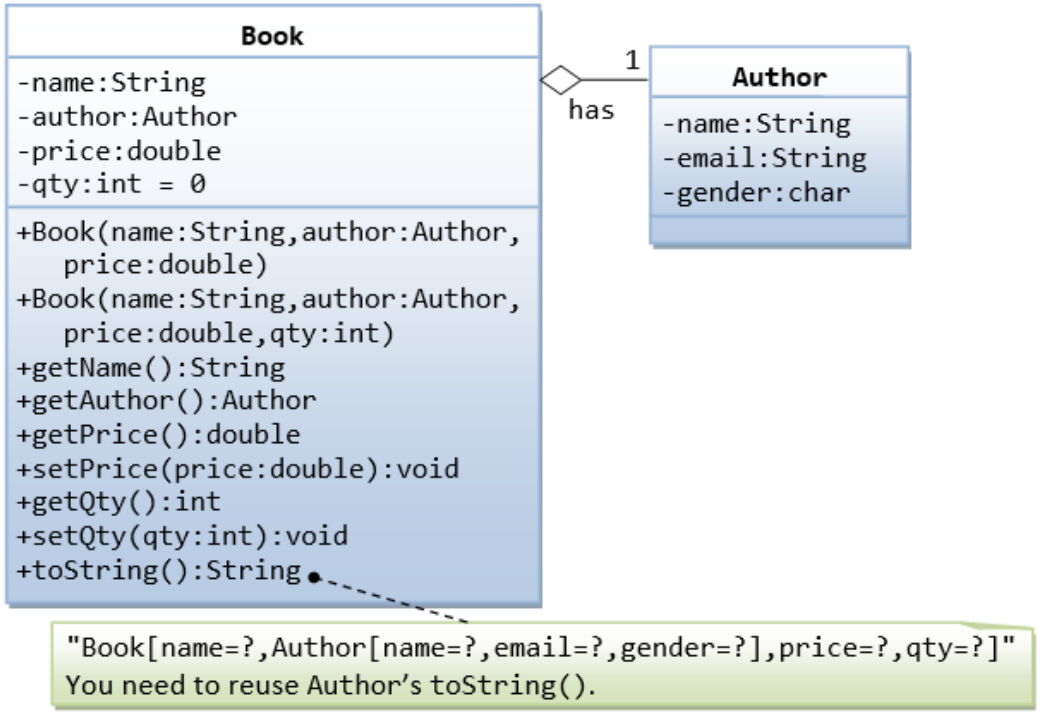
- public getters/setters: *getName()*, *getEmail()*, *setEmail()*, and *getGender()*;

  (There are no setters for name and gender, as these attributes cannot be changed.)

- A *toString()* method that returns "Author[name = ?, email = ?, gender = ?]", e.g., "Author[name = Tan Ah Teck, email = ahTeck@somewhere.com, gender = m]".

Write the Author class. Also write a test driver called TestAuthor to test all the public methods, e.g.,

```
Author ahTeck = new Author("Tan Ah Teck", "ahteck@nowhere.com", 'm'); // Test the constructor
System.out.println(ahTeck);  // Test toString()
ahTeck.setEmail("paulTan@nowhere.com");  // Test setter
System.out.println("name is: " + ahTeck.getName());      // Test getter
System.out.println("eamil is: " + ahTeck.getEmail());    // Test getter
System.out.println("gender is: " + ahTeck.getGender()); // Test getter
```



A class called Book is designed (as shown in the class diagram) to model a book written by one author. It contains:

- Four private instance variables: *name* (*String*), *author* (of the class Author you have just created, assume that a book has one and only one author), *price* (*double*), and *qty* (*int*);
- Two constructors:

```
public Book(String name, Author author, double price) {
   ......
}
```

```
public Book(String name, Author author, double price, int qty) {
   ......
}
```

- public methods *getName()*, *getAuthor()*, *getPrice()*, *setPrice()*, *getQty()*, *setQty()*.
- A *toString()* that returns "Book[name = ?, Author[name = ?, email = ?, gender = ?], price = ?, qty = ?". You should reuse Author's *toString()*.

Write the Book class (which uses the Author class written earlier). Also write a test driver called TestBook to test all the 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.,

```
// Construct an author instance
Author ahTeck = new Author("Tan Ah Teck", "ahteck@nowhere.com", 'm');
System.out.println(ahTeck);  // Author's toString()

Book dummyBook = new Book("Java for dummy", ahTeck, 19.95, 99);  // Test
    ↪ Book's Constructor
System.out.println(dummyBook);  // Test Book's toString()

// Test Getters and Setters
dummyBook.setPrice(29.95);
dummyBook.setQty(28);
System.out.println("name is: " + dummyBook.getName());
System.out.println("price is: " + dummyBook.getPrice());
System.out.println("qty is: " + dummyBook.getQty());
System.out.println("Author is: " + dummyBook.getAuthor());  // Author's
    ↪ toString()
System.out.println("Author's name is: " + dummyBook.getAuthor().getName());
System.out.println("Author's email is: " + dummyBook.getAuthor().getEmail
    ↪ ());

// Use an anonymous instance of Author to construct a Book instance
Book anotherBook = new Book("more Java",
new Author("Paul Tan", "paul@somewhere.com", 'm'), 29.95);
System.out.println(anotherBook);  // toString()
```

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 anAuthor, anAuthor.name refers to the name of the author. There is no need (and not recommended) to call the variables *bookName* and *authorName*.

Try

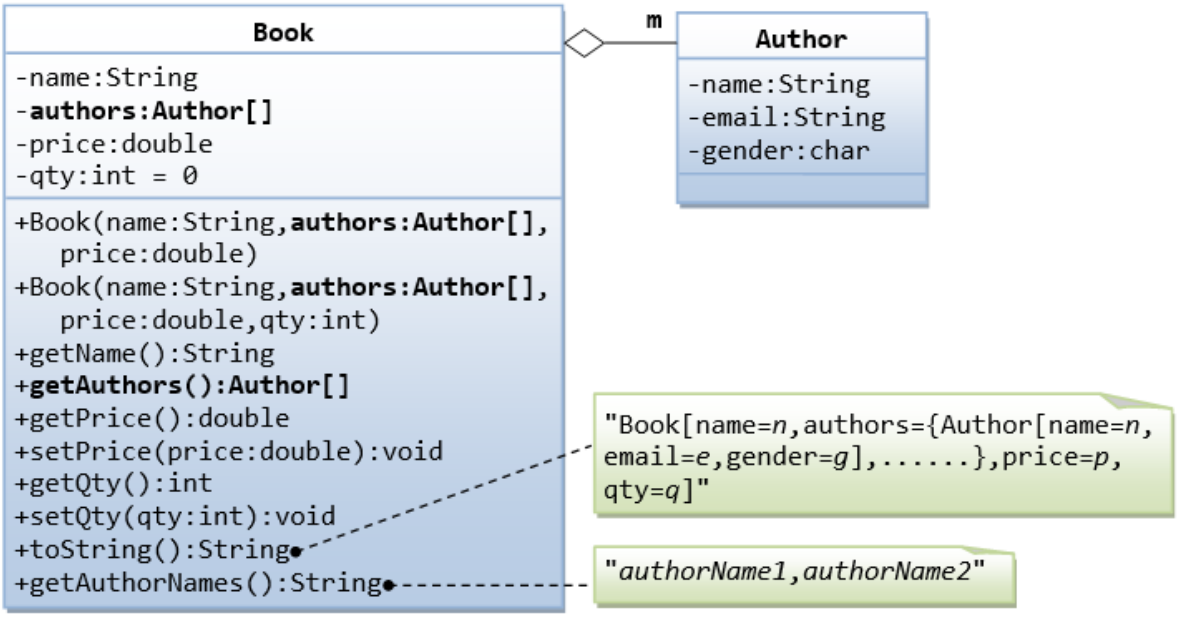
1. Printing the name and email of the author from a Book instance.

   (**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,

```java
public String getAuthorName() {
   return author.getName();  // cannot use author.name as name is
      ↪ private in Author class
}
```

2.2 (Advanced) The Author and Book Classes Again - An Array of Objects as an Instance Variable



In the earlier exercise, 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.

Notes

- The constructors take an array of Author (i.e., Author[ ]), instead of an Author instance. In this design, once a Book instance is constructor, you cannot add or remove author.

- The *toString()* method shall return "Book[name = ?, authors = Author[name = ?, email = ?, gender = ?], ..., price = ?, qty = ?]".

You are required to:

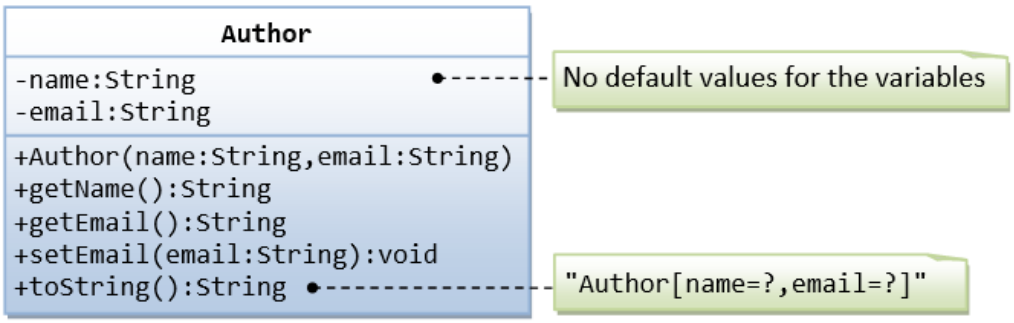
1. Write the code for the Book class. You shall re-use the Author class written earlier.
2. Write a test driver (called TestBook) to test the Book class.

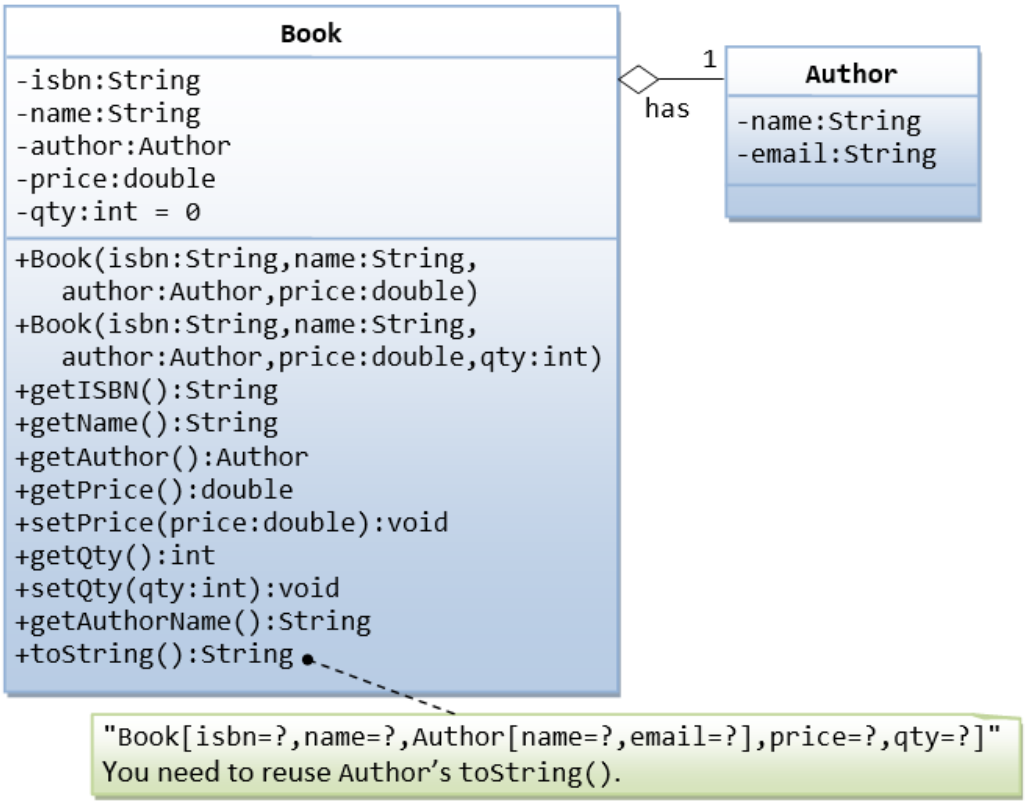
Try

```java
// 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()
```

2.3 The Author and Book Classes - Your Turn

A class called Author, which models an author of a book, is designed as shown in the class diagram. A class called Book, which models a book written by ONE author and composes an instance of Author as its instance variable, is also shown. Write the Author and Book classes.





Below is a test driver:

```java
/**
 * Test driver class
 */
public class TestMain {

   public static void main(String[] args) {

      // Test Author class
      Author author1 = new Author("Tan Ah Teck", "ahteck@nowhere.com");
      System.out.println(author1);

      author1.setEmail("ahteck@somewhere.com");
      System.out.println(author1);
      System.out.println("name is: " + author1.getName());
      System.out.println("email is: " + author1.getEmail());

      // Test Book class
      Book book1 = new Book("12345", "Java for dummies", a1, 8.8, 88);
      System.out.println(book1);

      book1.setPrice(9.9);
```

```
22      book1.setQty(99);
        System.out.println(book1);
24      System.out.println("isbn is: " + book1.getName());
        System.out.println("name is: " + book1.getName());
26      System.out.println("price is: " + book1.getPrice());
        System.out.println("qty is: " + book1.getQty());
28      System.out.println("author is: " + book1.getAuthor());  // Author's
           ↪ toString()
        System.out.println("author's name: " + book1.getAuthorName());
30      System.out.println("author's name: " + book1.getAuthor().getName());
        System.out.println("author's email: " + book1.getAuthor().getEmail());
32    }
  }
```

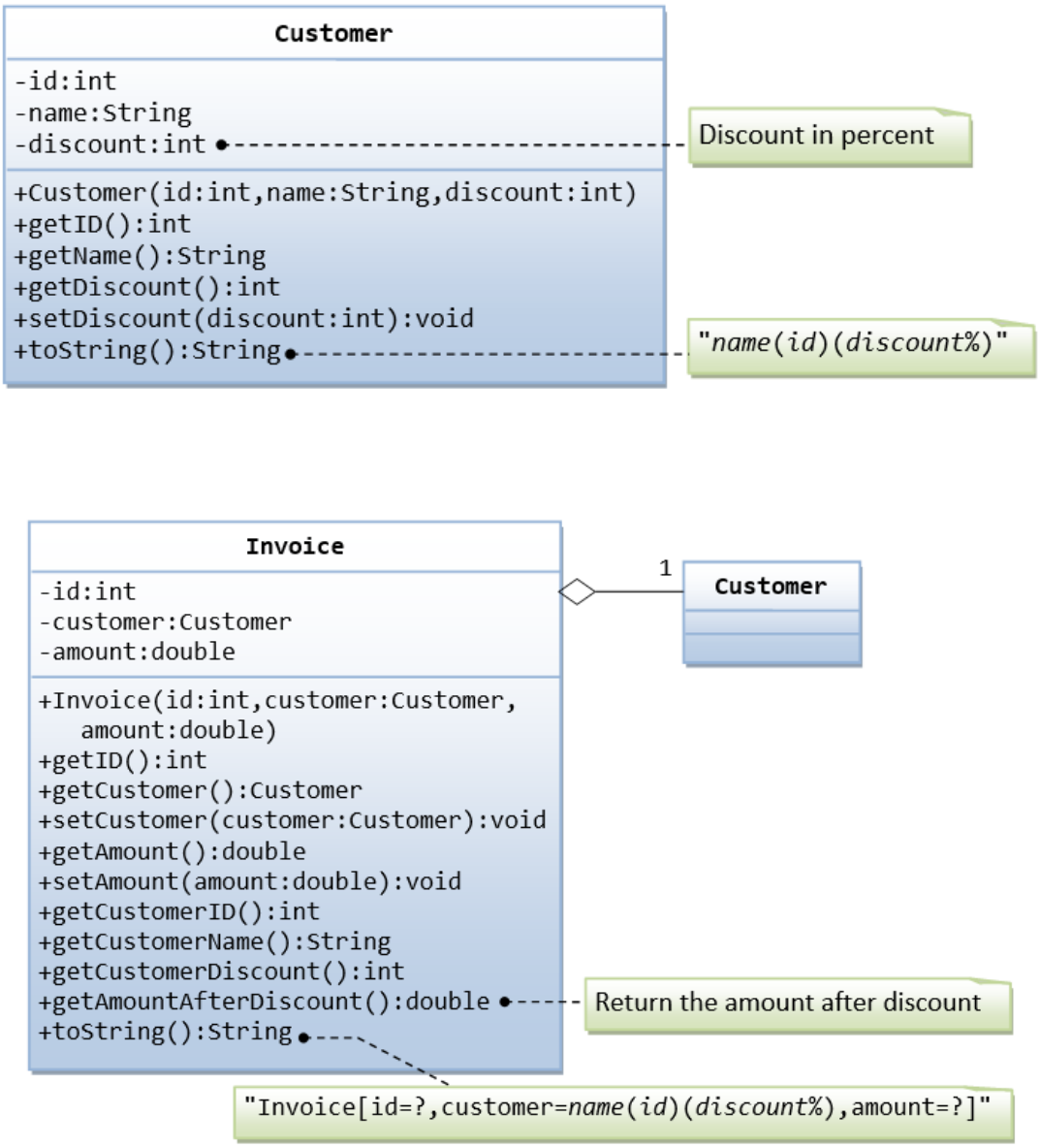
The expected output is:

Command window

```
1  Author[name = Tan Ah Teck, email = ahteck@nowhere.com]
   Author[name = Tan Ah Teck, email = ahteck@somewhere.com]
3  name is: Tan Ah Teck
   email is: ahteck@somewhere.com
5  Book[isbn = 12345,name = Java for dummies, Author[name = Tan Ah Teck,
       email = ahteck@somewhere.com], price = 8.8, qty = 88]
   Book[isbn = 12345, name = Java for dummies, Author[name = Tan Ah Teck,
       email = ahteck@somewhere.com], price = 9.9, qty = 99]
7  isbn is: Java for dummies
   name is: Java for dummies
9  price is: 9.9
   qty is: 99
11 author is: Author[name = Tan Ah Teck, email = ahteck@somewhere.com]
   author's name: Tan Ah Teck
13 author's name: Tan Ah Teck
   author's email: ahteck@somewhere.com
```

2.4 The Customer and Invoice classes

A class called Customer, which models a customer in a transaction, is designed as shown in the class diagram. A class called Invoice, which models an invoice for a particular customer and composes an instance of Customer as its instance variable, is also shown. Write the Customer and Invoice classes.



Below is a test driver:

```
/**
 * Test driver class
 */
public class TestMain {
   public static void main(String[] args) {

      // Test Customer class
      Customer customer1 = new Customer(88, "Tan Ah Teck", 10);
      System.out.println(customer1);  // Customer's toString()
```

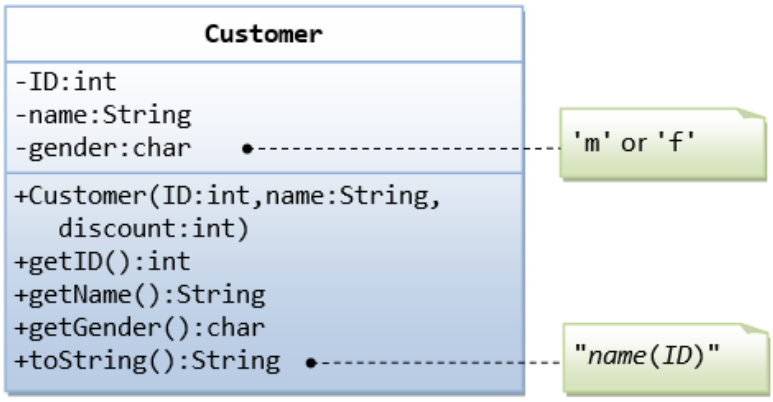
```java

      customer1.setDiscount(8);
      System.out.println(customer1);
      System.out.println("id is: " + customer1.getID());
      System.out.println("name is: " + customer1.getName());
      System.out.println("discount is: " + customer1.getDiscount());

      // Test Invoice class
      Invoice invoice1 = new Invoice(101, c1, 888.8);
      System.out.println(inv1);

      invoice1.setAmount(999.9);
      System.out.println(invoice1);
      System.out.println("id is: " + invoice1.getID());
      System.out.println("customer is: " + invoice1.getCustomer());  //
         ↪ Customer's toString()
      System.out.println("amount is: " + invoice1.getAmount());
      System.out.println("customer's id is: " + invoice1.getCustomerID());
      System.out.println("customer's name is: " + invoice1.getCustomerName
         ↪ ());
      System.out.println("customer's discount is: " + invoice1.
         ↪ getCustomerDiscount());
      System.out.printf("amount after discount is: %.2f%n", inv1.
         ↪ getAmountAfterDiscount());
   }
}
```

The expected output is:

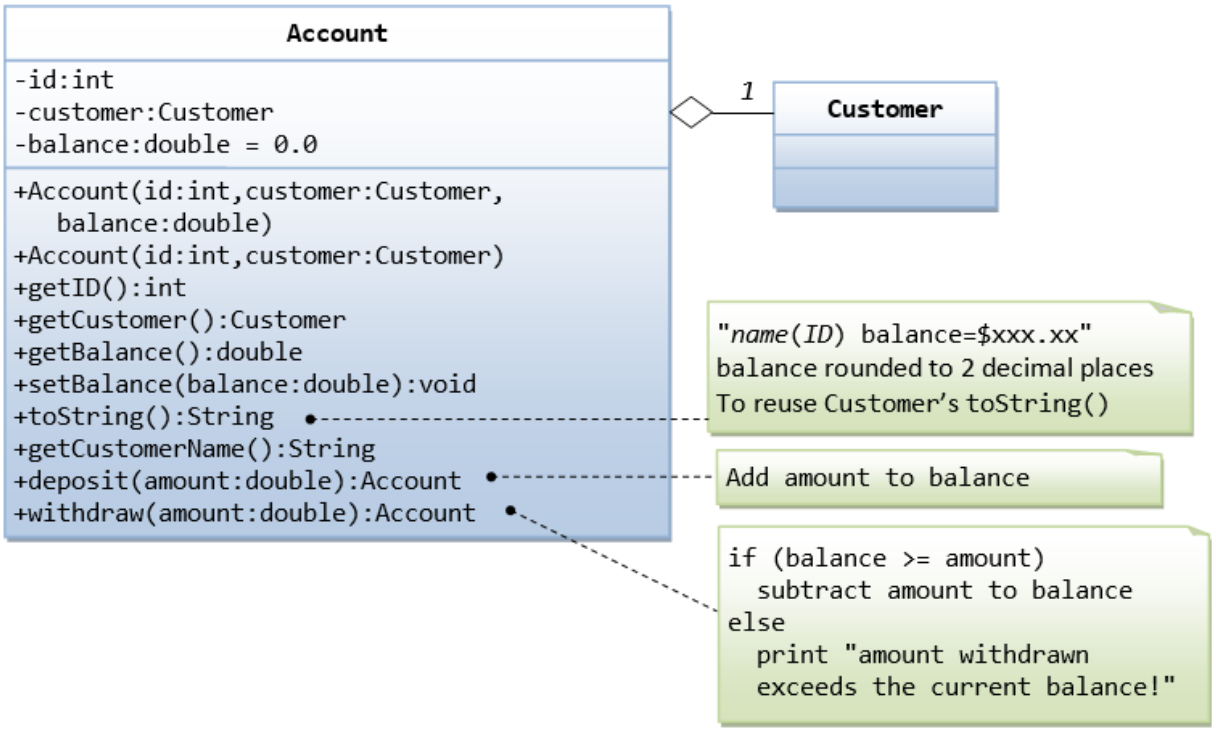
Command window

```
Tan Ah Teck(88)(10%)
Tan Ah Teck(88)(8%)
id is: 88
name is: Tan Ah Teck
discount is: 8
Invoice[id = 101, customer = Tan Ah Teck(88)(8%), amount = 888.8]
Invoice[id = 101, customer = Tan Ah Teck(88)(8%), amount = 999.9]
id is: 101
customer is: Tan Ah Teck(88)(8%)
amount is: 999.9
customer's id is: 88
customer's name is: Tan Ah Teck
customer's discount is: 8
amount after discount is: 919.91
```

2.5 The Customer and Account classes



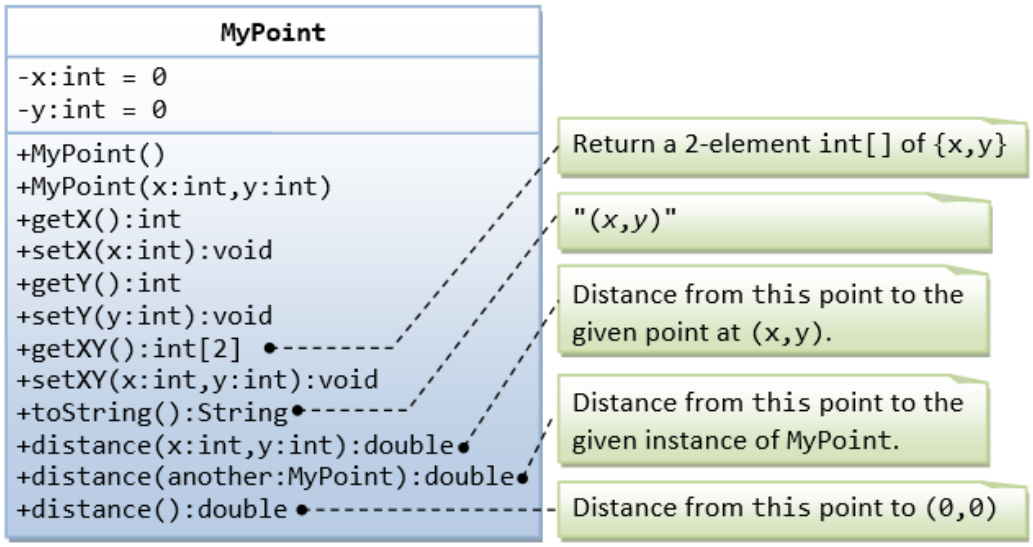
The Customer class models a customer is design as shown in the class diagram. Write the codes for the Customer class and a test driver to test all the public methods.



The Account class models a bank account, design as shown in the class diagram, composes a Customer instance (written earlier) as its member. Write the codes for the Account class and a test driver to test all the public methods.

2.6 The MyPoint Class

A class called MyPoint, which models a 2D point with x and y coordinates, is designed as shown in the class diagram.



It contains:

- Two instance variables x (*int*) and y (*int*).
- A default (or "no-argument" or "no-arg") constructor that construct a point at the default location of $(0, 0)$.
- A overloaded constructor that constructs a point with the given x and y coordinates.
- Getter and setter for the instance variables x and y.
- A method setXY() to set both x and y.
- A method getXY() which returns the x and y in a 2-element int array.
- A *toString()* method that returns a string description of the instance in the format "(x, y)".
- A method called *distance(int x, int y)* that returns the distance from this point to another point at the given (x, y) coordinates, e.g.,

```
MyPoint point1 = new MyPoint(3, 4);
System.out.println(point1.distance(5, 6));
```

- An overloaded distance(MyPoint another) that returns the distance from this point to the given MyPoint instance (called another), e.g.,

```
MyPoint point1 = new MyPoint(3, 4);
MyPoint point2 = new MyPoint(5, 6);
System.out.println(point1.distance(point2));
```

- Another overloaded distance() method that returns the distance from this point to the origin $(0, 0)$, e.g.,

```
MyPoint point1 = new MyPoint(3, 4);
System.out.println(point1.distance());
```

You are required to:

1. Write the code for the class MyPoint. Also write a test program (called TestMyPoint) to test all the methods defined in the class.

   Hints

```
// Overloading method distance()
// This version takes two ints as arguments
public double distance(int x, int y) {
   int xDiff = this.x - x;
   int yDiff = ......
   return Math.sqrt(xDiff*xDiff + yDiff*yDiff);
}

// This version takes a MyPoint instance as argument
public double distance(MyPoint another) {
   int xDiff = this.x - another.x;
   .......
}
```

```
// Test program to test all constructors and public methods
MyPoint point1 = new MyPoint();   // Test constructor
System.out.println(point1);       // Test toString()

// Test setters
point1.setX(8);
```

```
point1.setY(6);

// Test getters
System.out.println("x is: " + point1.getX());
System.out.println("y is: " + point1.getY());
p1.setXY(3, 0);    // Test setXY()
System.out.println(point1.getXY()[0]);  // Test getXY()
System.out.println(point1.getXY()[1]);
System.out.println(point1);

MyPoint point2 = new MyPoint(0, 4);  // Test another constructor
System.out.println(point2);

// Testing the overloaded methods distance()
System.out.println(point1.distance(point2));  // which version?
System.out.println(point2.distance(point1));  // which version?
System.out.println(point1.distance(5, 6));  // which version?
System.out.println(point1.distance());      // which version?
```

2. Write a program that allocates 10 points in an array of MyPoint, and initializes to $(1, 1), (2, 2), \ldots, (10, 10)$.

   Hints

   You need to allocate the array, as well as each of the 10 MyPoint instances. In other words, you need to issue 11 new, 1 for the array and 10 for the MyPoint instances.

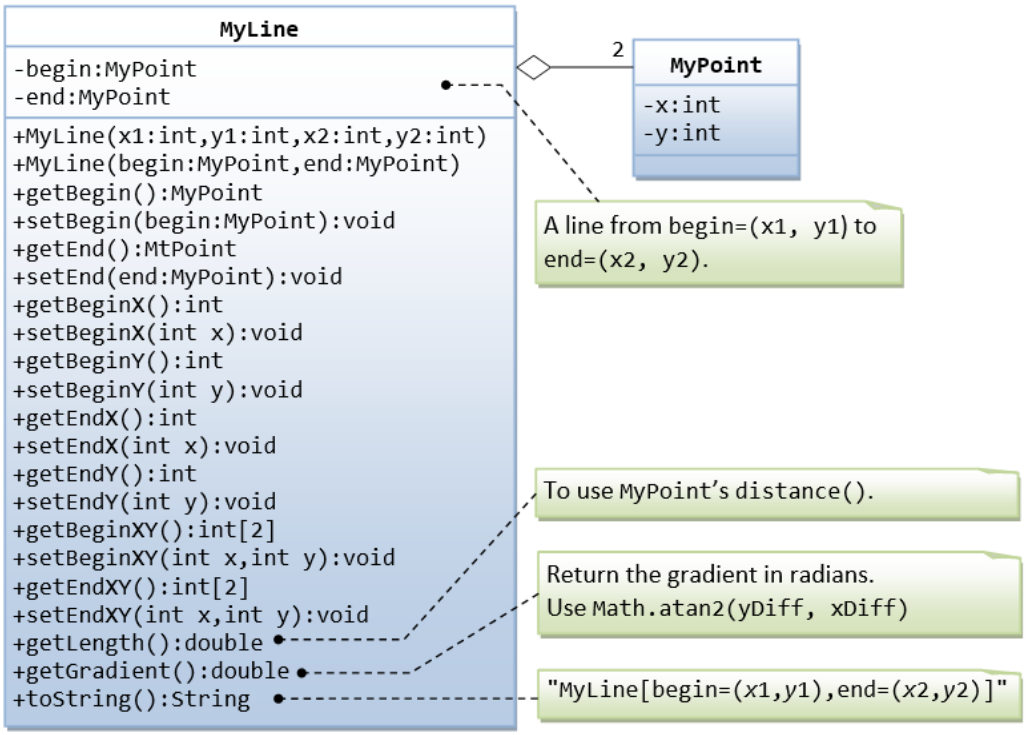
```
MyPoint[] points = new MyPoint[10];  // Declare and allocate an
   ↪ array of MyPoint
for (int i = 0; i < points.length; i++) {
   points[i] = new MyPoint(...);     // Allocate each of MyPoint instances
}
// use a loop to print all the points
```

Notes

Point is such a common entity that JDK certainly provided for in all flavors.

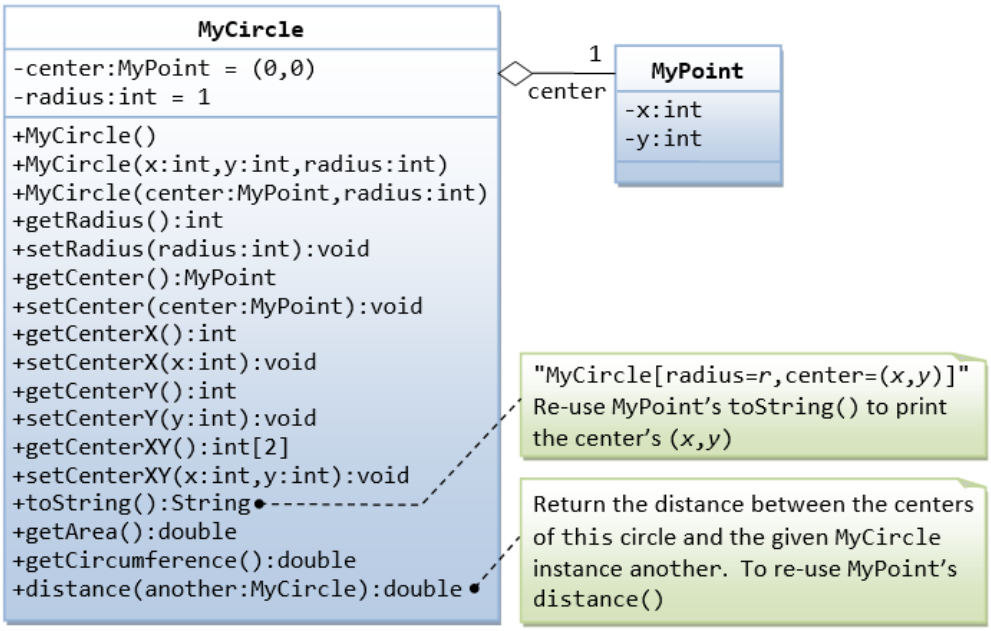
2.7 The MyLine and MyPoint Classes

A class called MyLine, which models a line with a begin point at (x1, y1) and an end point at (x2, y2), is designed as shown in the class diagram. The MyLine class uses two MyPoint instances (written in the earlier exercise) as its begin and end points. Write the MyLine class. Also write a test driver to test all the public methods in the MyLine class.



2.8 The MyCircle and MyPoint Classes

A class called MyCircle, which models a circle with a center and a radius, is designed as shown in the class diagram. The MyCircle class uses a MyPoint instance (written in the earlier 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.
- A default constructor that construct a circle with center at $(0, 0)$ and radius of 1.
- Various getters and setters.
- A *toString()* method that returns a string description of this instance in the format "MyCircle[radius = r, center = (x, y)]". You shall reuse the *toString()* of MyPoint.
- *getArea()* and *getCircumference()* methods that return the area and circumference of this circle in *double*.
- A *distance(MyCircle another)* method that returns the distance of the centers from this instance and the given MyCircle instance. You should use MyPoint's *distance()* method to compute this distance.

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

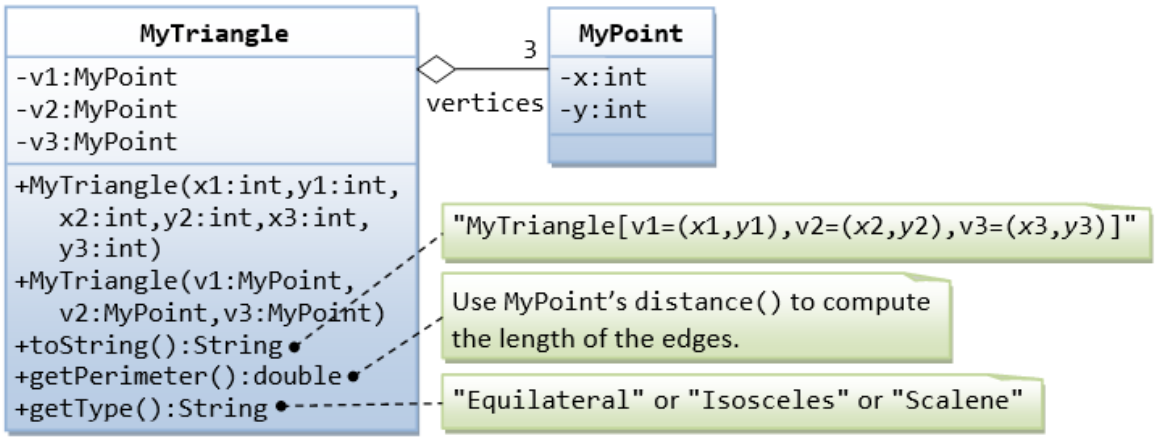
Hints

```
// Constructors
public MyCircle(int x, int y, int radius) {
  // Need to construct an instance of MyPoint for the variable center
  center = new MyPoint(x, y);
  this.radius = radius;
}
public MyCircle(MyPoint center, int radius) {
  // An instance of MyPoint already constructed by caller; simply assign.
  this.center = center;
  ......
}
public MyCircle() {
  center = new MyPoint(.....);  // construct MyPoint instance
  this.radius = ......
}

// Returns the x-coordinate of the center of this MyCircle
public int getCenterX() {
  return center.getX();    // cannot use center.x and x is private in MyPoint
}

// Returns the distance of the center for this MyCircle and another MyCircle
public double distance(MyCircle another) {
```

```
    return center.distance(another.center); // use distance() of MyPoint
25 }
```

2.9 The MyTriangle and MyPoint Classes

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



It contains:

- Three private instance variables $v1, v2, v3$ (instances of MyPoint), for the three vertices.
- A constructor that constructs a MyTriangle with three set of coordinates, $v1 = (x1, y1)$, $v2 = (x2, y2)$, $v3 = (x3, y3)$.
- 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 "MyTriangle[v1 = (x1, y1), v2 = (x2, y2), v3 = (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 driver (called TestMyTriangle) to test all the public methods defined in the class.

2.10 The MyRectangle and MyPoint Classes

Design a MyRectangle class which is composed of two MyPoint instances as its top-left and bottom-right corners. Draw the class diagrams, write the codes, and write the test drivers.