MACQUARIE
University

WCOM125/COMP125

Fundamentals of Computer Science

Classes and Objects - 1

Gaurav Gupta

July 11, 2018

Contents

List of exercises

Section 1.

Introduction

Java is an object-oriented language. Classes and objects are one of the key programming tools in Java. In this lecture we will cover some of the basic concepts about classes and objects and how they are handled in Java.

Section 2.

Object Oriented Programming – Brief Introduction

- Classes are models of real world entities inside our programs.
- Classes may have instance variables and methods. For example, a **Person** class can have a instance variable **name** of type **String** , and a method **eat()** of type **void** .
- *OO Design* deals with designing the classes to solve a problem.
- *OO Programming* deals with realising that design correctly.

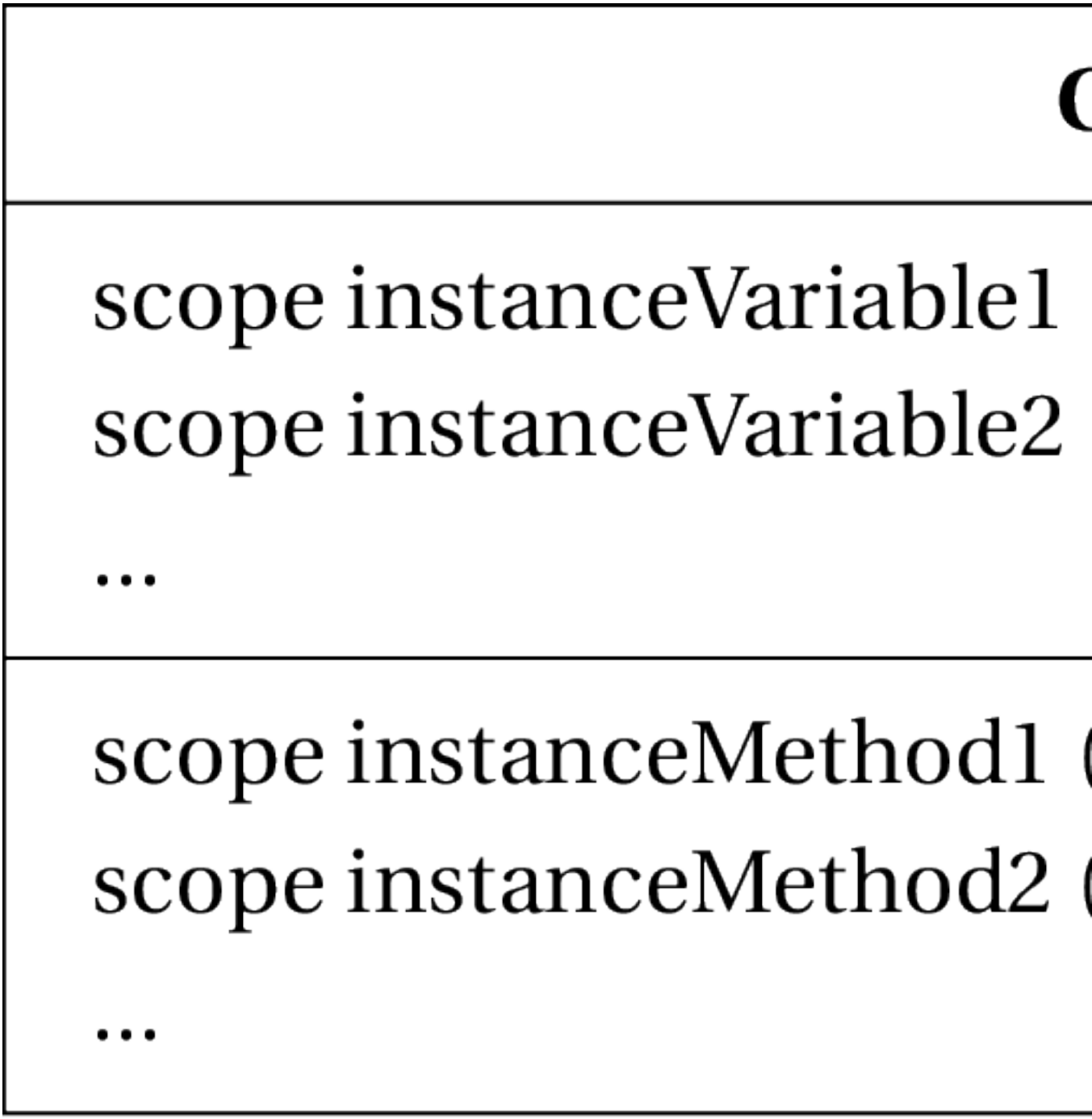
Section 3.

Classes and instances

Class definition specifies -

- instance variables each instance (object) of the class will have

- methods, including instance methods that are called on instances/ objects and class methods or static methods that are called on the class directly without invoking objects.


C
scope instanceVariable1
scope instanceVariable2
...
scope instanceMethod1 (
scope instanceMethod2 (
...

3.1 Example: Java's String class

- As we know `String` is a built in class for handling sequences of characters.
- We can think of it as a Java type, just like **`int`**, **`double`**, **`boolean`**. (But String is a non-primitive type.)
- Inside, it stores a sequence of characters (how, we don't care right now).
- The interface provides a number of operations on the character sequence.

3.2 Example: String object

```
1 String greeting = "Hello, World!";
2 System.out.println(greeting.length());
```

Here, **greeting** is an instance, or, object of **String** class. The statement **String greeting = "Hello, World!";** is called the **instantiation** statement. The class has an instance method **length** that is being invoked on object **greeting**

3.3 Standard way of instantiatiation

```
1 String lecturer1 = "Gaurav";
2 String lecturer2 = new String("Scott");
```

- The two variables are different instances of the `String` class.
- First is known as *lazy instantiation*, just for Strings.
- Second (using **`new`**) is generally how we create instances. More information about this will come a bit later in this lecture ...

Section 4.

Class definition and instantiation

4.1 Defining classes

- Each class is defined in a separate file with the same name and ending with `.java`
- All Java class definitions are separate files in the same folder (for now).

4.2 Adding instance variables

Instance variables can be declared as in the following two examples. Note the **`public`** modifier (for now):

```
1 public int     instanceVar1;
2 public String  instanceVar2;
```

4.3 Defining methods

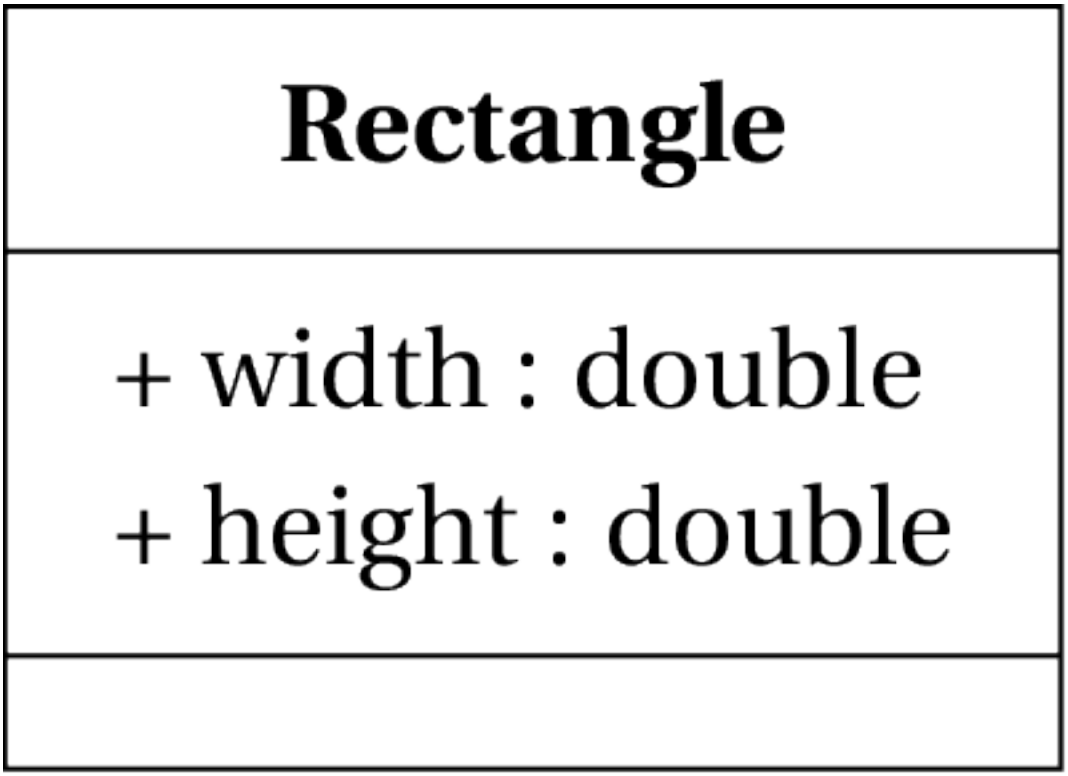
Method definitions have a heading and a method body

- The *heading* defines the name of the method and other useful information.
- The *body* defines the code to be executed when the method is called.

4.4 Example - Defining a class

```
1 public class Rectangle {
2     public double width;
3     public double height;
4 }
```

Defining a class: example Note that the above class definition merely provides a template or blueprint for the class. No complete program using this class has yet been written, and no object (instance) of this class has yet been created.


Rectangle
+ width : double
+ height : double

Exercise 1

Define a class

Define a class for a Circle that is represented by its radius.

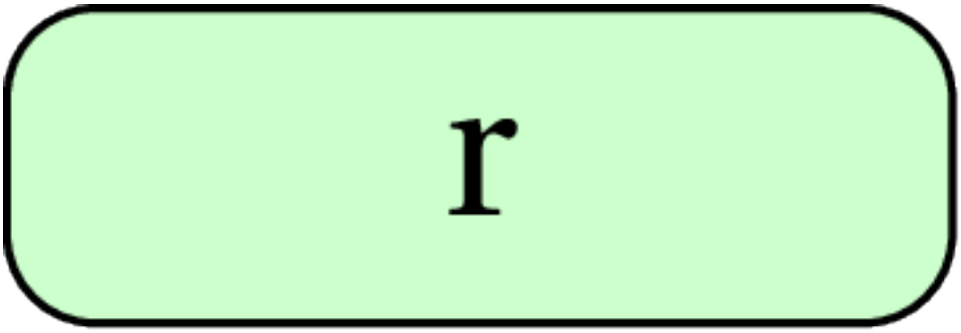
Write your answer here

(SOLUTION 1)

4.5 Declaration and instantiation

- An object or instance of the new class type is declared in main as follows:

```
ClassName  classVar; //declaration
Rectangle r; //example
```


r

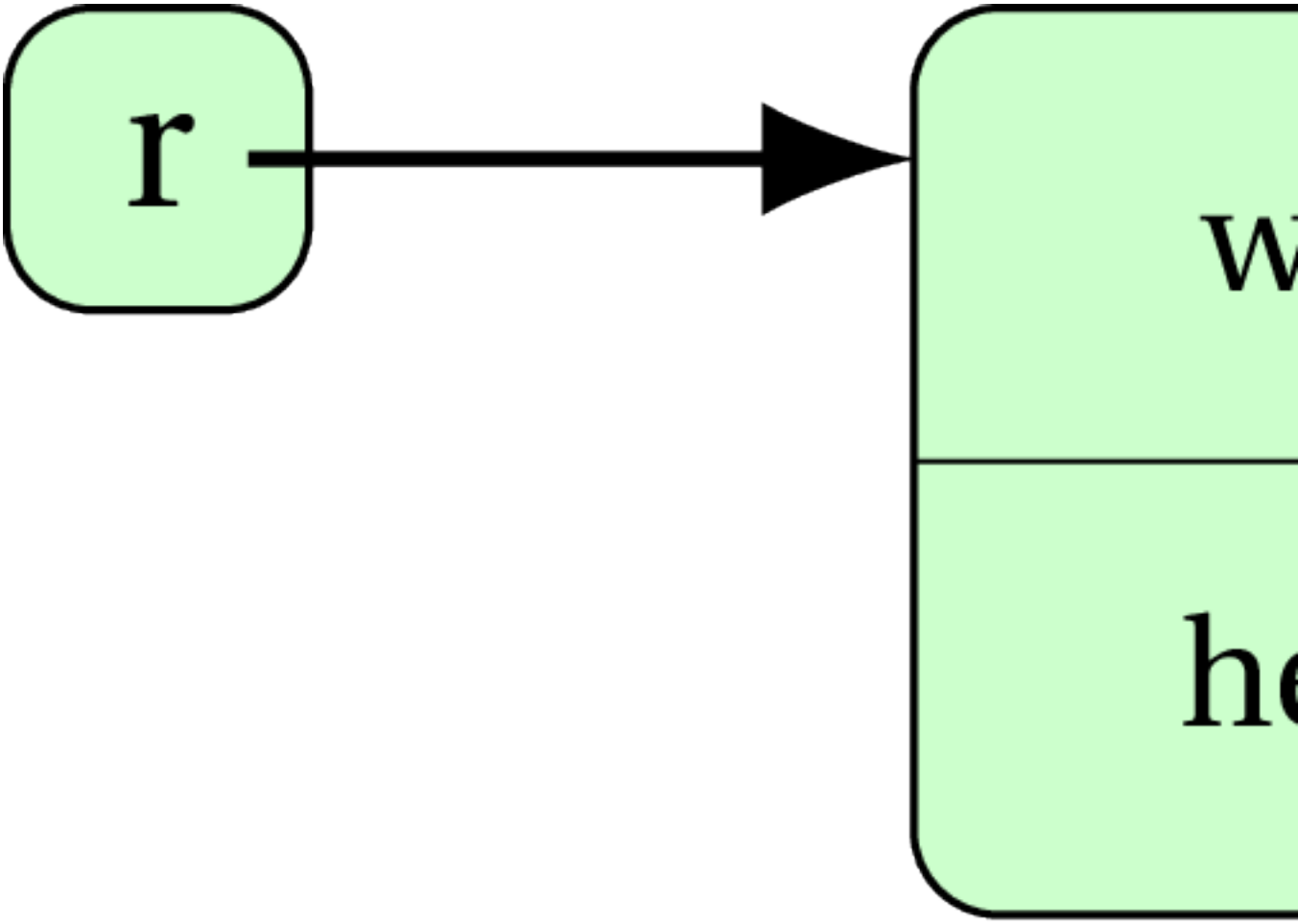
4.6 Declaration

Declaration creates a *reference* in the memory, which doesn't refer to any storage space yet.

4.7 Declaration and instantiation

- We perform the `instantiation` statement to allocate storage space for the instance variables of the object declared and to refer to that memory.

```
classVar = new ClassName(); //instantiation
r = new Rectangle(); //example
```


r
w
he

- These can be combined:

```
ClassName classVar = new ClassName();
//declaration + instantiation

Rectangle r = new Rectangle(); //example
```

Exercise 2

Declare and Instantiate an object

Declare and instantiate an object `myCircle` of class `Circle`

Write your answer here

(SOLUTION 2)

4.8 Adding method to a class

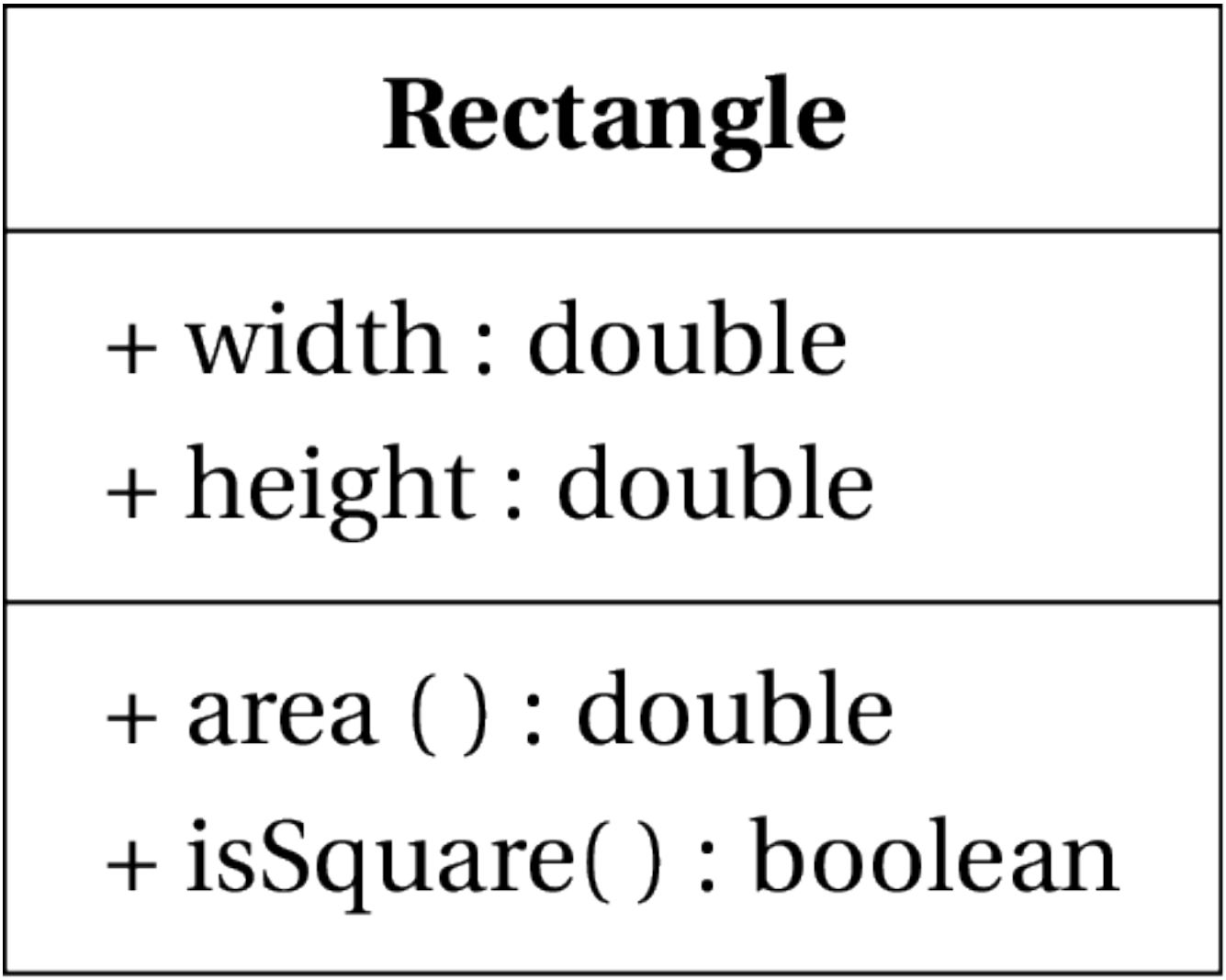
You can add methods inside the class that can be called on any *instance* of the class.

```
public class Rectangle {
    public double width;
    public double height;

    public double area() {
                return width * height;
    }

```

```
9       public boolean isSquare() {
10          if(width == height)
11                  return true;
12          else
13                  return false;
14          //equally correct:
15          //return width==height;
16      }
17  }
```


Rectangle
+ width : double
+ height : double
+ area ( ) : double
+ isSquare( ) : boolean

4.9 The dot (.) operator

The dot operator gives us access to the members (instance variables and method) for an object. Think of it as the **apostrophe s ('s)** of the human language (as in *"Gaurav's class"* or *"Matt's workshop"*)

```
1  Rectangle r = new Rectangle(); //example
2  r.width = 5;
```

- The expression **r** gives us access to the instance variable **width** of object **r**.
- When a method is invoked on an object using the dot operator, it calls the method as defined in the class **in the context** of that object and any instance variables used in that method are the ones belonging to that object.

```
1  Rectangle r = new Rectangle();
2  r.width = 5;
3  r.height = 8;
4  System.out.println(r.area());
```

Here, **r.area()** returns **width * height** and since the method is called on object **r**, it returns **r.width * r.height**. Had the method been called on another object **s**, it would return **s.width * s.height**.

Exercise 3

Access instance variables and call methods

Write a piece of code that sits outside the class definition and displays the radius of the object `myCircle` and also its area.

```
Write your answer here
(SOLUTION 3)
```

4.10 Are there any default values?

- Each *instance variables* is automatically initialised to the default value for its type when an object of the class is created.
- For example, an instance variable of type `int` is given the default value 0;
- And an instance variable of type `String` (or any class type) is given the default value `null`. (More about `null` later.)

Section 5.

Getters and setters

5.1 Bad client, bad bad client!

Once the object is created, we can start operating on it.

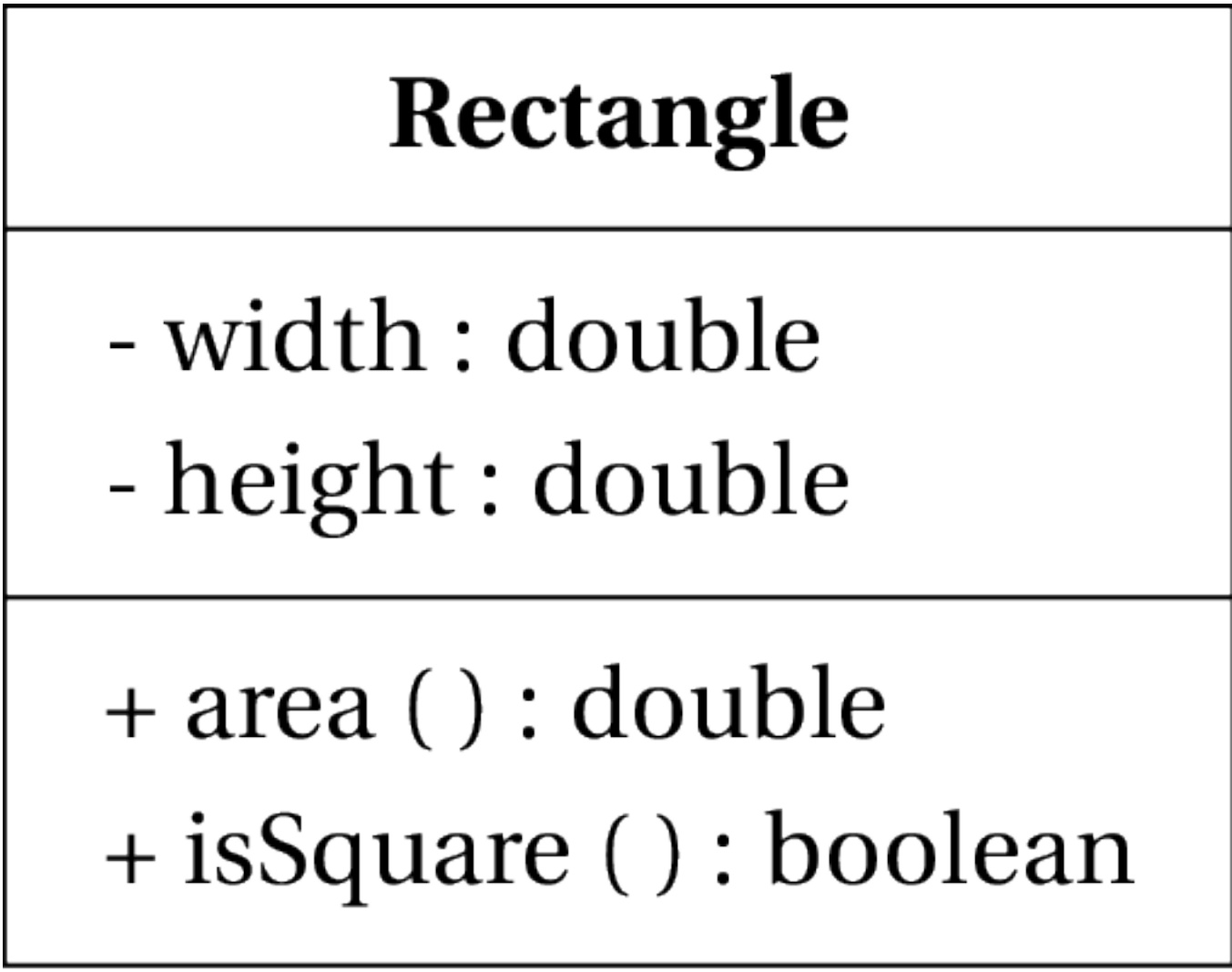
```
Rectangle r = new Rectangle();
r.width = -5; // :o :O
r.height = 8;
System.out.println(r.area());
//displays -40 :'(
```

5.2 Changing visibility to private

```
public class Rectangle {
    private double width;
    private double height;

    public double area() {
        return width * height;
    }

    public boolean isSquare() {
                return (width==height);
    }
}
```


Rectangle
- width : double
- height : double
+ area ( ) : double
+ isSquare ( ) : boolean

Now, the instance variables **width** and **height** are visible only within the class definition.

5.3 How does one access (read/write) private instance variables

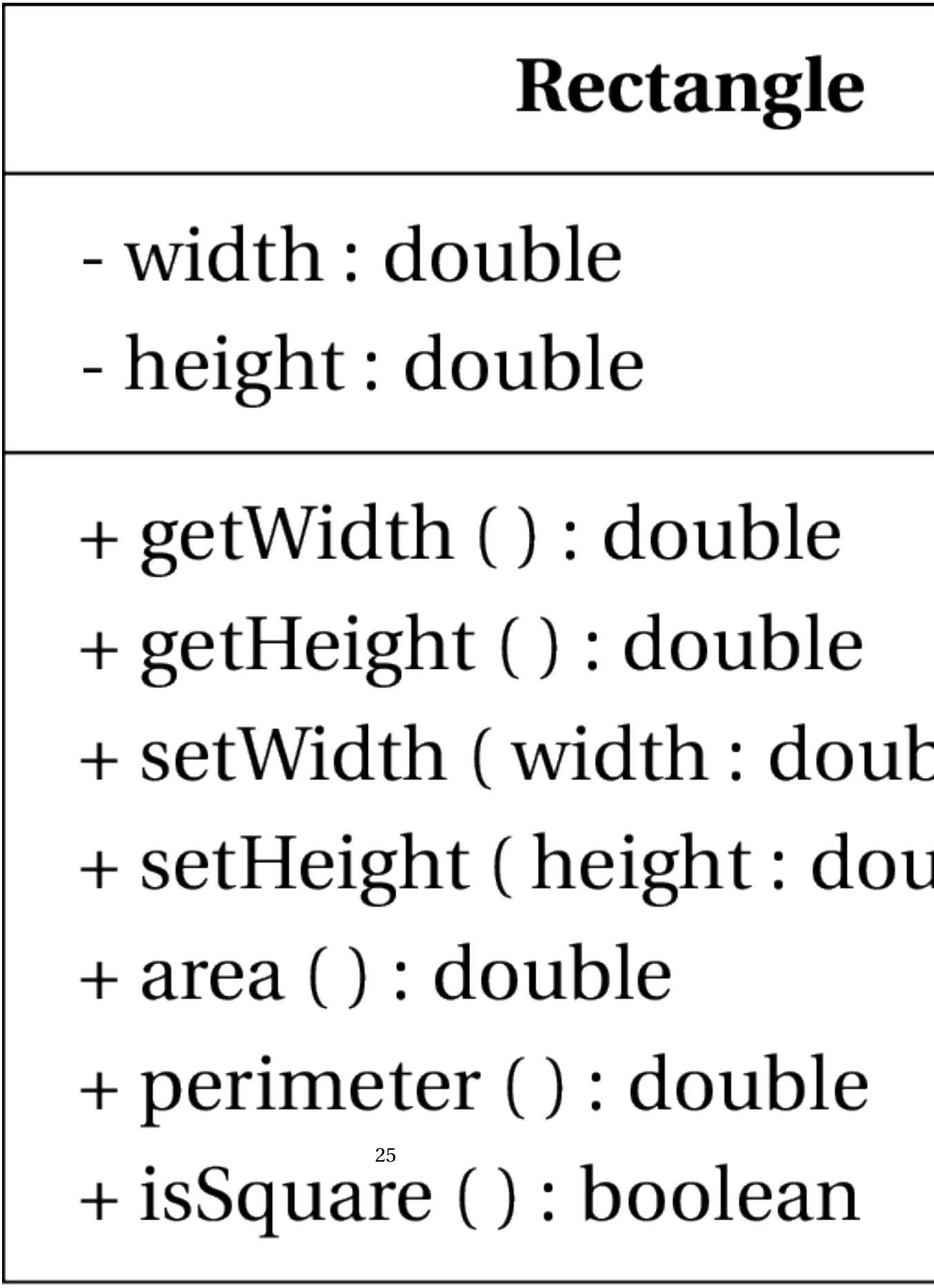
We access (read and write) private instance variables through public methods called **getters** and **setters**.

- **getters** return the value of the instance variable to the caller.
- **setters** set the value supplied by the caller to the instance variables.

```
1 public class Rectangle {
2     private double width, height;
3     public double getWidth() { return width; }
4     public void setWidth(double w) {
5       width = Math.abs(w);
6     }
7     //...same for height
8     //rest of the class definition...
9 }
```

5.4 Setters must provide validation where applicable

You can see that we validated the passed values before assigning to the instance variable as **width = Math.abs(w)**. This is a typical case and setters are in charge of validating data before assigning it to the instance variables.


Rectangle
- width : double
- height : double
+ getWidth ( ) : double
+ getHeight ( ) : double
+ setWidth (width : doub
+ setHeight (height : dou
+ area ( ) : double
+ perimeter ( ) : double
+ isSquare ( ) : boolean

Exercise 4

Add getters and setters

Add getters and setters to class `Circle`. The setter should result in radius becoming zero if the parameter passed is not positive.

Write your answer here

(SOLUTION 4)

Exercise 5

Write a client

Write a client (code sitting outside `Circle.java`, for example, in the `main` method of another class) that performs the following operations,

- Declare and instantiate object `myCircle` of class `Circle` that has a radius of 1.8
- Display radius of `myCircle`.
- Increase radius of `myCircle` by 1.4

Write your answer here

(SOLUTION 5)

5.5 I wish creating objects was easier

Let's say that the user wants to create a **Rectangle** whose **width** is 5 and **height** is 8. Following code achieves this,

```
1 Rectangle r = new Rectangle();
2 r.width = 5;
3 r.height = 8;
```

However, it would be really nice if one could pass the values for the instance variables in the instantiation statement itself, as,

```
1 Rectangle r = new Rectangle(5, 8);
```

This is done through **constructors** .

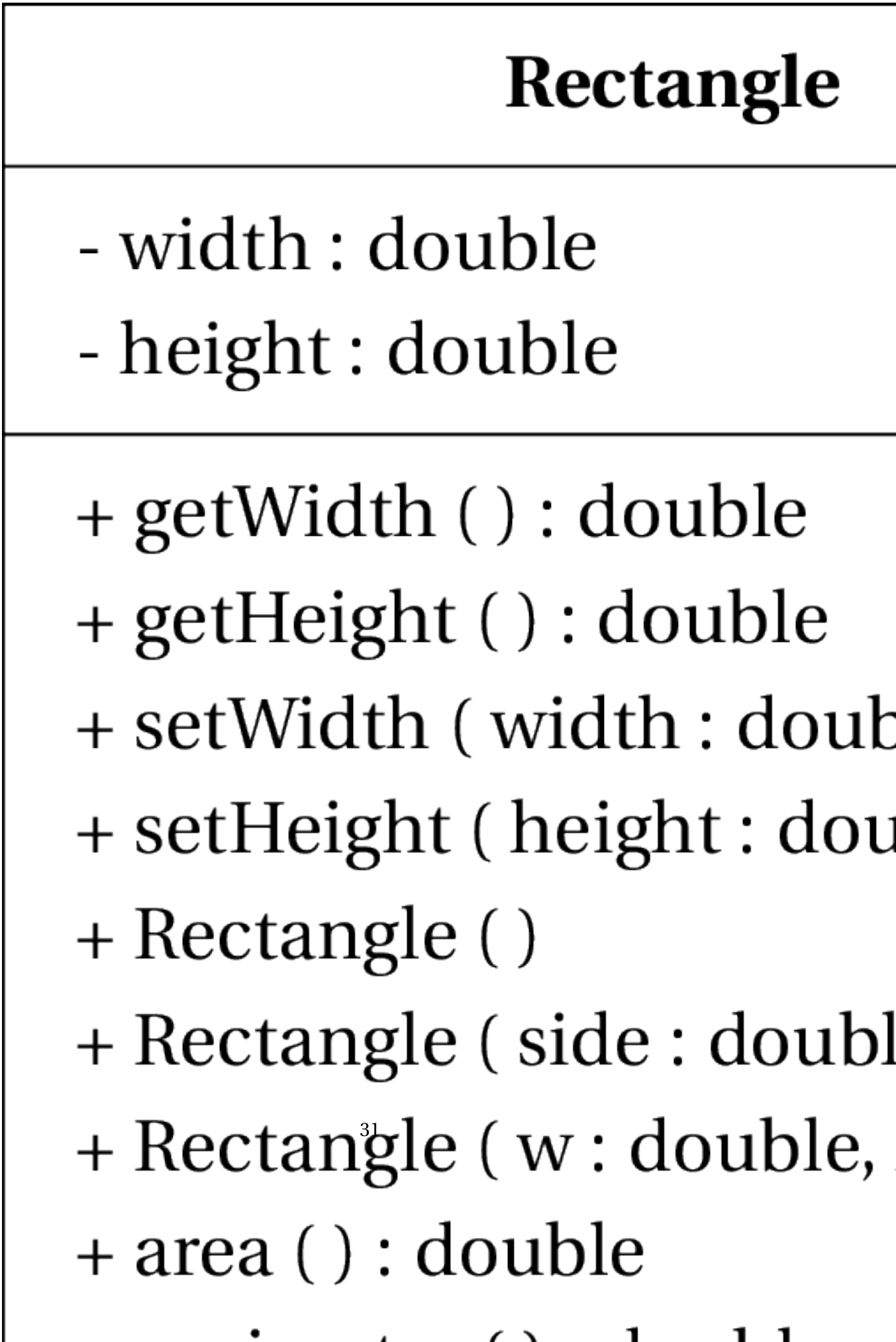
Section 6.

Constructors

- A constructor is a method defined in the class.
- A constructor must have the same name as the class.
- A constructor has no return type (not even void).
- There may be multiple constructors, each distinguished by its parameter list. Thus, we may have one constructor with no parameters, and another with one **int** parameter.
- A suitable constructor is automatically called during instantiation based on number of parameters passed. If an appropriate constructor is not found, a compilation error is generated.

6.1 Example - Constructor

```
1  public class Rectangle {
2      private double width, height;
3
4      //getters and setters
5
6      public Rectangle() { //default constructor
7                  setWidth(1);
8                  setHeight(1);
9      }
10
11     //parameterized constructor for a square
12     public Rectangle(double side) {
13                 setWidth(side);
14                 setHeight(side);
15     }
16
17     //parameterized constructor - generic
18     public Rectangle(double w, double h) {
19                 setWidth(w);
20                 setHeight(h);
21     }
22     //rest of the code
23 }
```


Rectangle
- width : double
- height : double
+ getWidth ( ) : double
+ getHeight ( ) : double
+ setWidth (width : doub
+ setHeight (height : dou
+ Rectangle ( )
+ Rectangle (side : doubl
+ Rectangle (w : double,
+ area ( ) : double

6.2 Constructors should call setters - always!

Constructors should always use setters to assign values to instance variables.

6.3 Default constructor

It should be noted that a default constructor (without any parameters) is pre-defined for you by Java and that's why you can instantiate objects without defining it yourself.

```
1 Rectange r = new Rectangle();
```

The default constructor assigns the default values for the appropriate data types to the instance variables. However, once you define a parameterized constructor, the built-in default constructor is taken away by Java. Thus, if you want to construct an object with default initial values for the instance variables, you need to re-define that!

6.4 Defining the default constructor

Let's say the default **Rectangle** instance should be of unit length. We can define the default constructor as,

```
1 public Rectangle() {
2         setLength(1);
3         setBreadth(1);
4 }
```

Exercise 6

Add default and parameterized constructor

Add two constructors to the class `Circle` .

1. No parameters passed (default constructor): Assigns the value 1.0 to radius ... through the setter.
2. Parameter passed for radius (parameterized constructor): Assigns the passed value to radius through the setter.

Write your answer here

(SOLUTION 6)

Section 7.

Displaying objects

Often we need to display the details of an object. For example, we might need to display name and age of a Person object, or the details of a Time object in the format **hours:minutes:seconds** , or in our example, **width** and **height** of a **Rectangle** object. It is quite inconvenient to display these details as,

```
Rectangle r = new Rectangle(5, 8);
System.out.println(r.width+`` by ''+r.height);
```

We *could* add a method **display** in the class **Rectangle** as,

```
public void display() {
    System.out.println(width+`` by ''+height);
}
```

And call this method on required object as,

```
Rectangle r = new Rectangle(5, 8);
r.display();
```

7.1 Problem with the display() method

But this would only let us **display** the object details, and not send to a file, or concatenate with any other output.

Java provides a standard way to return the String description of an object using the **toString()** method (with return type **String**).

7.2 Default toString() behaviour

When you display an object, what Java displays is the outcome of the toString() method on that object

```
Rectangle r = new Rectangle(1, 3);
System.out.println(r); //something like [I@70dea4e
```

Java saw that you want to display a **Rectangle** object and replaced it by the **toString()** method operating on that object as,

```
System.out.println(r);
//became
System.out.println(r.toString());
```

7.3 Over-riding toString() behaviour

We can over-ride **toString()** method as required. For the **Rectangle** class,

```
public String toString() {
    return width+`` by ''+height;
}
```

When we display an object, it invokes the method **toString()** and displays the value it returns.

```
Rectangle r = new Rectangle(5, 8);
System.out.println(r);
/*
automatically invokes r.toString()
and displays the value returned
*/
```

Exercise 7

Define toString method

Define the `toString` method in the `Circle` class such that it displays the object details in the following format -

`Circle radius: <radius>, area: <area>`

In a separate client, create a Circle object with radius 1.6 and display it on the console.

Write your answer here

(SOLUTION 7)

Section 8.

Class containing an array

In this unit, we'll extensively see classes containing arrays.

We must remember that both arrays and objects are references and refer to the memory holding the actual data (in the case of array, it's the array items, and in the case of objects, it's the instance variables).

Take the following as an example (instance variables are **public** for simplicity),

```
1 public class DynamicArray {
2         public int[] data;
3         public int nItems;
4 }
```

The client is as follows,

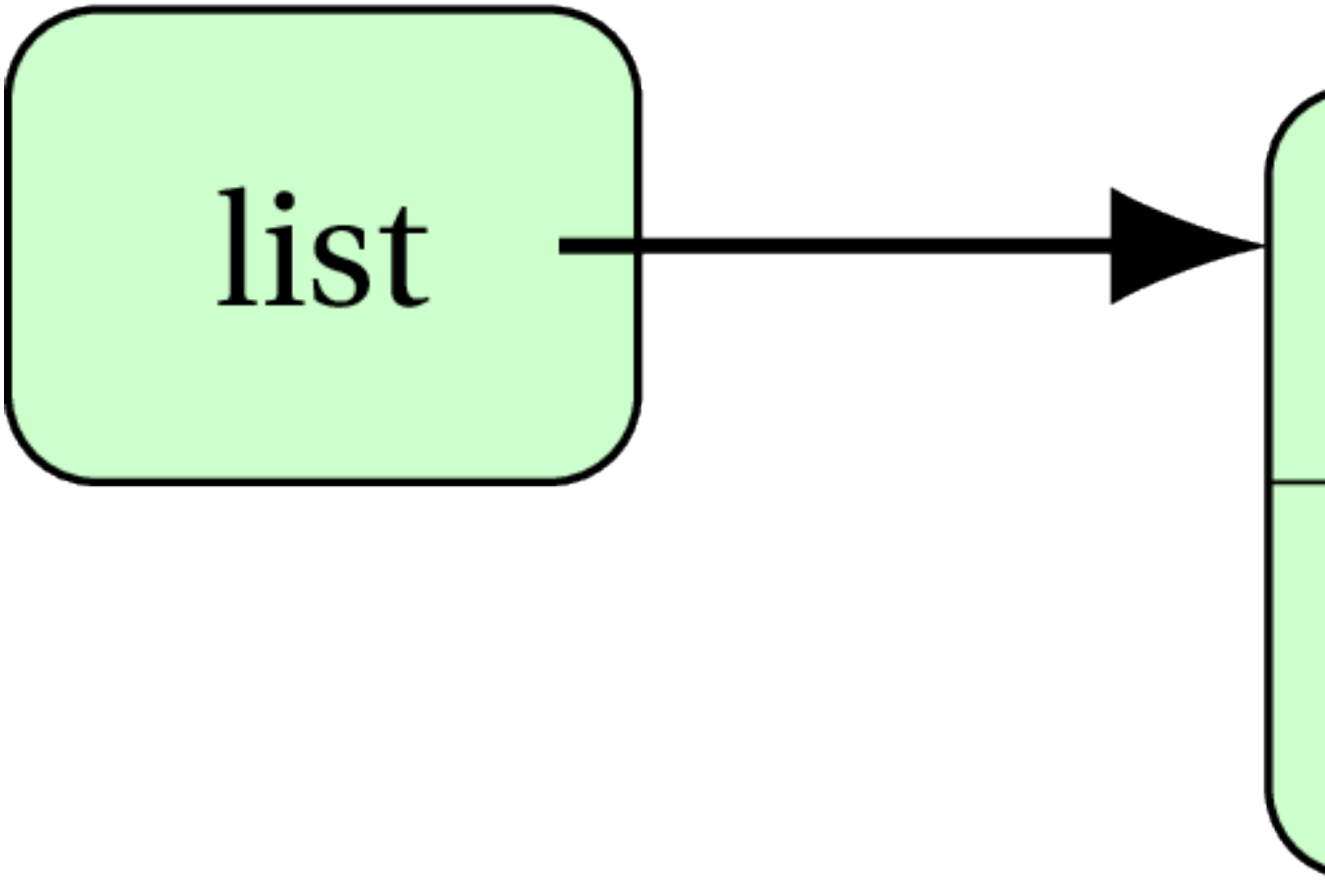
```
1 public class Client {
2         public static void main(String[] args) {
3                 DynamicArray list = new DynamicArray();
4         }
5 }
```

At this stage, the memory state looks like,

If we change the values of the instance variables as,

```
1 public class Client {
2         public static void main(String[] args) {
3                 DynamicArray list = new DynamicArray();
4                 list.data = new int[4];
5                 list.nItems = 0;
6         }
7 }
```

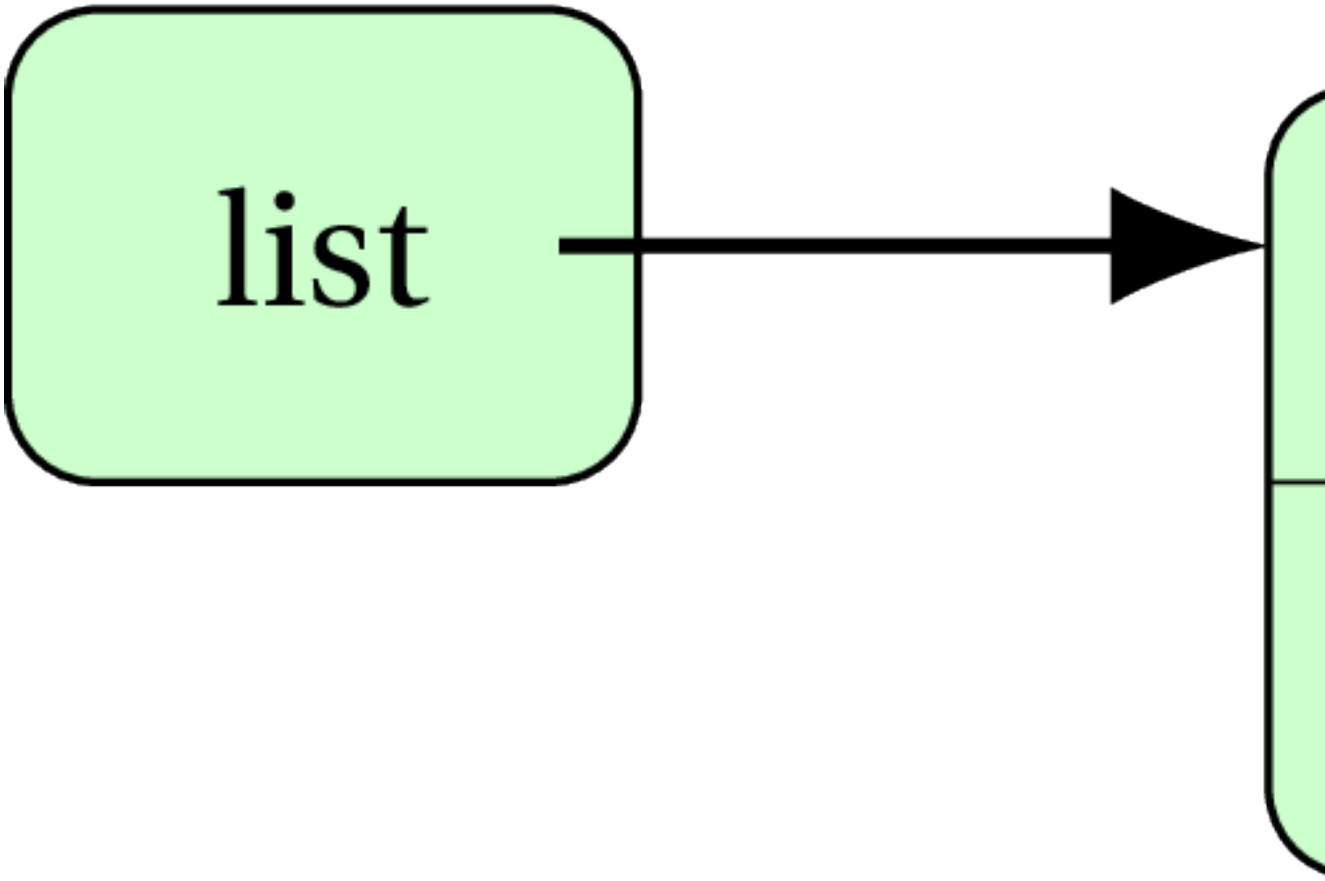
It now becomes,


list

Finally, we can modify the items of the array as,

```
public class Client {
        public static void main(String[] args) {
                DynamicArray list = new DynamicArray();
                list.data = new int[4];
                list.data[0] = 5;
                list.data[1] = 12;
                list.nItems = 2;
        }
}
```

This gives us,


list

Section 9.

Sample solutions for exercises

Solution: Exercise 1

```
public class Circle {
        public double radius;
        /*
        * note that int is a wrong choice as radius
        * CAN be a floating-point value like 1.5 or 2.4
        */
}
```

Solution: Exercise 2

```
public class Client {
        public static void main(String[] args) {
                Circle myCircle = new Circle();
        }
}
```

Although, you can just write the relevant part in written exams:

```
Circle myCircle = new Circle();
```

Solution: Exercise 3

```
System.out.println(myCircle.getRadius());
System.out.println(myCircle.area());
```

Solution: Exercise 4

```
//setter
public void setRadius(double r) {
```

```
        if(r < 0)
                radius = 0;
        else
                radius = r;
}

//getter
public double getRadius() {
        return radius;
}
```

Solution: Exercise 5

```
public class Client {
        public static void main(String[] args) {
                Circle myCircle = new Circle();
                myCircle.setRadius(1.8);
                System.out.println(myCircle.getRadius());
                myCircle.setRadius(myCircle.getRadius() + 1.4);
                /* or you can split it up as:
                * double current = myCircle.getRadius();
                * double updated = current + 1.4;
                * myCircle.setRadius(updated);
                */
        }
}
```

Solution: Exercise 6

```
//default constructor
public Circle() {
        setRadius(1);
}

//parameterized constructor
public Circle(double r) {
        setRadius(r); //let setter handle the validation
```

```
}
```

Solution: Exercise 7

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