Objects - Classes

Part 2

Declaring Classes

public class Rectangle {

1. Instance Variables
2. Constructor(s)
3. Getters/Setters
4. Other methods

}

Access Modifiers:

Public: Other classes can access the field

Private: Only its own class can access the field

Constructor

In order to create an object we must use the **new** operator to instantiate a new class object. **Instantiation** is the process of creating an instance of an object which allocates memory for a new object and references that object in memory.

Let's assume class rectangle has 2 instance variables width and height. We want to create an instance of that class (an object) and initialize the width and height values:

```
Rectangle rect = new Rectangle(4, 6)
```

Constructor

The name of the constructor must be the same as the name of the class.

```
public class Rectangle{
   private int width;
   private int height;
   public Rectangle(int myWidth, int myHeight){
        width = myWidth;
        height = myHeight;
    }
}
```

Using the Constructor

```
private int width;
private int height;
```

Instance variables

```
public Rectangle(int myWidth, int myHeight){

    width = myWidth;

    height = myHeight;

}
```

The arguments are passed using call by value into the parameters.

```
Rectangle rect = new  Rectangle(4, 7);
```

Arguments parameters must match the types identified in the parameters.

Creating Multiple Constructors

Let’s say we want to create squares. We can actually write:

Rectangle rect = new Rectangle(16);

We can add additional constructors that take different number of parameters.

```
public Rectangle(int size){
    width = size;
    height = size;
}
```

Constructor: No-argument (default constructor)

Constructors do not need to have parameters in order to create an object

No-argument constructor set objects to a default value.

```
public Rectangle(){

}
```

```
Rectangle rect = new  Rectangle();
```

Overloading

Having multiple constructors with the same name but different parameters is called **overloading**.

The compiler knows which constructor to use.

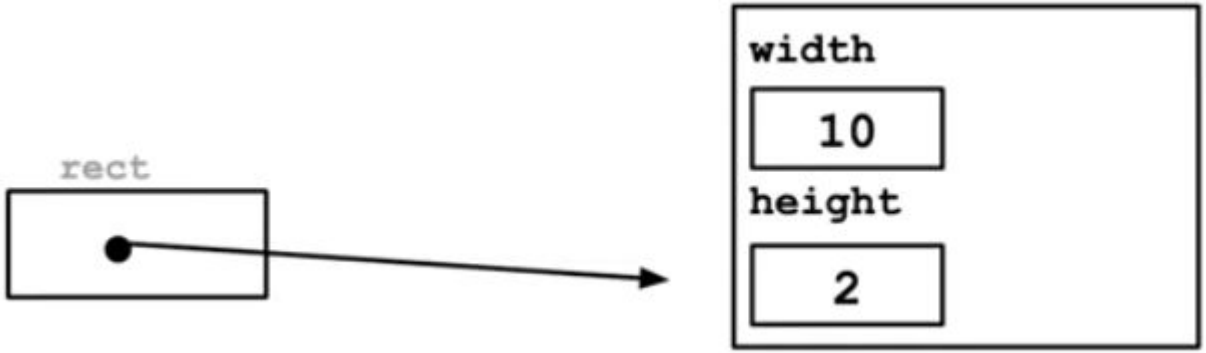
```
public class Rectangle{
   private int width;
   private int height;

   public Rectangle(){

   }
   public Rectangle(int size){
      width = size;
      height = size;
   }
   public Rectangle(int myWidth, int myHeight){
     width = myWidth;
     height = myHeight;
   }
}
```

Objects in Memory

In memory the variables simply stores a location or a reference



Objects in Memory

When we write:

Rectangle rect;

The variable is not pointing at any object data. When an object reference is not pointing to any object data, it is considered to be **null.** Null objects references do not allocate any memory.

Static and Non-static methods

A **static method** is a method that belongs to a class, but it does not belong to an instance of that class and this method can be called without the instance or object of that class. They may not use non-static methods. Example: static methods in class “Math.abs()”, “Math.pow()”, “Math.PI”

```
public static void printMsg(){
  System.out.println("Hi! Rectangle class");
}
```

Non-static belongs to each object that is generated from the class. Methods can access any **static** method and **static** variable.

```
public int calcArea(){
  return width * height;
}
```

Exercise

Create a class Employee

Define instance variables

Define at least 3 construtor (1 of them default constructor)

Declare static and non-static methods

Create multiple objects from your Driver and test the constructors and methods.