

## Lab 3. 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: "Java Style and Commenting Guide".
- 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.

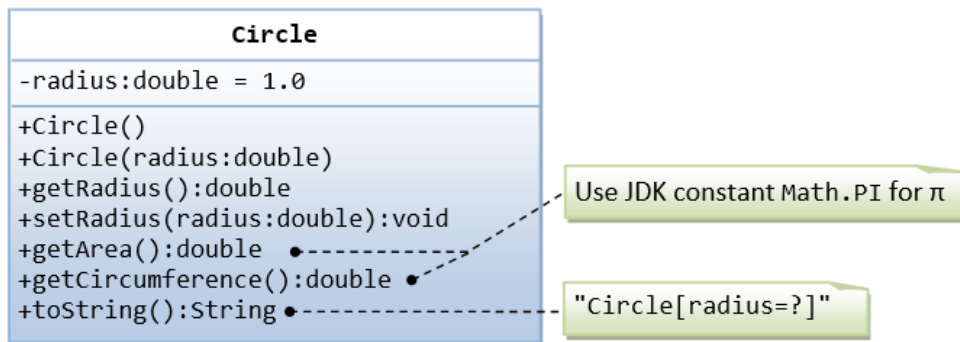
#### 2. Program Documentation: Comment! Comment! and more Comment to explain your code to other people and to yourself three days later.

#### 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:

```

1  /**
2   * The Circle class models a circle with a radius and color.
3   */
4  public class Circle { // Save as "Circle.java"
5      // private instance variable, not accessible from outside this class
6      private double radius;
7      private String color;
8
9      // Constructors (overloaded)
10     /** Constructs a Circle instance with default value for radius and
11         * color */
12     public Circle() { // 1st (default) constructor
13         radius = 1.0;
14         color = "red";
15     }
  
```



```

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

21
    /** Returns the radius */
23  public double getRadius() {
        return radius;
25  }

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

```

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:



```

1  /**
   * A Test Driver for the Circle class
3  */
    public class TestCircle { // Save as "TestCircle.java"
5      public static void main(String[] args) {
        // Declare an instance of Circle class called circle1.
7        // Construct the instance circle1 by invoking the "default"
        ↪ constructor
        // which sets its radius and color to their default value.
9        Circle circle1 = new Circle();
        // Invoke public methods on instance circle1, via dot operator.
11       System.out.println("The circle has radius of "
        + circle1.getRadius() + " and area of " + circle1.getArea());
13       //The circle has radius of 1.0 and area of 3.141592653589793

15       // Declare an instance of class circle called circle2.
        // Construct the instance circle2 by invoking the second
        ↪ constructor
17       // with the given radius and default color.
        Circle circle2 = new Circle(2.0);
19       // Invoke public methods on instance circle2, via dot operator.
        System.out.println("The circle has radius of "
21       + circle2.getRadius() + " and area of " + circle2.getArea());

```



```

23      }
      //The circle has radius of 2.0 and area of 12.566370614359172
    }
  }

```

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*.



```

2      // 3rd constructor 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 this instance.



```

1      // 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(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:



```

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

5
      // Setter for instance variable color
7      public void setColor(String newColor) { ..... }

```

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



```

1      Circle circle4 = new Circle();    // construct an instance of
      ↪ Circle
2      circle4.setRadius(5.5);           // change radius
      System.out.println("radius is: " + circle4.getRadius()); //
      ↪ Print radius via getter
4      circle4.setColor("green");        // Change color
      System.out.println("color is: " + circle4.getColor()); //
      ↪ Print color via getter
6
      // You cannot do the following because setRadius() returns void,
      ↪ which cannot be printed
8      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,



```

1      // Instance variable
      private double radius;
3
      /** Constructs a Circle instance with the given radius and
      ↪ default color */
5      public Circle(double radius) {
          this.radius = radius; // "this.radius" refers to the
          ↪ instance variable
7          // "radius" refers to the method's parameter
          color = "red";
9      }

11     /** Sets the radius to the given value */
      public void setRadius(double radius) {
13         this.radius = radius; // "this.radius" refers to the
          ↪ instance variable
          // "radius" refers to the method's argument
15     }

```

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
2     ↪ form of Circle[radius = ?, color = ?] */
3  public String toString() {
4     return "Circle[radius = " + radius + " color = " + color + "]"
5     ↪ ;
6  }

```

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,

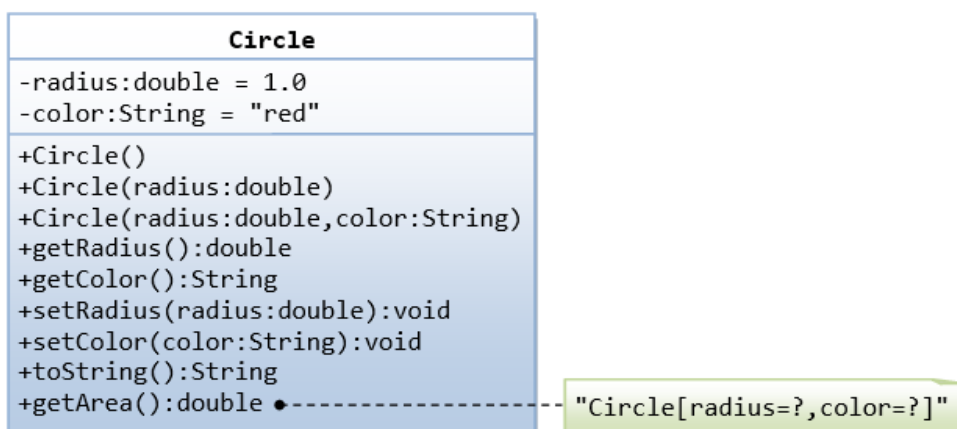


```

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

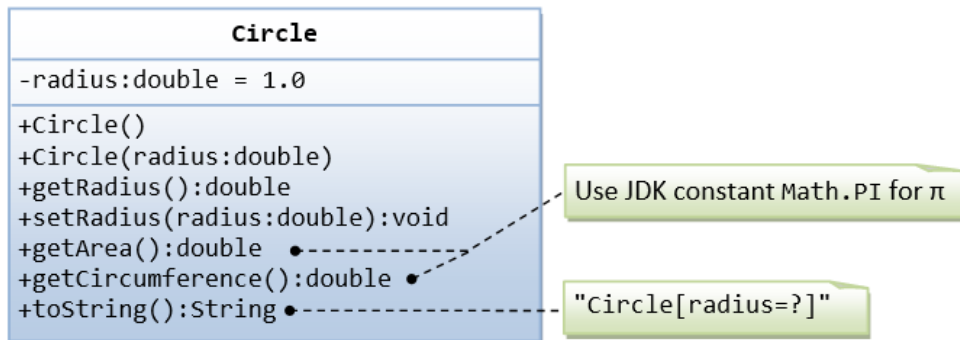
```

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.



```

1  public class TestMain {
    public static void main(String[] args) {
2      // Test Constructors and toString()
3      Circle circle1 = new Circle(1.1);
4      System.out.println(circle1); // toString()
5      Circle circle2 = new Circle(); // default constructor
6      System.out.println(circle2);
7
8      // Test setter and getter
9      circle1.setRadius(2.2);
10     System.out.println(circle1); // toString()
11     System.out.println("radius is: " + circle1.getRadius());
12
13     // Test getArea() and getCircumference()
14     System.out.printf("area is: %.2f\n", circle1.getArea());
15     System.out.printf("circumference is: %.2f\n", circle1.
16         ↪ getCircumference());
17     }
18 }
  
```

The expected output is:

## Command window

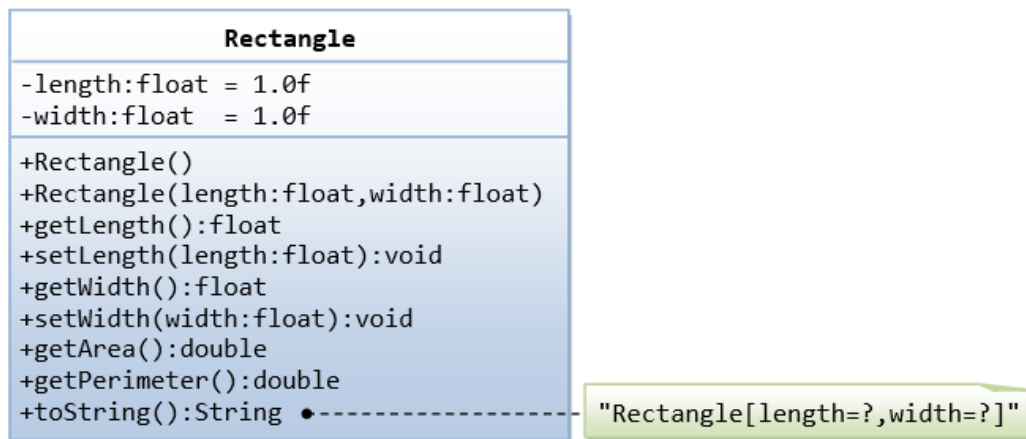
```

Circle[radius = 1.1]
2 Circle[radius = 1.0]
Circle[radius = 2.2]
4 radius is: 2.2
area is: 15.21
6 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:



```

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

8   // Test setters and getters
    rectangle1.setLength(5.6f);
10   rectangle1.setWidth(7.8f);
    System.out.println(rectangle1); // toString()
12   System.out.println("length is: " + rectangle1.getLength());
    System.out.println("width is: " + rectangle1.getWidth());
14   // Test getArea() and getPerimeter()
16

```





```

18      System.out.printf("area is: %.2f%n", rectangle1.getArea());
        System.out.printf("perimeter is: %.2f%n", rectangle1.getPerimeter()
        ↪ );
20    }
    }

```

The expected output is:

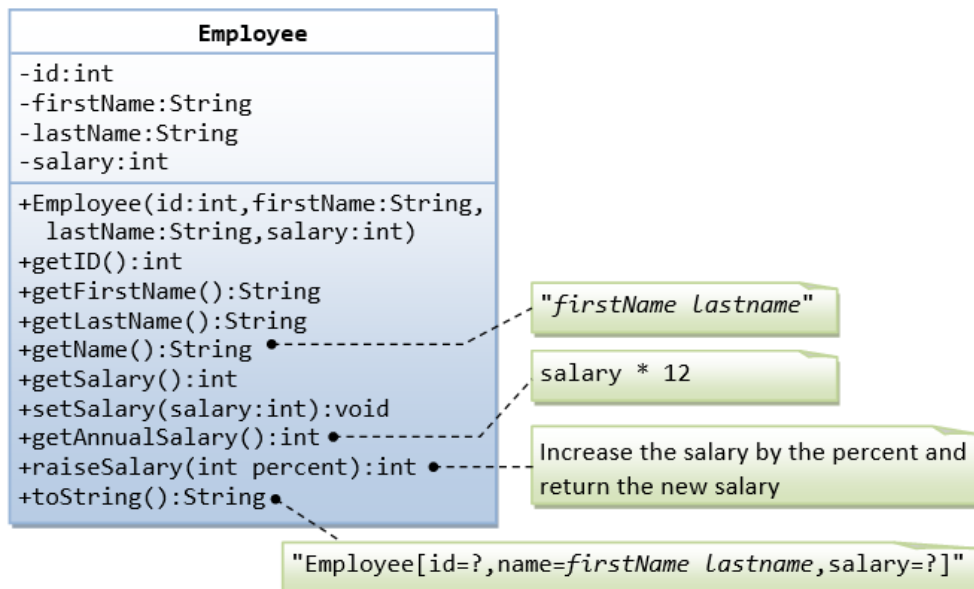
```

Command window
1  Rectangle[length = 1.2, width = 3.4]
   Rectangle[length = 1.0, width = 1.0]
3  Rectangle[length = 5.6, width = 7.8]
   length is: 5.6
5   width is: 7.8
   area is: 43.68
7   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:



```

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

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

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

11     // Test raiseSalary()
        System.out.println(employee1.raiseSalary(10));
12     System.out.println(employee1);
13 }
14 }

```

The expected out is:

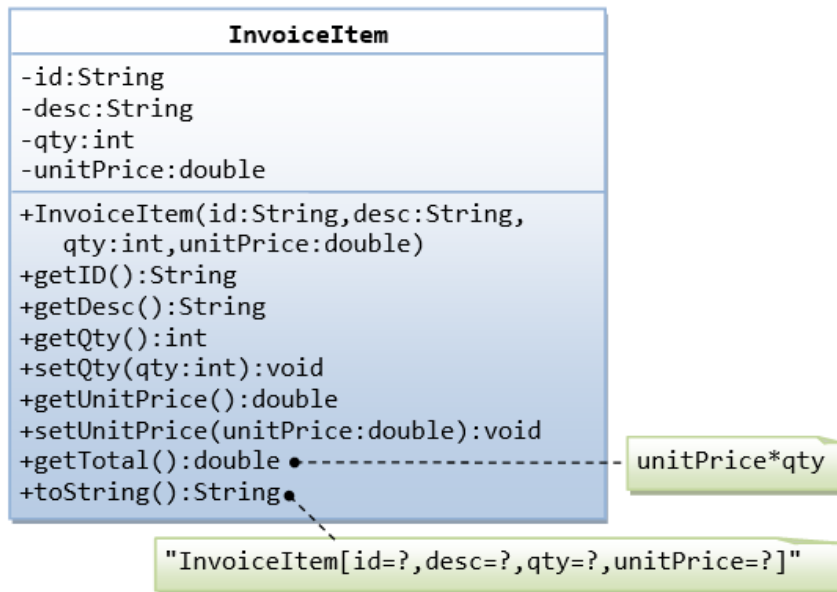
```

Command window
Employee[id = 8, name = Peter Tan, salary = 2500]
2 Employee[id = 8, name = Peter Tan, salary = 999]
id is: 8
4 firstname is: Peter
lastname is: Tan
6 salary is: 999
name is: Peter Tan
8 annual salary is: 11988
1098
10 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:



```

1 public class TestMain {
2     public static void main(String[] args) {
3         // Test constructor and toString()
4         InvoiceItem inv1 = new InvoiceItem("A101", "Pen Red", 888, 0.08);
5         System.out.println(inv1); // toString();
6
7         // Test Setters and Getters
8         inv1.setQty(999);
9         inv1.setUnitPrice(0.99);
10        System.out.println(inv1); // toString();
11        System.out.println("id is: " + inv1.getID());
12        System.out.println("desc is: " + inv1.getDesc());
13        System.out.println("qty is: " + inv1.getQty());
14        System.out.println("unitPrice is: " + inv1.getUnitPrice());
15
16        // Test getTotal()
17        System.out.println("The total is: " + inv1.getTotal());
18    }
19 }
  
```

The expected output is:

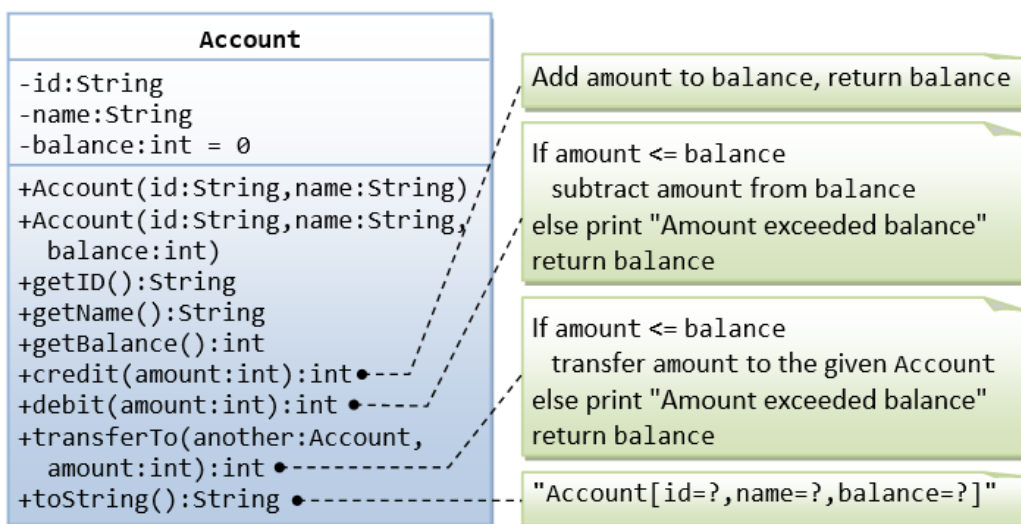
```

Command window
1 InvoiceItem[id = A101, desc = Pen Red, qty = 888, unitPrice = 0.08]
  InvoiceItem[id = A101, desc = Pen Red, qty = 999, unitPrice = 0.99]
3 id is: A101
  desc is: Pen Red
5 qty is: 999
  unitPrice is: 0.99
7 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:

```

1 public class TestMain {
2     public static void main(String[] args) {
3         // Test constructor and toString()
4         Account account1 = new Account("A101", "Tan Ah Teck", 88);
5         System.out.println(account1); // toString();
6         Account account2 = new Account("A102", "Kumar"); // default balance
7         System.out.println(account2);
8
9         // Test Getters
10        System.out.println("ID: " + account1.getID());
11        System.out.println("Name: " + account1.getName());

```



```

13      System.out.println("Balance: " + account1.getBalance());
14
15      // Test credit() and debit()
16      account1.credit(100);
17      System.out.println(account1);
18      account1.debit(50);
19      System.out.println(account1);
20      account1.debit(500); // debit() error
21      System.out.println(account1);
22
23      // Test transfer()
24      account1.transferTo(account2, 100); // toString()
25      System.out.println(account1);
26      System.out.println(account2);
27  }

```

The expected output is:

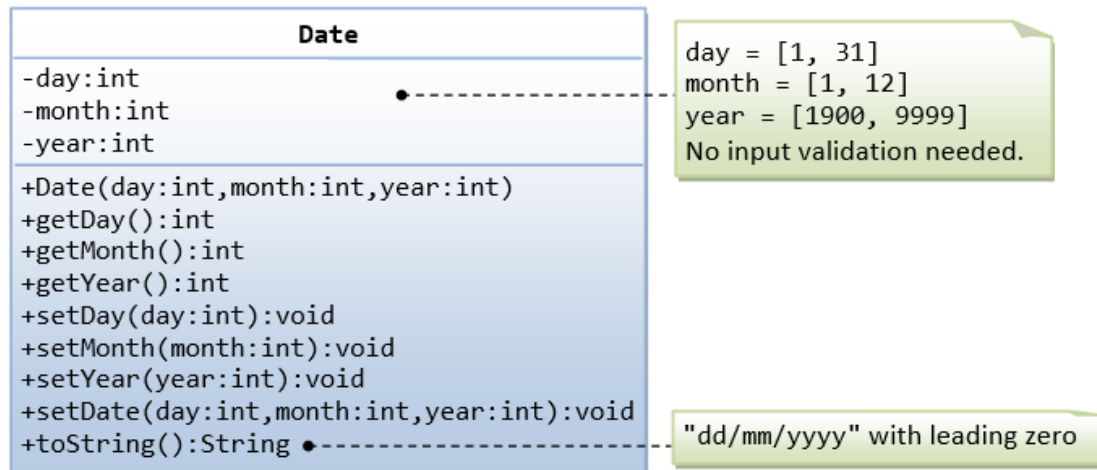
```

Command window
1  Account[id = A101,name = Tan Ah Teck,balance = 88]
2  Account[id = A102, name = Kumar, balance = 0]
3  ID: A101
4  Name: Tan Ah Teck
5  Balance: 88
6  Account[id = A101, name = Tan Ah Teck, balance = 188]
7  Account[id = A101, name = Tan Ah Teck, balance = 138]
8  Amount exceeded balance
9  Account[id = A101, name = Tan Ah Teck, balance = 138]
10 Account[id = A101, name = Tan Ah Teck, balance = 38]
11 Account[id = A102, name = Kumar, balance = 100]


```

## 1.7 The Date Class

A class called Date, which models a calendar date, is designed as shown in the following class diagram. Write the Date class.



Below is a test driver to test the Date class:



```

1  public class TestMain {
2      public static void main(String [] args) {
3          // Test constructor and toString()
4          Date date1 = new Date(1, 2, 2014);
5          System.out.println(date1); // toString()
6
7          // Test Setters and Getters
8          date1.setMonth(12);
9          date1.setDay(9);
10         date1.setYear(2099);
11         System.out.println(date1); // toString()
12         System.out.println("Month: " + date1.getMonth());
13         System.out.println("Day: " + date1.getDay());
14         System.out.println("Year: " + date1.getYear());
15
16         // Test setDate()
17         date1.setDate(3, 4, 2016);
18         System.out.println(date1); // toString()
19     }
20 }
  
```

The expected output is:

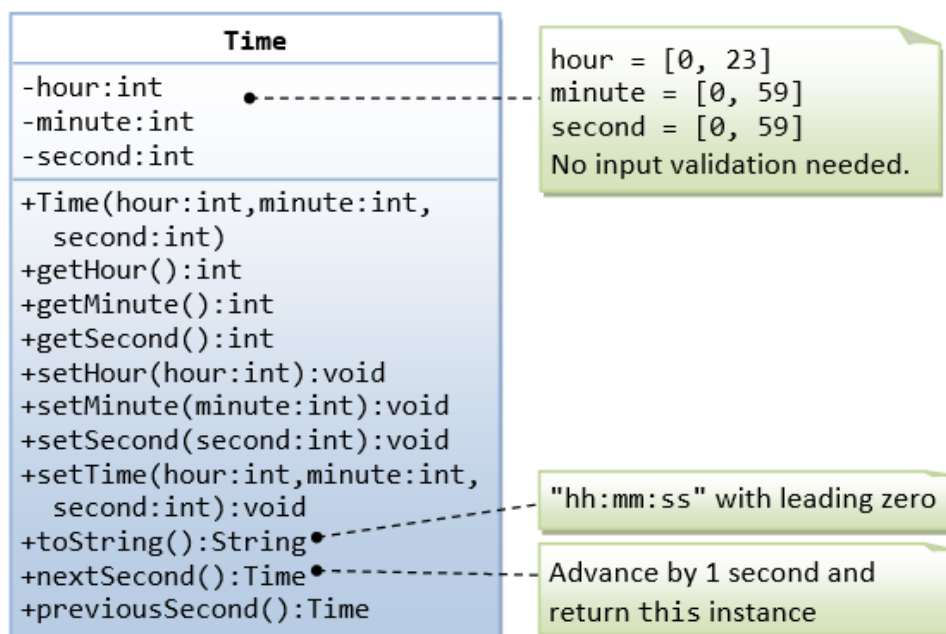
```

Command window
1 01/02/2014
2 09/12/2099
   Month: 12
4   Day: 9
   Year: 2099
6 03/04/2016

```

## 1.8 The Time Class

A class called `Time`, which models a time instance, is designed as shown in the following class diagram. The methods `nextSecond()` and `previousSecond()` shall advance or rewind this instance by one second, and return this instance, so as to support chaining operation such as `time1.nextSecond().nextSecond()`. Write the `Time` class.



Below is a test driver for testing the `Time` class:



```

public class TestMain {
2   public static void main(String[] args) {
    // Test constructors and toString()
4   Time time1 = new Time(1, 2, 3);
    System.out.println(time1); // toString()

6   // Test Setters and Getters
8   time1.setHour(4);

```



```

10    time1.setMinute(5);
    time1.setSecond(6);
    System.out.println(time1); // toString()
12    System.out.println("Hour: " + time1.getHour());
    System.out.println("Minute: " + time1.getMinute());
14    System.out.println("Second: " + time1.getSecond());

16    // Test setTime()
    time1.setTime(23, 59, 58);
18    System.out.println(time1); // toString()

20    // Test nextSecond();
    System.out.println(time1.nextSecond());
22    System.out.println(time1.nextSecond().nextSecond());

24    // Test previousSecond()
    System.out.println(time1.previousSecond());
26    System.out.println(time1.previousSecond().previousSecond());
    }
28 }

```

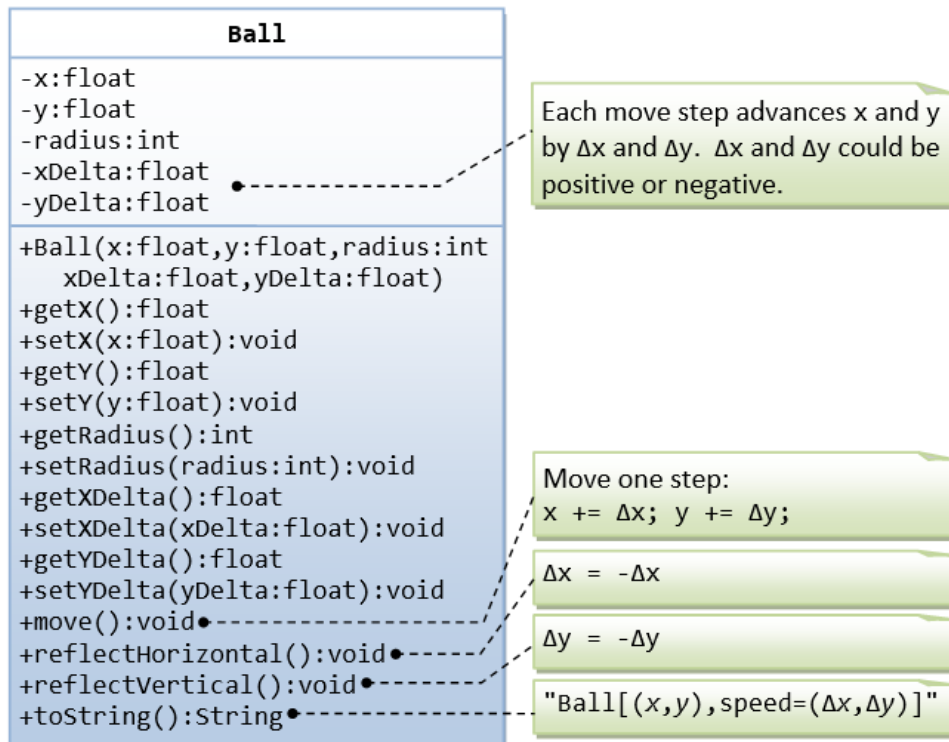
The expected output is:

Command window	
	01:02:03
2	04:05:06
	Hour: 4
4	Minute: 5
	Second: 6
6	23:59:58
	23:59:59
8	00:00:01
	00:00:00
10	23:59:58

## 1.9 The Ball Class

A class called Ball, which models a bouncing ball, is designed as shown in the following class diagram. It contains its radius,  $x$  and  $y$  position. Each move-step advances the  $x$  and  $y$  by  $\Delta x$  and  $\Delta y$ , respectively.  $\Delta x$  and  $\Delta y$  could be positive or negative. The *reflectHorizontal()* and *reflectVertical()* methods could be used to bounce the ball off the walls. Write the Ball class. Study the test driver on how the ball bounces.





Below is a test driver:



```

public class TestMain {
2   public static void main(String[] args) {
    // Test constructor and toString()
4   Ball ball = new Ball(1.1f, 2.2f, 10, 3.3f, 4.4f);
    System.out.println(ball); // toString()

6   // Test Setters and Getters
8   ball.setX(80.0f);
    ball.setY(35.0f);
10  ball.setRadius(5);
    ball.setXDelta(4.0f);
12  ball.setYDelta(6.0f);
    System.out.println(ball); // toString()
14  System.out.println("x is: " + ball.getX());
    System.out.println("y is: " + ball.getY());
16  System.out.println("radius is: " + ball.getRadius());
    System.out.println("xDelta is: " + ball.getXDelta());
18  System.out.println("yDelta is: " + ball.getYDelta());

20  // Bounce the ball within the boundary
    float xMin = 0.0f;
22  float xMax = 100.0f;
    float yMin = 0.0f;
24  float yMax = 50.0f;
  
```



```

26     ball.move();
    System.out.println(ball);
28     float xNew = ball.getX();
    float yNew = ball.getY();
30     int radius = ball.getRadius();
    // Check boundary value to bounce back
32     if ((xNew + radius) > xMax || (xNew - radius) < xMin) {
        ball.reflectHorizontal();
34     }
    if ((yNew + radius) > yMax || (yNew - radius) < yMin) {
36     ball.reflectVertical();
    }
38 }
40 }

```

The expected output is:

```

Command window
Ball[(1.1, 2.2), speed = (3.3, 4.4)]
2 Ball[(80.0, 35.0), speed = (4.0, 6.0)]
x is: 80.0
4 y is: 35.0
radius is: 5
6 xDelta is: 4.0
yDelta is: 6.0
8 Ball[(84.0, 41.0), speed = (4.0, 6.0)]
Ball[(88.0, 47.0), speed = (4.0, 6.0)]
10 Ball[(92.0, 41.0), speed = (4.0, -6.0)]
Ball[(96.0, 35.0), speed = (4.0, -6.0)]
12 Ball[(92.0, 29.0), speed = (-4.0, -6.0)]
Ball[(88.0, 23.0), speed = (-4.0, -6.0)]
14 Ball[(84.0, 17.0), speed = (-4.0, -6.0)]
Ball[(80.0, 11.0), speed = (-4.0, -6.0)]
16 Ball[(76.0, 5.0), speed = (-4.0, -6.0)]
Ball[(72.0, -1.0), speed = (-4.0, -6.0)]
18 Ball[(68.0, 5.0), speed = (-4.0, 6.0)]
Ball[(64.0, 11.0), speed = (-4.0, 6.0)]
20 Ball[(60.0, 17.0), speed = (-4.0, 6.0)]
Ball[(56.0, 23.0), speed = (-4.0, 6.0)]
22 Ball[(52.0, 29.0), speed = (-4.0, 6.0)]

```

### Try

Modify the constructor to take in speed and direction (in polar coordinates) instead of  $\Delta x$  and  $\Delta y$  (in cartesian coordinates), which is more convenient for the users.

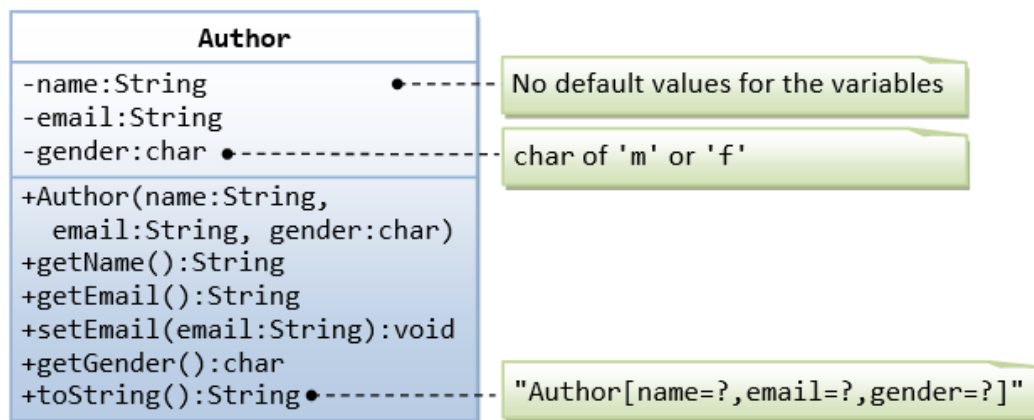


```
public Ball(float x, float y, int radius, int speed, int
    ↪ directionInDegree)
```

## 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]"

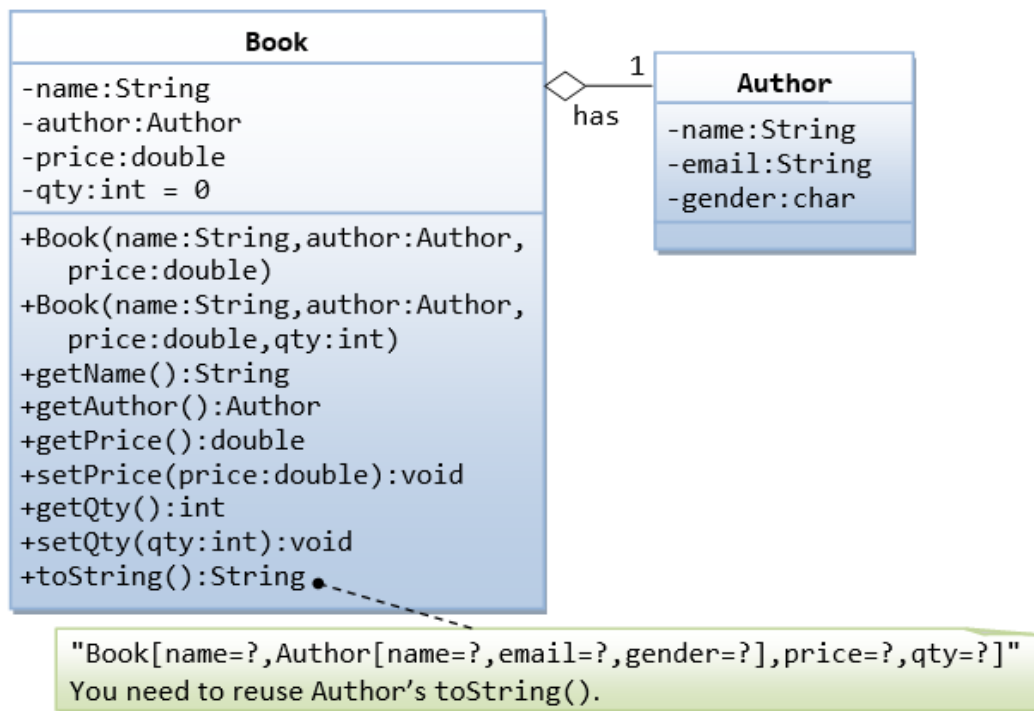
Write the Author class. Also write a test driver called TestAuthor to test all the public methods, e.g.,



```

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

```



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) { .....
    ↪ }
2 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.,



```

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

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

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

17 // Use an anonymous instance of Author to construct a Book instance
19 Book anotherBook = new Book("more Java",
  new Author("Paul Tan", "paul@somewhere.com", 'm'), 29.95);
21 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,

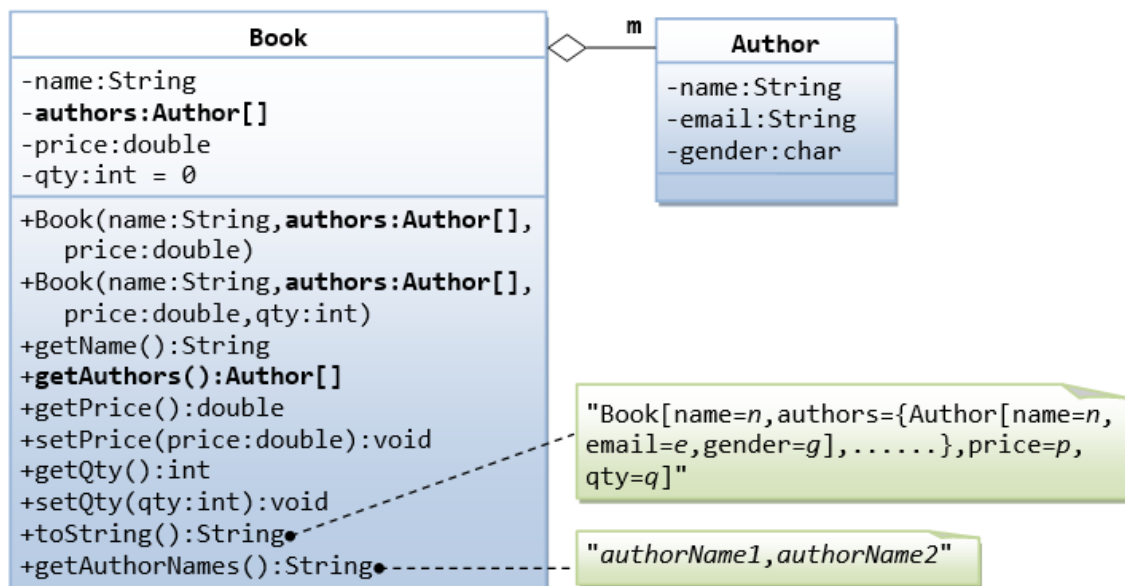


```

1  public String getAuthorName() {
    return author.getName(); // cannot use author.name as name is
                             ↪ private in Author class
3  }

```

## 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

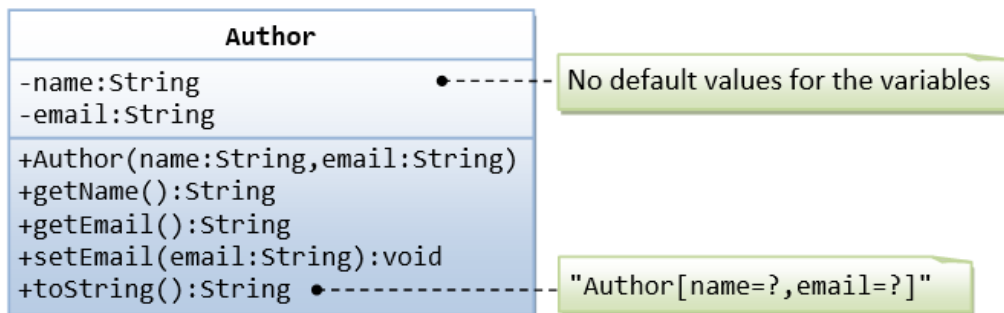


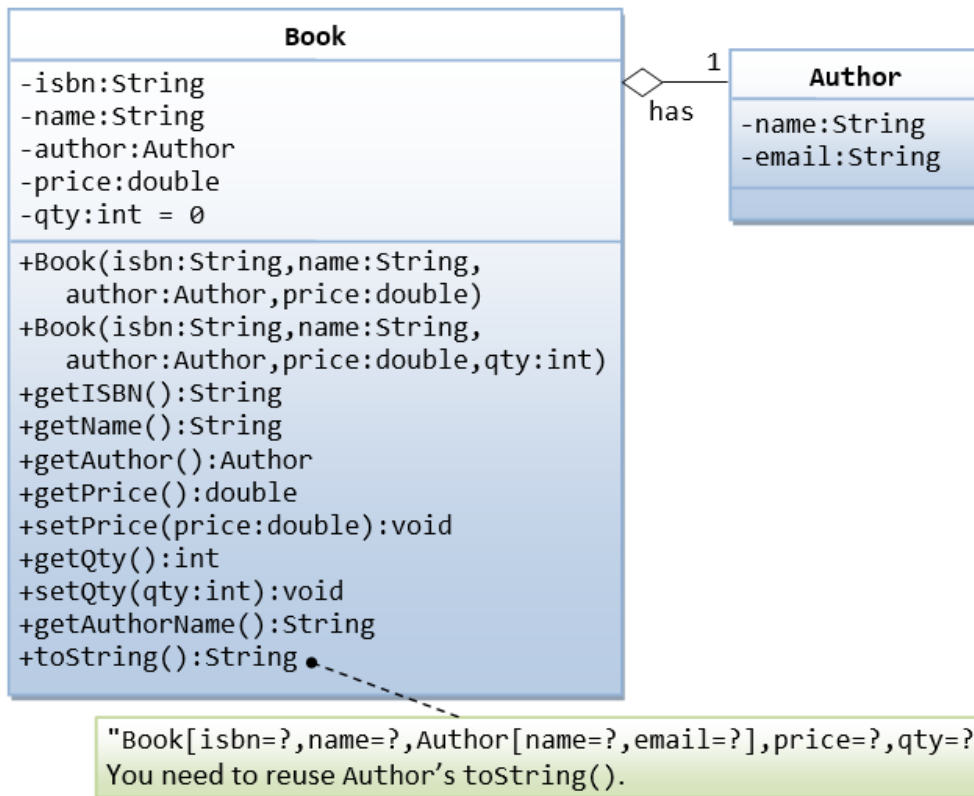
```
// Declare and allocate an array of Authors
2 Author[] authors = new Author[2];
  authors[0] = new Author("Tan Ah Teck", "AhTeck@somewhere.com", 'm');
4  authors[1] = new Author("Paul Tan", "Paul@nowhere.com", 'm');

6 // Declare and allocate a Book instance
  Book javaDummy = new Book("Java for Dummy", authors, 19.99, 99);
8 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:



```

public class TestMain {
2   public static void main(String[] args) {
    // Test Author class
4   Author author1 = new Author("Tan Ah Teck", "ahteck@nowhere.com");
    System.out.println(author1);
6
    author1.setEmail("ahteck@somewhere.com");
8   System.out.println(author1);
    System.out.println("name is: " + author1.getName());
10   System.out.println("email is: " + author1.getEmail());

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

16   book1.setPrice(9.9);
    book1.setQty(99);
18   System.out.println(book1);
    System.out.println("isbn is: " + book1.getISBN());
20   System.out.println("name is: " + book1.getName());
    System.out.println("price is: " + book1.getPrice());
22   System.out.println("qty is: " + book1.getQty());
  }
}
  
```





```

System.out.println("author is: " + book1.getAuthor()); // Author's
    ↪ toString()
24 System.out.println("author's name: " + book1.getAuthorName());
System.out.println("author's name: " + book1.getAuthor().getName())
    ↪ ;
26 System.out.println("author's email: " + book1.getAuthor().getEmail
    ↪ ());
    }
28 }

```

The expected output is:

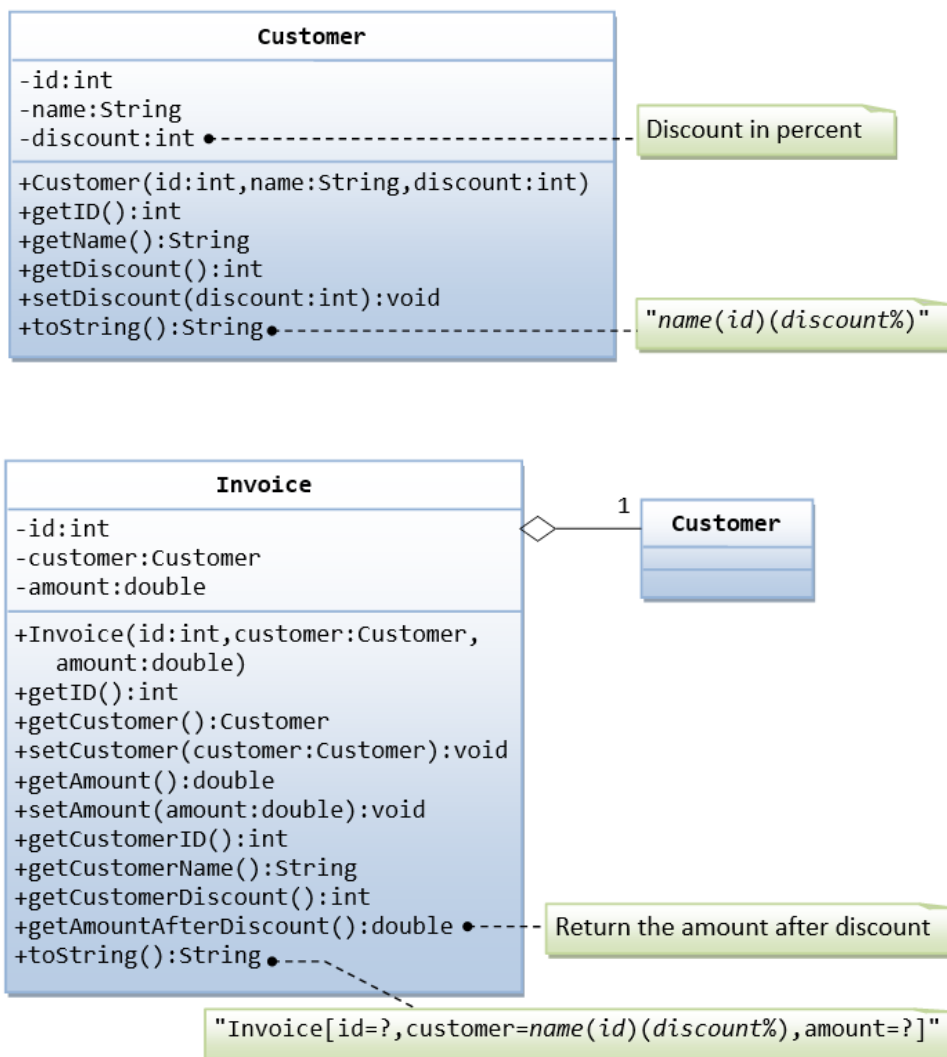
```

Command window
Author[name = Tan Ah Teck, email = ahteck@nowhere.com]
2 Author[name = Tan Ah Teck, email = ahteck@somewhere.com]
name is: Tan Ah Teck
4 email is: ahteck@somewhere.com
Book[isbn = 12345,name = Java for dummies, Author[name = Tan Ah Teck,
    email = ahteck@somewhere.com], price = 8.8, qty = 88]
6 Book[isbn = 12345, name = Java for dummies, Author[name = Tan Ah Teck,
    email = ahteck@somewhere.com], price = 9.9, qty = 99]
isbn is: Java for dummies
8 name is: Java for dummies
price is: 9.9
10 qty is: 99
author is: Author[name = Tan Ah Teck, email = ahteck@somewhere.com]
12 author's name: Tan Ah Teck
author's name: Tan Ah Teck
14 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:



```

public class TestMain {
2   public static void main(String[] args) {
    // Test Customer class
4   Customer customer1 = new Customer(88, "Tan Ah Teck", 10);
    System.out.println(customer1); // Customer's toString()

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

12   // Test Invoice class
14   Invoice invoice1 = new Invoice(101, c1, 888.8);
    
```



```

16      System.out.println(inv1);
18      invoice1.setAmount(999.9);
18      System.out.println(invoice1);
20      System.out.println("id is: " + invoice1.getID());
20      System.out.println("customer is: " + invoice1.getCustomer()); //
      ↪ Customer's toString()
22      System.out.println("amount is: " + invoice1.getAmount());
22      System.out.println("customer's id is: " + invoice1.getCustomerID())
      ↪ ;
      System.out.println("customer's name is: " + invoice1.
      ↪ getCustomerName());
24      System.out.println("customer's discount is: " + invoice1.
      ↪ getCustomerDiscount());
      System.out.printf("amount after discount is: %.2f%n", inv1.
      ↪ getAmountAfterDiscount());
26  }
    }

```

The expected output is:

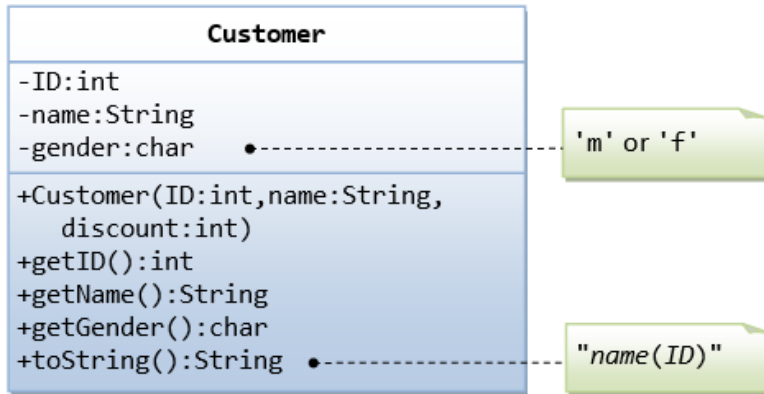
#### Command window

```

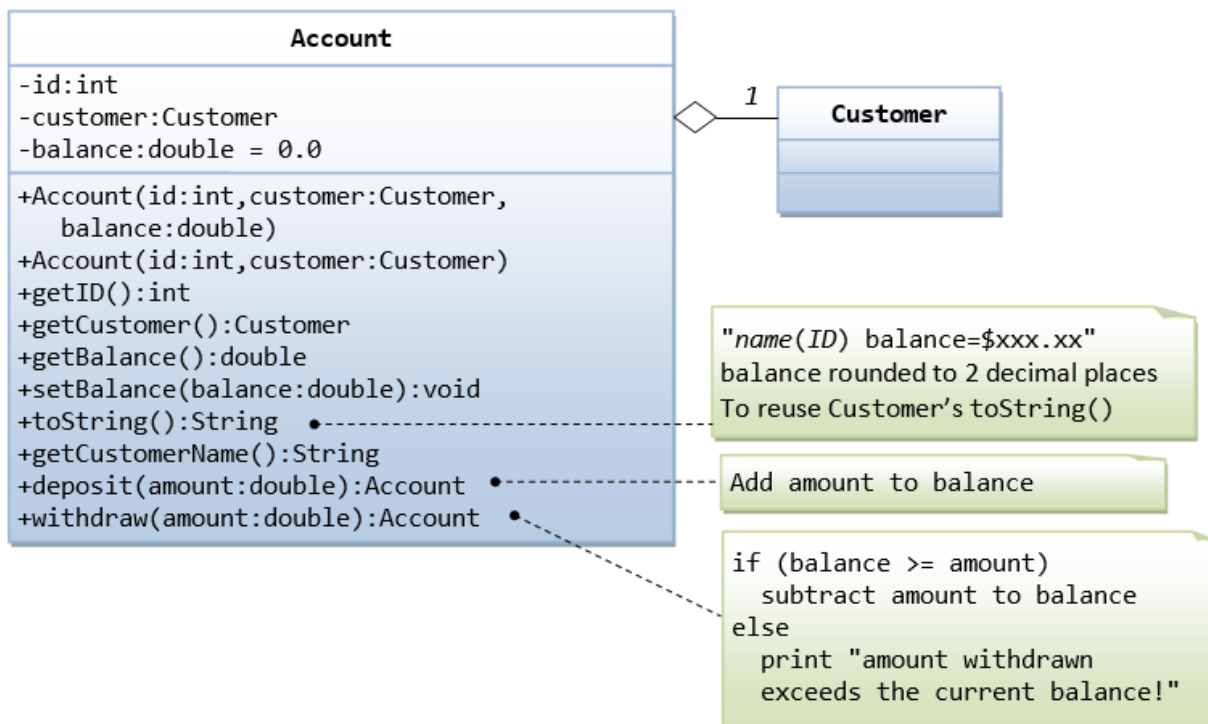
1  Tan Ah Teck(88)(10%)
   Tan Ah Teck(88)(8%)
3  id is: 88
   name is: Tan Ah Teck
5  discount is: 8
   Invoice[id = 101, customer = Tan Ah Teck(88)(8%), amount = 888.8]
7  Invoice[id = 101, customer = Tan Ah Teck(88)(8%), amount = 999.9]
   id is: 101
9  customer is: Tan Ah Teck(88)(8%)
   amount is: 999.9
11 customer's id is: 88
   customer's name is: Tan Ah Teck
13 customer's discount is: 8
   amount after discount is: 919.91

```

## 2.5 The Customer and Account classes



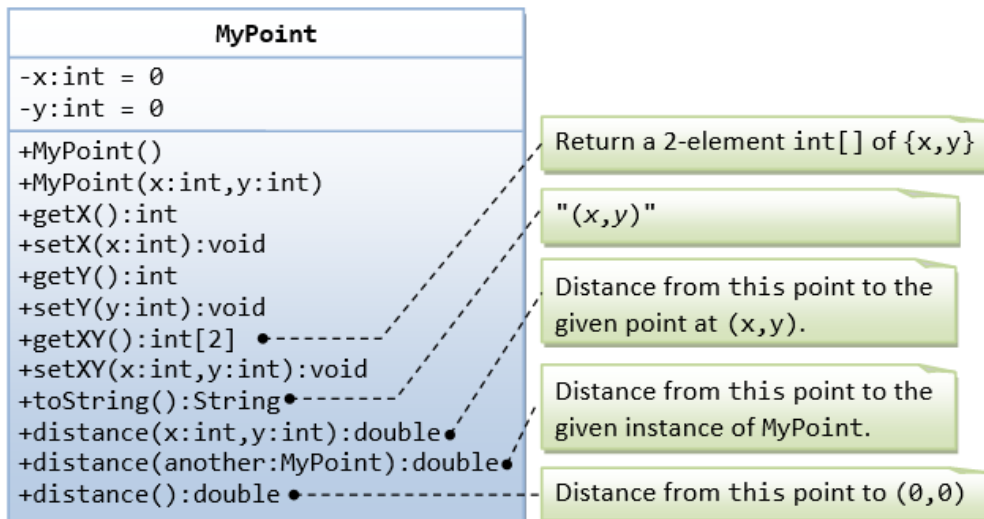
The Customer class models a customer is design as shown in the class diagram. Write the codes for the Customer class and a test driver to test all the public methods.



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 constructs 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.,



```

2 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.,



```

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

```

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



```

2  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



```

1  // Overloading method distance()
   // This version takes two ints as arguments
3  public double distance(int x, int y) {
   int xDiff = this.x - x;
5   int yDiff = .....
   return Math.sqrt(xDiff*xDiff + yDiff*yDiff);
7  }

9  // This version takes a MyPoint instance as argument
   public double distance(MyPoint another) {
11     int xDiff = this.x - another.x;
       .....
13     }

```



```

// Test program to test all constructors and public methods
2  MyPoint point1 = new MyPoint(); // Test constructor
   System.out.println(point1);    // Test toString()
4  point1.setX(8); // Test setters
   point1.setY(6);
6  System.out.println("x is: " + point1.getX()); // Test getters
   System.out.println("y is: " + point1.getY());
8  p1.setXY(3, 0); // Test setXY()
   System.out.println(point1.getXY()[0]); // Test getXY()
10  System.out.println(point1.getXY()[1]);

```



```

System.out.println(point1);
12
MyPoint point2 = new MyPoint(0, 4); // Test another constructor
14 System.out.println(point2);
    // Testing the overloaded methods distance()
16 System.out.println(point1.distance(point2)); // which version
    ↪ ?
    System.out.println(point2.distance(point1)); // which version
    ↪ ?
18 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), ..., (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
2 for (int i = 0; i < points.length; i++) {
    points[i] = new MyPoint(...); // Allocate each of MyPoint
    ↪ instances
4 }
    // use a loop to print all the points

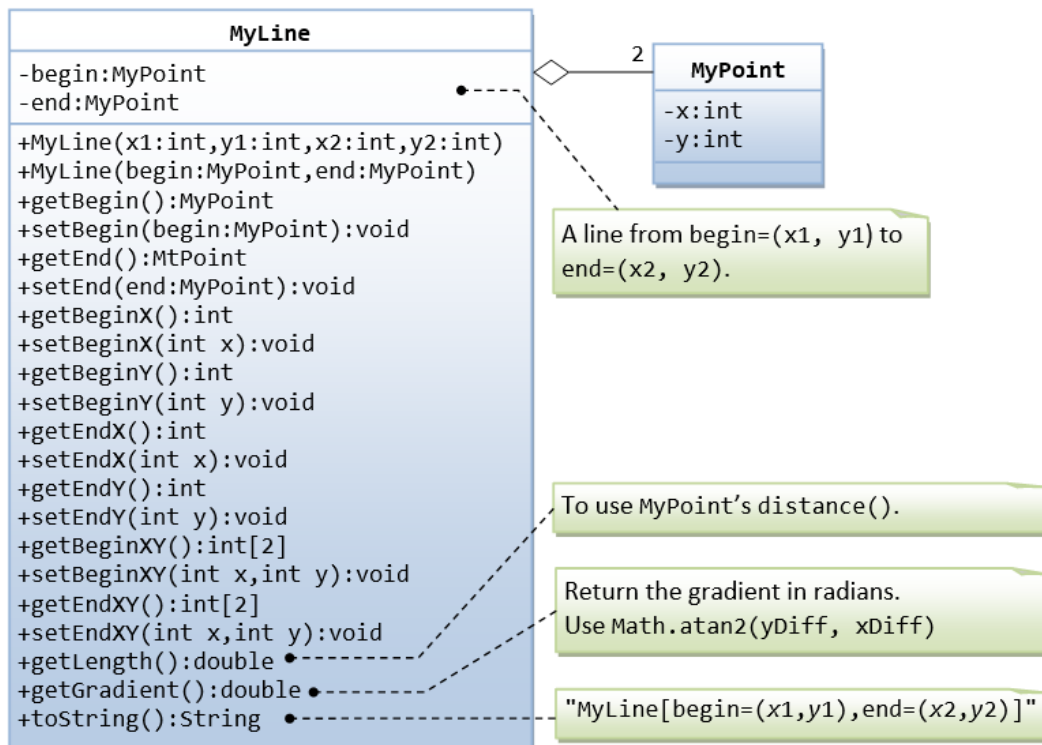
```

### Notes

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

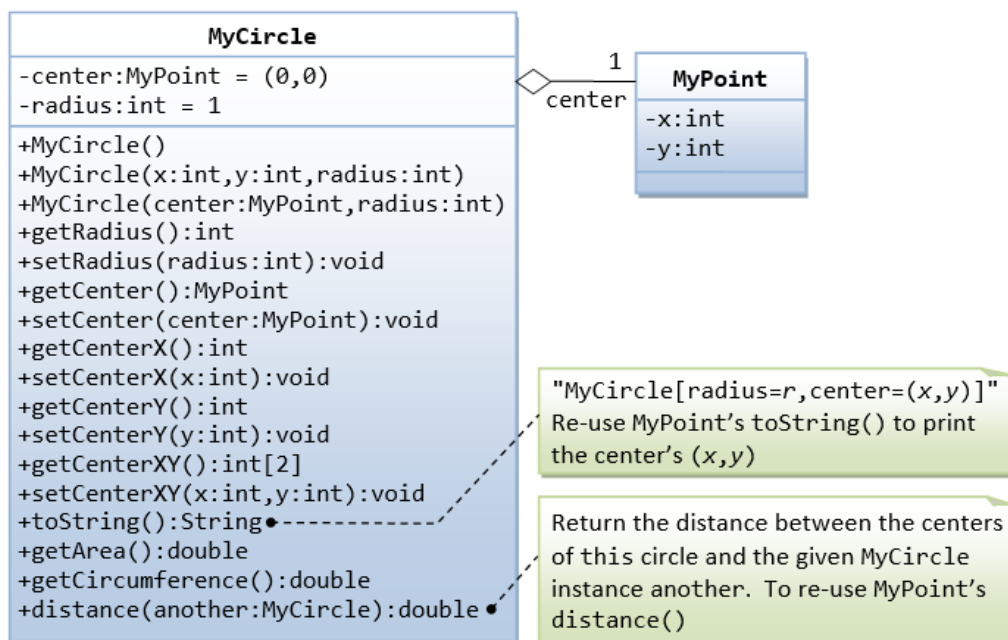
## 2.7 The MyLine and MyPoint Classes

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
2 public MyCircle(int x, int y, int radius) {
    // Need to construct an instance of MyPoint for the variable center
4     center = new MyPoint(x, y);
    this.radius = radius;
6 }
    public MyCircle(MyPoint center, int radius) {
8     // An instance of MyPoint already constructed by caller; simply
        ↪ assign.
        this.center = center;
10     .....
    }
12 public MyCircle() {
    center = new MyPoint(.....); // construct MyPoint instance
14     this.radius = .....
    }
16
    // Returns the x-coordinate of the center of this MyCircle
18 public int getCenterX() {
    return center.getX(); // cannot use center.x and x is private in
        ↪ MyPoint
20 }

```



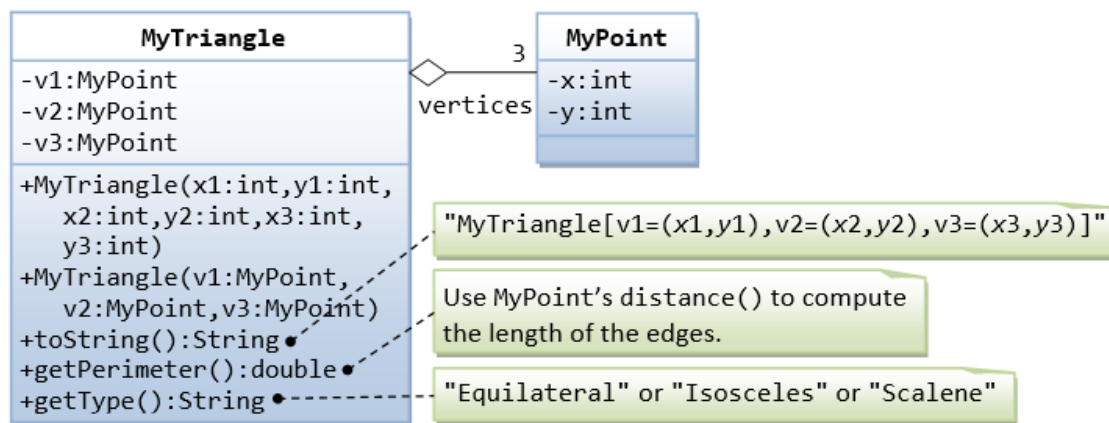
```

22 // Returns the distance of the center for this MyCircle and another
    ↪ MyCircle
    public double distance(MyCircle another) {
24     return center.distance(another.center); // use distance() of MyPoint
    }

```

## 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.