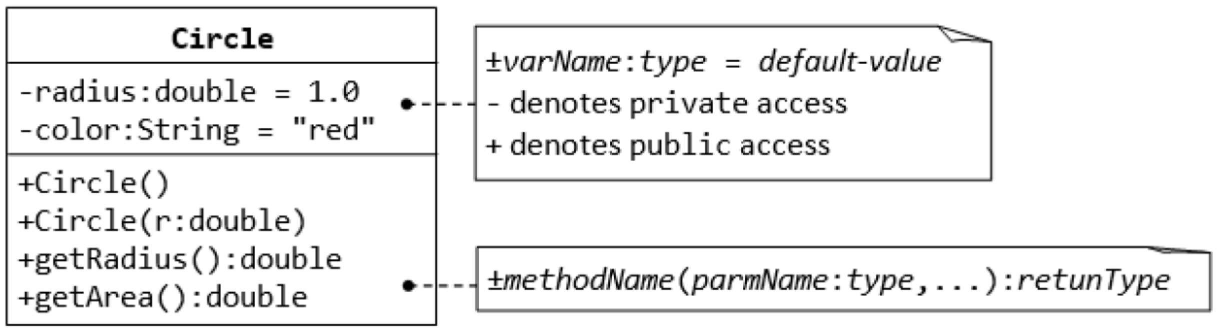
# CSC 302: OBJECT ORIENTED PROGRAMMING

# LAB EXERCISE

Muhammad Salis Ali
ms.ali@fud.edu.ng

# CSC 302: Object Oriented Programming (Assignment) 21/01/2020

## 1.1 The Circle Class (An Introduction to Classes and Objects)



A class called `Circle` is designed as shown in the class diagram above. 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:

```
/*
* 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;
   // The default constructor with no argument.
   // It sets the radius and color to their default value.
   public Circle() {
       radius = 1.0;
       color = "red";
   }
   // 2nd constructor with given radius, but with a default color
   public Circle(double r) {
       radius = r;
       color = "red";
   }
   // A public method for retrieving the radius
   public double getRadius() {
      return radius;
   }
   // A public method for computing the area of circle
   public double getArea() {
      return radius * radius * Math.PI;
   }
}
```

Compile "`Circle.java`". Can you run the `Circle` class? Why?

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

Let us write a test program called `TestCircle` (in another source file called `TestCircle.java`) which uses the Circle class, as follows:

```
public class TestCircle { // Save as "TestCircle.java"
    public static void main(String[] args) {
      // Declare an instance of Circle class called c1.
      // Construct the instance c1 by invoking the "default" constructor
      // which sets its radius and color to their default value.
      Circle c1 = new Circle();
      // Invoke public methods on instance c1, via dot operator.
      System.out.println("The circle has radius of " + c1.getRadius() + "
    and area of " + c1.getArea());
      // Declare an instance of class circle called c2.
      // Construct the instance c2 by invoking the second constructor
      // with the given radius and default color.
      Circle c2 = new Circle(2.0);
      // Invoke public methods on instance c2, via dot operator.
      System.out.println("The circle has radius of "
      + c2.getRadius() + " and area of " + c2.getArea());
    }
}
```

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

## 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
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.

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

Modify the test program to test this method.

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

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

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

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

```
Circle c4 = new Circle(); // construct an instance of Circle
c4.setRadius(5.0); // change radius
System.out.println("radius is: " + c4.getRadius()); // Print radius via
getter
c4.setColor(......); // Change color
System.out.println("color is: " + c4.getColor()); // Print color via getter
```

```
// You cannot do the following because setRadius() returns void,
// which cannot be printed.
System.out.println(c4.setRadius(4.0));
```

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

```
// Instance variable
private double radius;

// Constructor
public Circle(double radius) {
   this.radius = radius;   // "this.radius" refers to the instance variable
                           // "radius" refers to the method's parameter
   color = .......
}

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

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

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

```
// Return a description of this instance in the form of
// Circle[radius=r,color=c]
public String toString() {
   return "Circle[radius=" + radius + " color=" + color + "]";
}
```

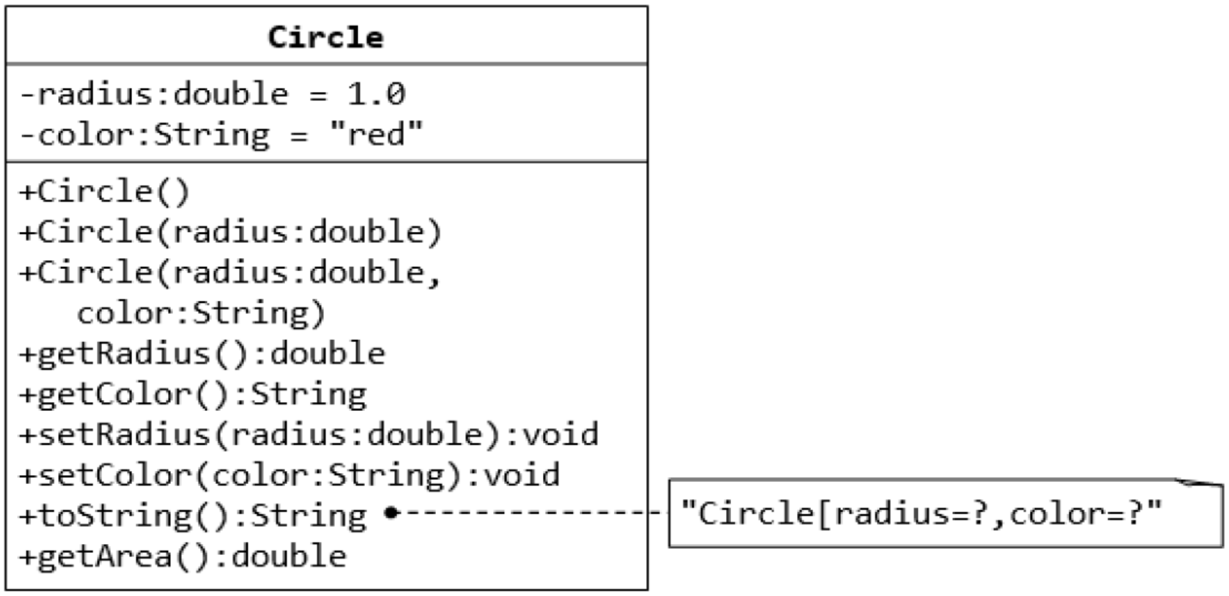
Try calling `toString()` method explicitly, just like any other method:

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

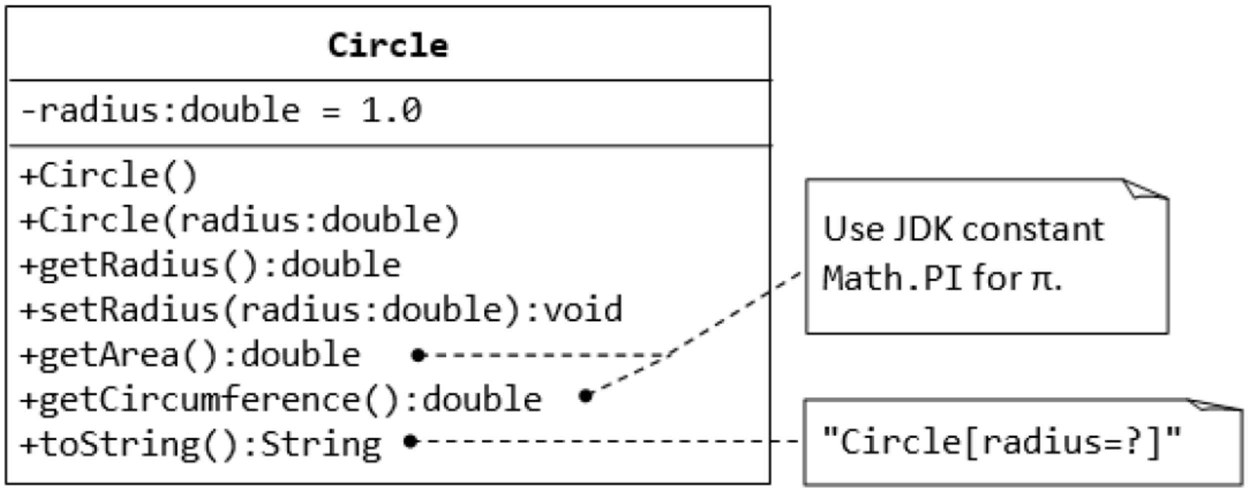
`toString()` is called implicitly when an instance is passed to `println()` method, for example,

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

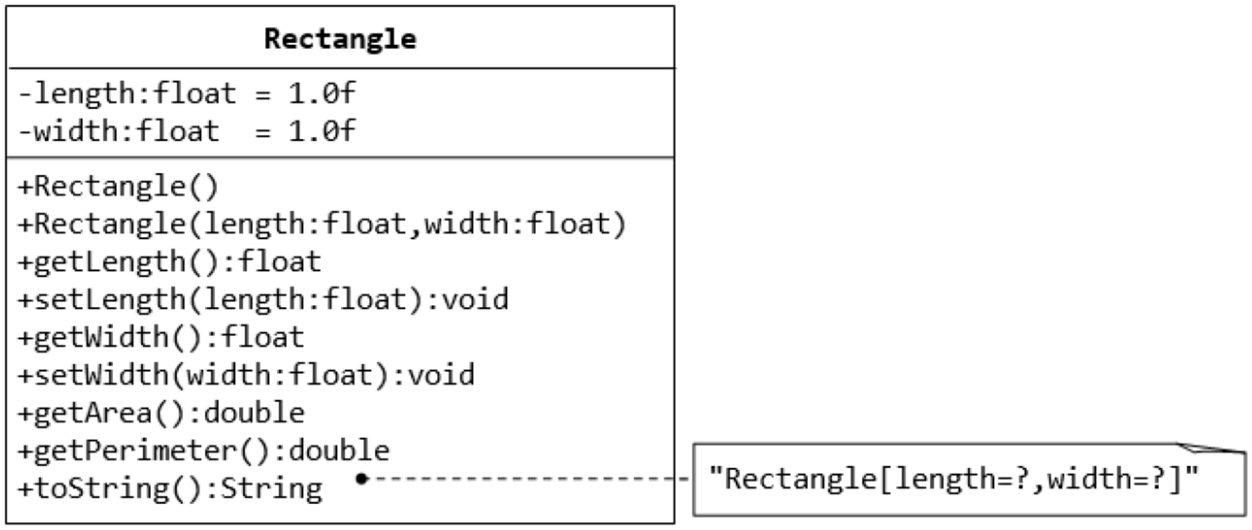
The final class diagram for the `Circle` class is as follows:



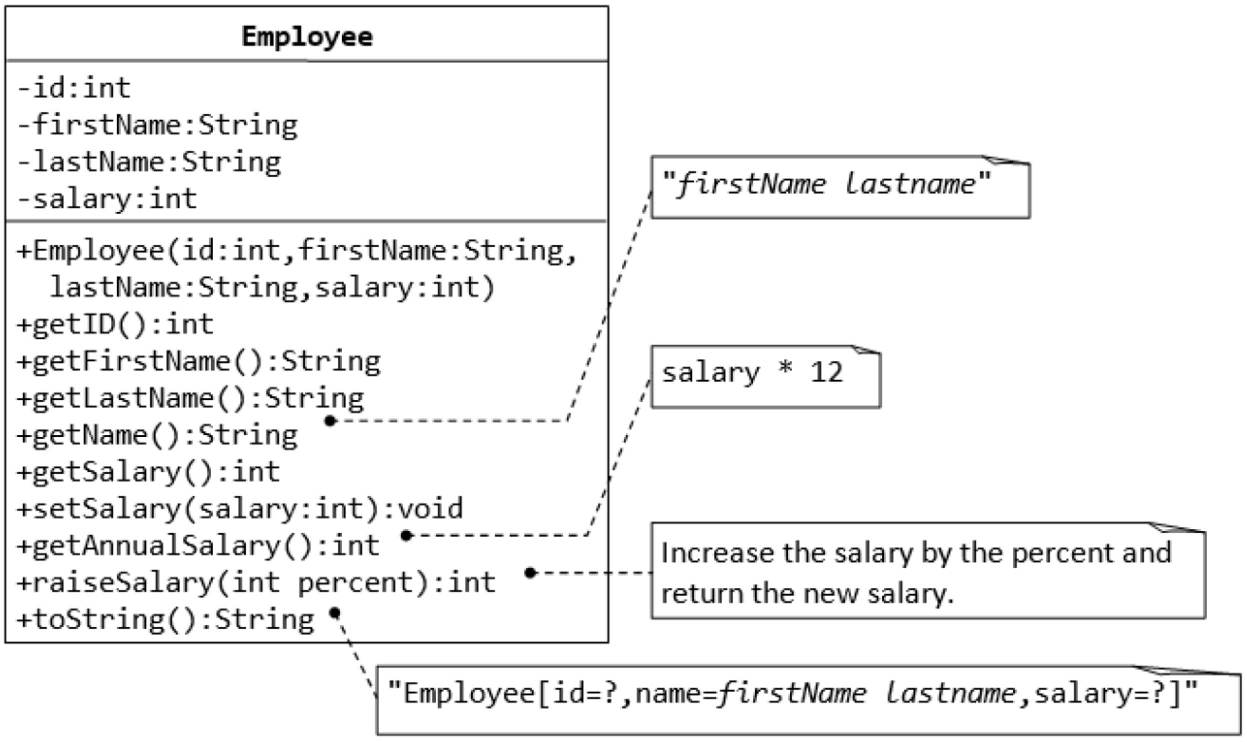
1.2 A Simplified `Circle` Class



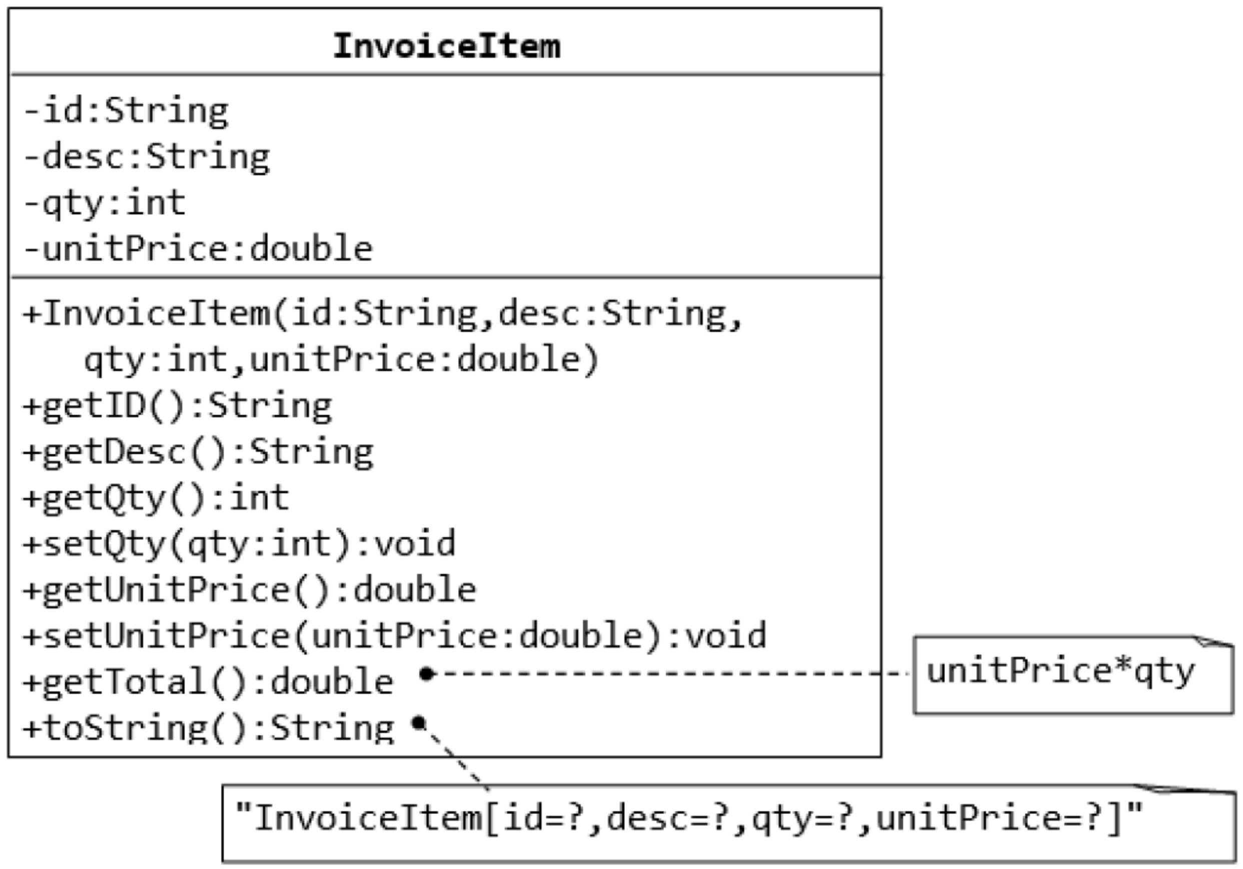
## 1.3 The `Rectangle` Class



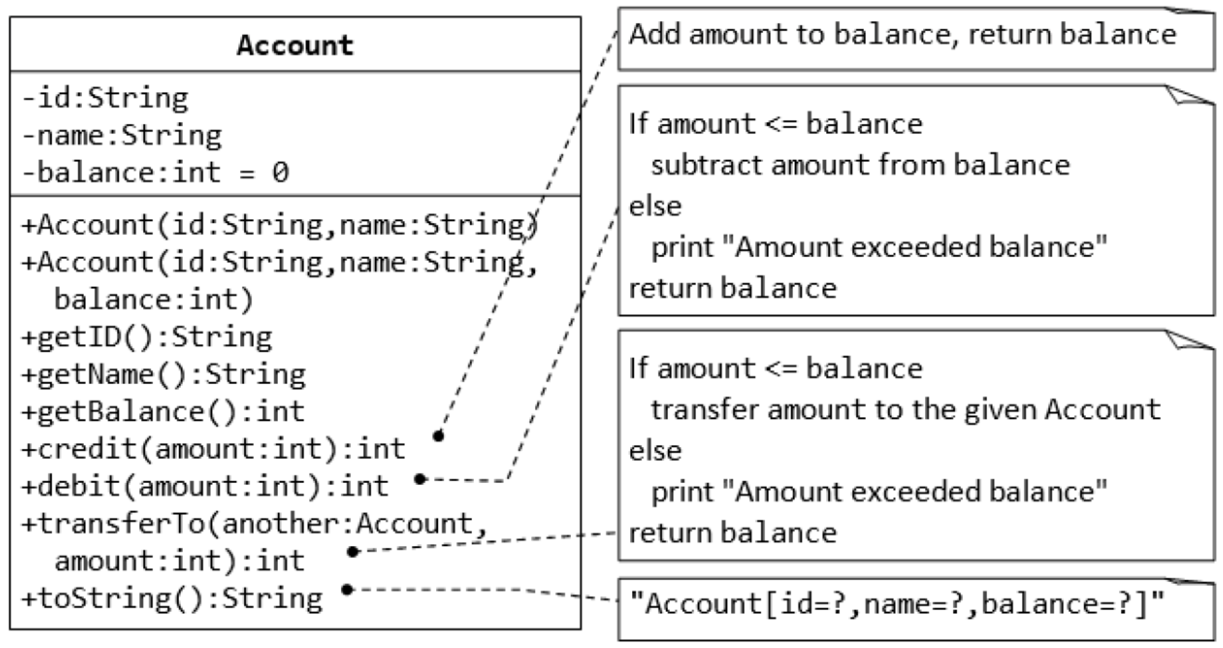
## 1.4 The `Employee` Class


Employee
-id:int
-firstName:String
-lastName:String
-salary:int
+Employee(id:int,firstName:String, lastName:String,salary:int)
+getID():int
+getFirstName():String
+getLastName():String
+getName():String
+getSalary():int
+setSalary(salary:int):void
+getAnnualSalary():int
+raiseSalary(int percent):int
+toString():String
"firstName lastname"
salary * 12
Increase the salary by the percent and return the new salary.
"Employee[id=?,name=firstName lastname,salary=?]"

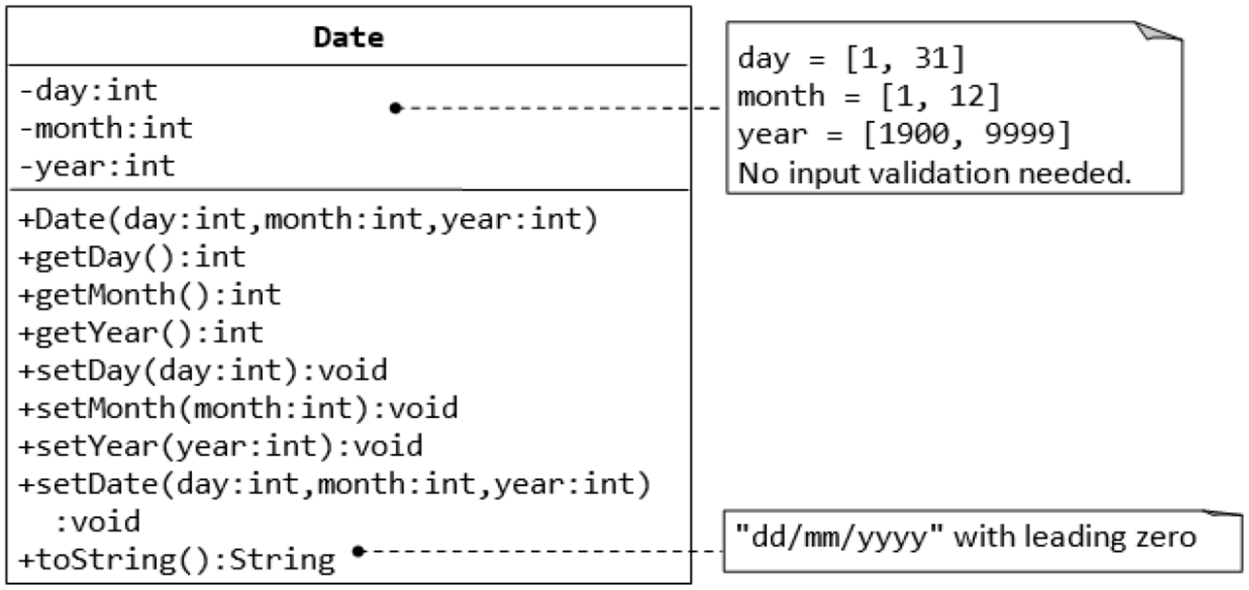
## 1.5 The `InvoiceItem` Class



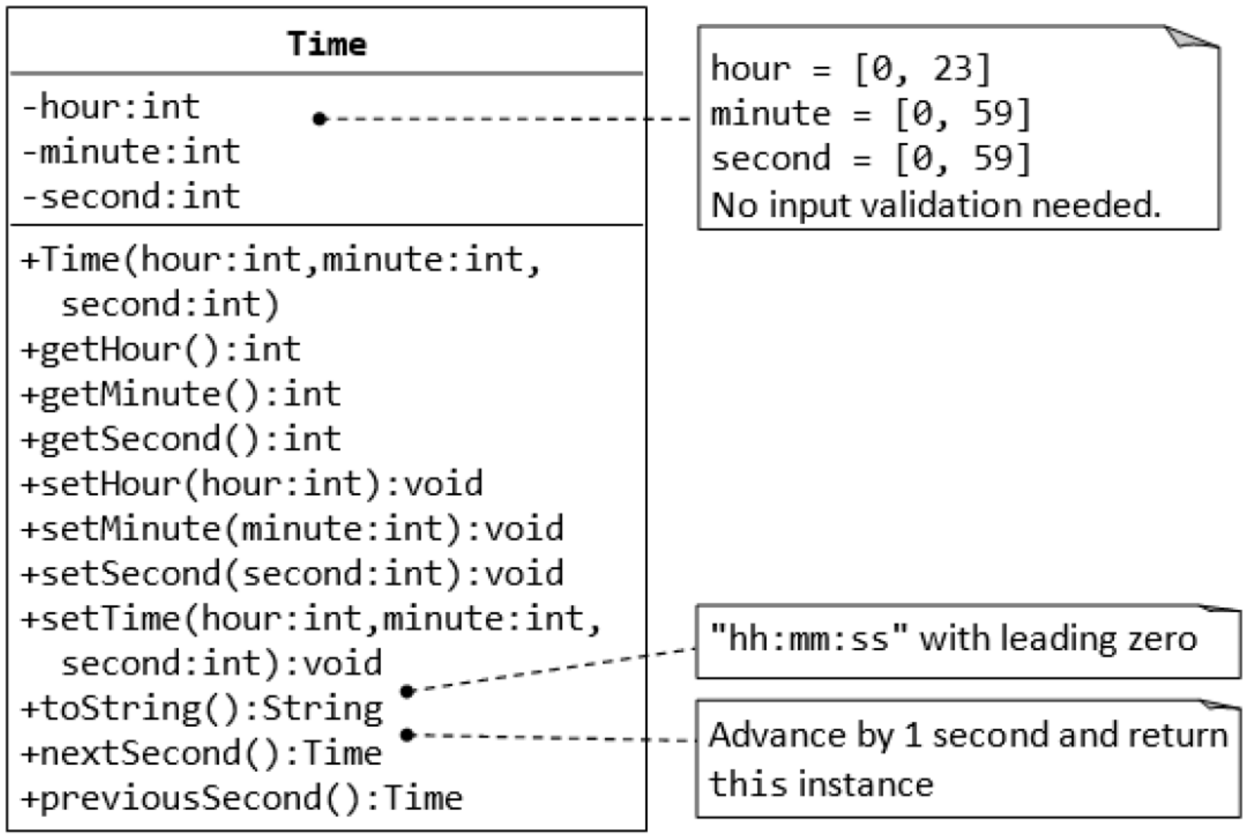
## 1.6 The `Account` Class


Account
-id:String
-name:String
-balance:int = 0
+Account(id:String,name:String)
+Account(id:String,name:String, balance:int)
+getID():String
+getName():String
+getBalance():int
+credit(amount:int):int
+debit(amount:int):int
+transferTo(another:Account, amount:int):int
+toString():String
Add amount to balance, return balance
If amount <= balance subtract amount from balance else print "Amount exceeded balance" return balance
If amount <= balance transfer amount to the given Account else print "Amount exceeded balance" return balance
"Account[id=?,name=?,balance=?]"

## 1.7 The `Date` Class

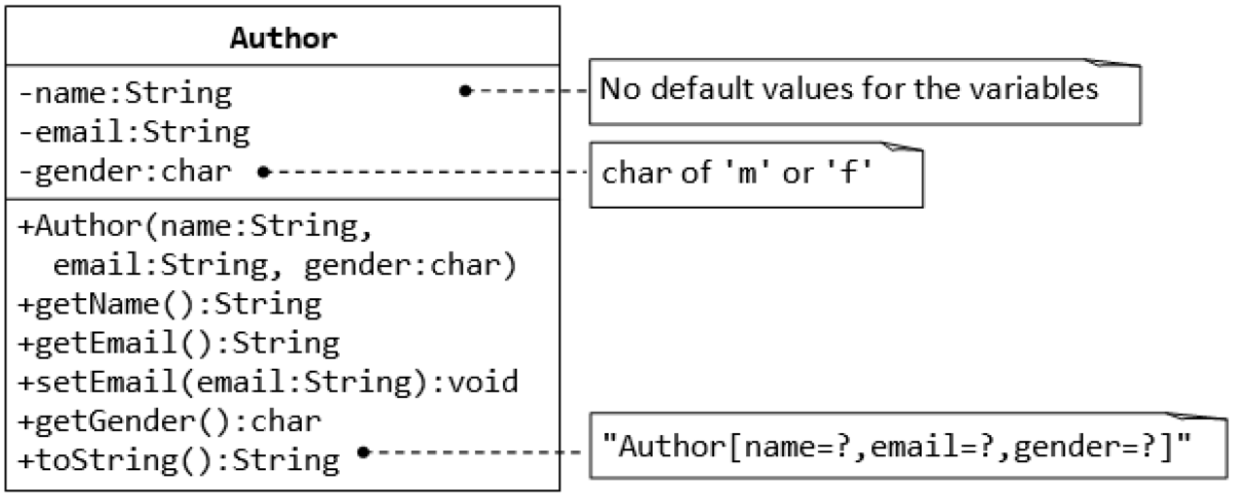


## 1.8 The `Time` Class


Time
-hour:int
-minute:int
-second:int
+Time(hour:int,minute:int,second:int)
+getHour():int
+getMinute():int
+getSecond():int
+setHour(hour:int):void
+setMinute(minute:int):void
+setSecond(second:int):void
+setTime(hour:int,minute:int,second:int):void
+toString():String
+nextSecond():Time
+previousSecond():Time
hour = [0, 23]
minute = [0, 59]
second = [0, 59]
No input validation needed.
"hh:mm:ss" with leading zero
Advance by 1 second and return this instance

## 2. Exercises on Composition

### 2.1 The Author and Book Classes (An Introduction to OOP Composition)


Author
-name:String
-email:String
-gender:char
+Author(name:String, email:String, gender:char)
+getName():String
+getEmail():String
+setEmail(email:String):void
+getGender():char
+toString():String
No default values for the variables
char of 'm' or 'f'
"Author[name=?,email=?,gender=?]"

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;

```
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.)

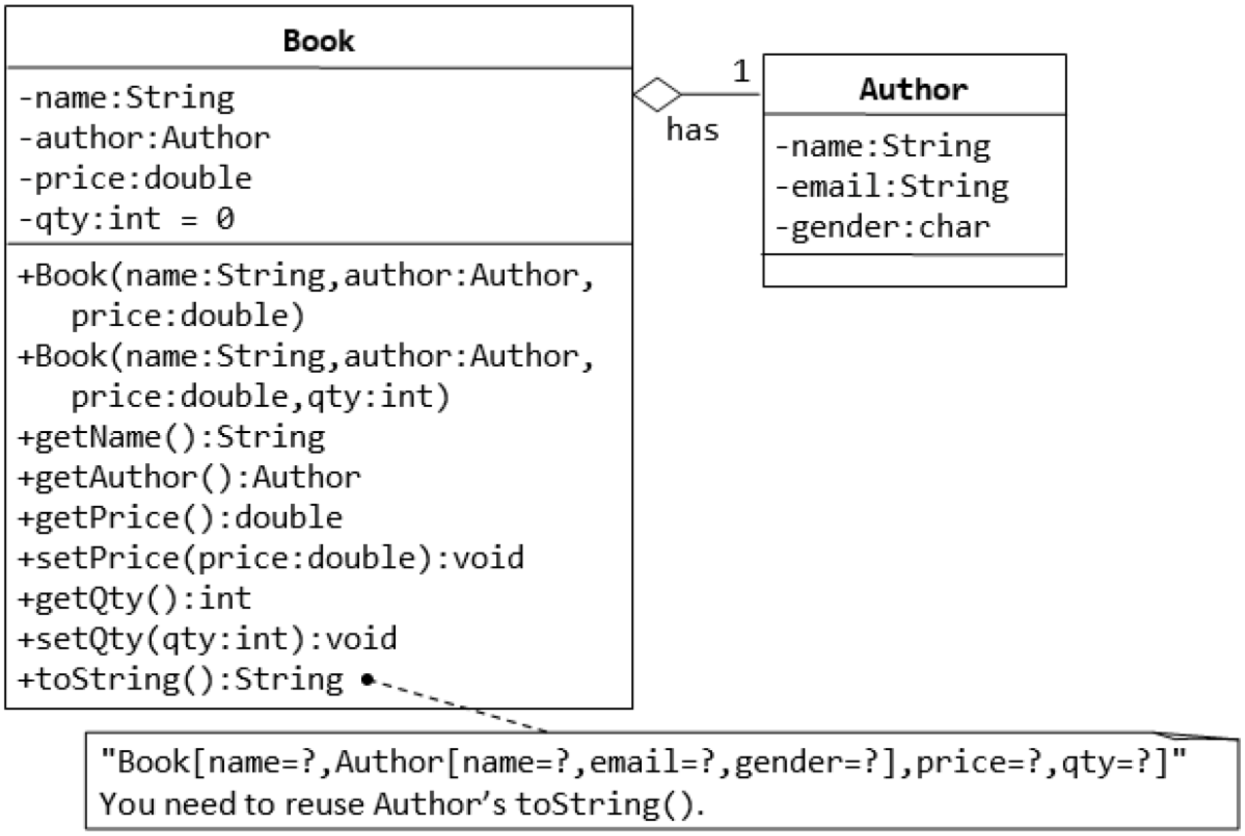
`public` getters/setters: `getName()`, `getEmail()`, `setEmail()`, and `getGender()`;

(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 gExerciseOOP_MyPolynomial.pngett
```


Book
-name:String
-author:Author
-price:double
-qty:int = 0
+Book(name:String,author:Author, price:double)
+Book(name:String,author:Author, price:double,qty:int)
+getName():String
+getAuthor():Author
+getPrice():double
+setPrice(price:double):void
+getQty():int
+setQty(qty:int):void
+toString():String
1
has
Author
-name:String
-email:String
-gender:char
"Book[name=?,Author[name=?,email=?,gender=?],price=?,qty=?]"
You need to reuse Author's toString().

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

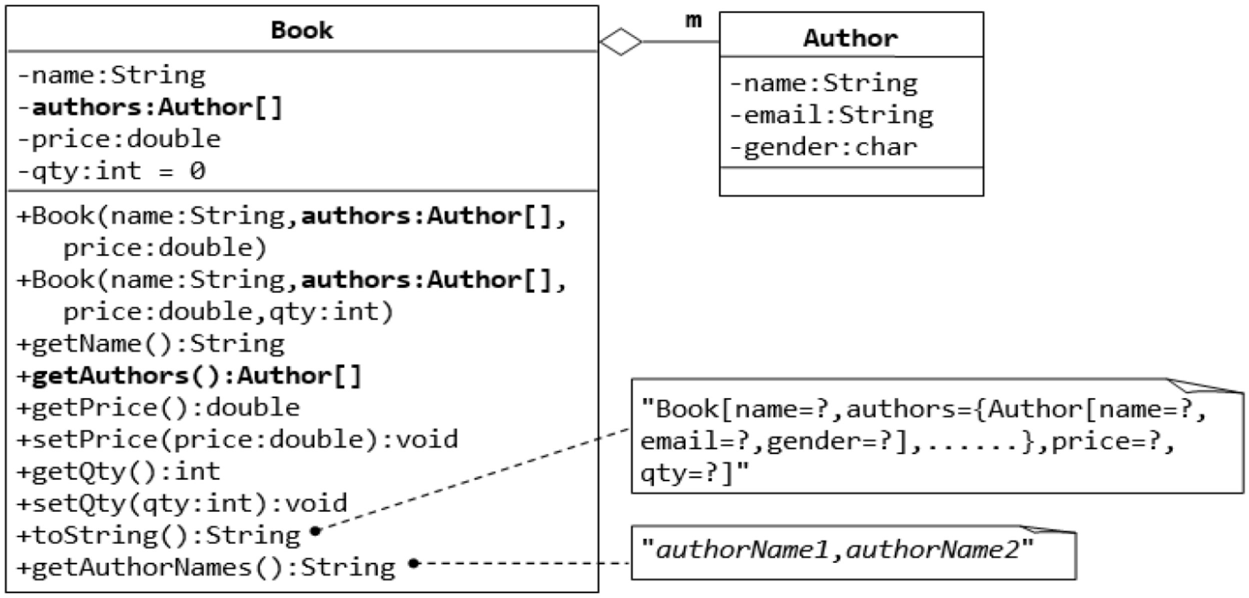
TRY:

1. Printing the `name` and `email` of the `author` from a `Book` instance. (Hint: `aBook.getAuthor().getName()`, `aBook.getAuthor().getEmail()`).

2. Introduce new methods called `getAuthorName()`, `getAuthorEmail()`, `getAuthorGender()` in the `Book` class to return the `name`, `email` and `gender` of the `author` of the `book`. For example,

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

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


Book
-name:String
-authors:Author[]
-price:double
-qty:int = 0
+Book(name:String,authors:Author[], price:double)
+Book(name:String,authors:Author[], price:double,qty:int)
+getName():String
+getAuthors():Author[]
+getPrice():double
+setPrice(price:double):void
+getQty():int
+setQty(qty:int):void
+toString():String
+getAuthorNames():String
m
Author
-name:String
-email:String
-gender:char
"Book[name=?,authors={Author[name=?, email=?,gender=?],......},price=?, qty=?]"
"authorName1,authorName2"

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 authors.

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 constructed, you cannot add or remove author.
- The `toString()` method shall return "`Book[name=?,authors={Author[name=?,email=?,gender=?],......},price=?,qty=?]`".

You are required to:

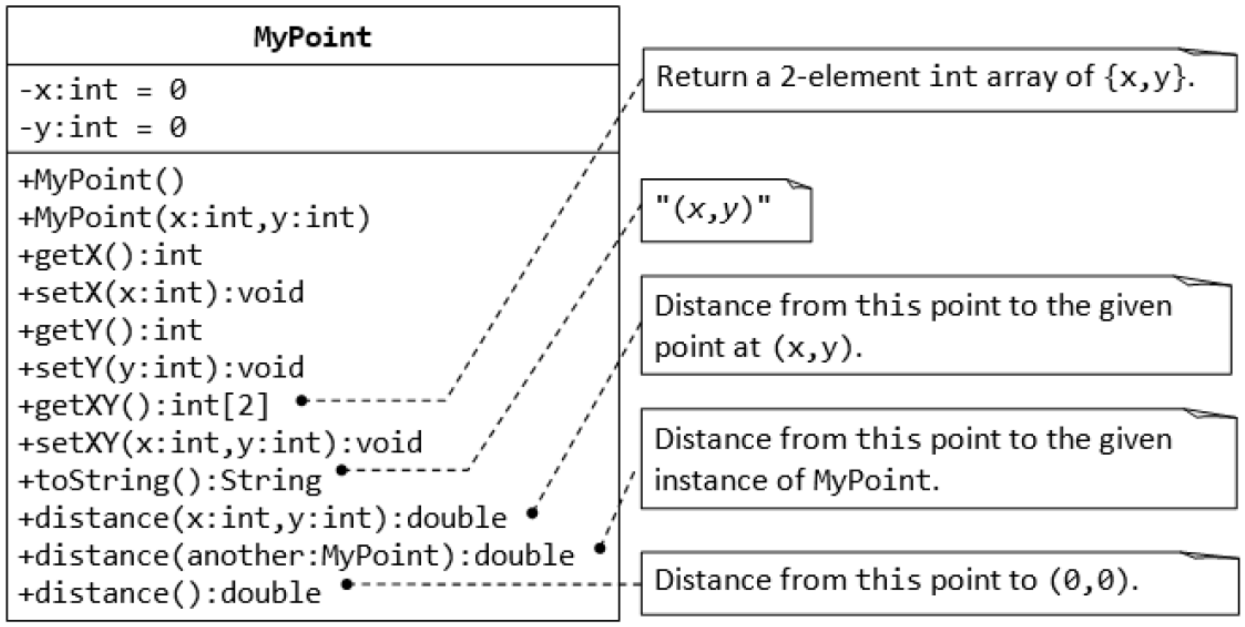
1. Write the code for the `Book` class. You shall re-use the `Author` class written earlier.
2. Write a test driver (called `TestBook`) to test the `Book` class.

**Hints:**

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

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

## 2.3 The MyPoint Class


MyPoint
-x:int = 0
-y:int = 0
+MyPoint()
+MyPoint(x:int,y:int)
+getX():int
+setX(x:int):void
+getY():int
+setY(y:int):void
+getXY():int[2]
+setXY(x:int,y:int):void
+toString():String
+distance(x:int,y:int):double
+distance(another:MyPoint):double
+distance():double
Return a 2-element int array of {x,y}.
"(x,y)"
Distance from this point to the given point at (x,y).
Distance from this point to the given instance of MyPoint.
Distance from this point to (0,0).

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)`.
- An 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 p1 = new MyPoint(3, 4);
System.out.println(p1.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 p1 = new MyPoint(3, 4);
MyPoint p2 = new MyPoint(5, 6);
System.out.println(p1.distance(p2));
```

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

```
MyPoint p1 = new MyPoint(3, 4);
System.out.println(p1.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;
   .......
}
```

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

MyPoint p2 = new MyPoint(0, 4);  // Test another constructor
System.out.println(p2);
// Testing the overloaded methods distance()
System.out.println(p1.distance(p2));    // which version?
System.out.println(p2.distance(p1));    // which version?
System.out.println(p1.distance(5, 6));  // which version?
System.out.println(p1.distance());      // which version?
```

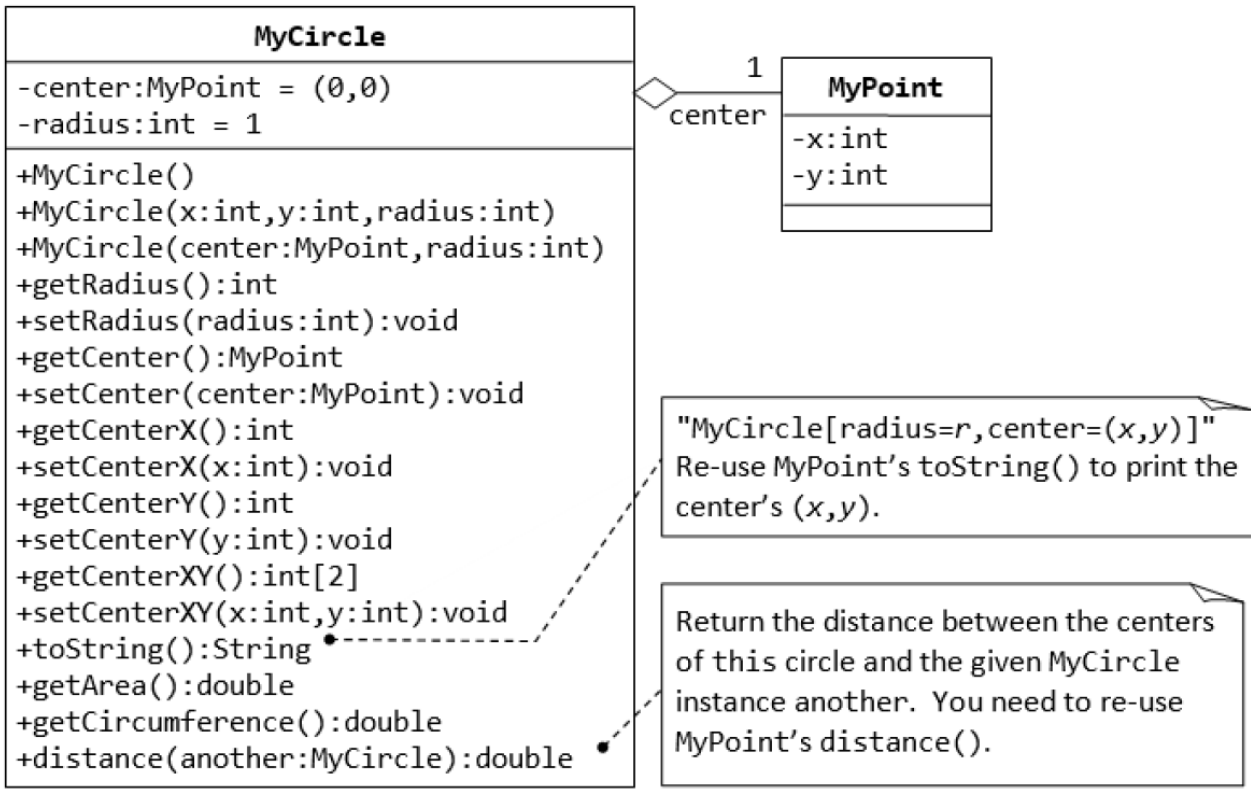
2. Write a program that allocates 10 points in an array of `MyPoint`, and initializes to `(1, 1)`, `(2, 2), ... (10, 10)`.

**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.

```java
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
```

## 2.4 The MyCircle and MyPoint Classes


MyCircle
-center:MyPoint = (0,0)
-radius:int = 1
+MyCircle()
+MyCircle(x:int,y:int,radius:int)
+MyCircle(center:MyPoint,radius:int)
+getRadius():int
+setRadius(radius:int):void
+getCenter():MyPoint
+setCenter(center:MyPoint):void
+getCenterX():int
+setCenterX(x:int):void
+getCenterY():int
+setCenterY(y:int):void
+getCenterXY():int[2]
+setCenterXY(x:int,y:int):void
+toString():String
+getArea():double
+getCircumference():double
+distance(another:MyCircle):double
1
center
MyPoint
-x:int
-y:int
"MyCircle[radius=r,center=(x,y)]" Re-use MyPoint's toString() to print the center's (x,y).
Return the distance between the centers of this circle and the given MyCircle instance another. You need to re-use MyPoint's distance().

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

The class contains:

- Two *private* instance variables: `center` (an instance of `MyPoint`) and `radius(int)`.
- A constructor that constructs a `circle` with the given c`enter'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
}
```

## 2.5 The MyTriangle and MyPoint Classes

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

It contains:

- Three (3) *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.

## 2.6 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.