

CS620c

The scanner class as an example of OO
usage, first principles of OO (Object
Oriented) programming

Joe Duffin

└

Callan Building Room 2.108

Email: Joseph.Duffin@nuim.ie

First Principles of Object Oriented programming

When we write software we store representations of real world concepts or objects in program memory and we then perform some operations on this data representation to transform their values.

In the languages before OO languages, the data storage and method to operated on these were often distributed in many places throughout the software.

This meant that it was difficult to keep track of changes when they were necessary or perform upgrades to software as a result faults or updates to add improved functionality

Part of the OO design philosophy is to **encapsulate** or bundle both the data representations and the means to transform this data in the one location. Another part is to hide this data from outside the class only allowing the class methods to access the data. This is called **information hiding**.

In other words, we create data types that have memory storage and also methods that are capable of operating on this data.

Keyboard Input (Essential steps)

- To get keyboard input you need three things...
 1. The Scanner class is defined in the java.util package (toolbox). So you must have the following statement at the **top of your program** to bring this **Scanner (tool)** into your program.

```
import java.util.*; OR  
import java.util.Scanner;
```

2. Need to create an instance of the Scanner class (e.g in the main method)

```
Scanner sc = new Scanner(System.in);
```

3. Need to call one or more Scanner methods to read in data input

```
int number = sc.nextInt( );
```

Object Oriented aspects of this.

```
import java.util.Scanner;
```

Scanner is a Class (or type) or a template for creating an object in your program. Think of a template or stencil for making cookies. **Class is to object as template is to cookie.**

The line below uses the **Class constructor** to **create** an object **sc** of type Scanner. What we are doing here is creating an object which will be attached to the standard input (keyboard) and become a software representation for the keyboard in your program (**sc** below)

```
Scanner sc = new Scanner(System.in) ;
```

Now we can send messages to this newly created object **sc** or use method calls on it to retrieve information from the keyboard. The list of all possible method calls on an object of a certain class is called the class API.

```
int number = sc.nextInt() ;
```

Important Methods

- **nextInt()** // reads in the next token as an int
- **nextFloat()** // reads in the next token as an float
- **nextDouble()** // reads in the next token as an double
- **nextLine()** // reads in the next token as an String
- **hasNextInt()** //returns true if there is another int to read in
- **hasNextFloat()** //returns true if there is another float to read in
- **hasNextDouble()** //returns true if there is another double to read in
- **hasNextLine()** //returns true if there is another String to read in
- Look at the API for a full list of methods ...

Take a look the Scanner class in the Java API for examples.

<http://docs.oracle.com/javase/7/docs/api/>

Getting input from the keyboard example

```
import java.util.Scanner; //import the Scanner class from the java.util library before using it in your code

public class GettingInput
{
    public static void main(String []args){

        // myScanner is a software representation of the keyboard. Methods are called on this
        // to the retrieve information from the keyboard.
        Scanner myScanner = new Scanner(System.in);

        System.out.println("Please enter your first name and press enter");
        String myName = myScanner.nextLine();

        System.out.println("Hi " + myName);

        System.out.println("Please enter an integer and press enter : ");
        int myNumber = myScanner.nextInt();
        System.out.println("you entered the number " + myNumber);

    }
}
```

First Principles and concepts used Object Oriented programming.

Class Scope	This is when a variable (attribute) is declared just inside the first line of a class after the first brace {. The scope of this variable covers the whole class which includes any methods inside the class. (NB: revise scope rules).
Data encapsulation	This OO concept is achieved by bundling the methods (behaviour) and the variable (attributes) of a class representation together in the class.

OO principles and concepts

private	<p>This is an access modifier (a java keyword) which is put in front of your class variables declarations (variables declared just inside your class).</p> <p>Only methods defined inside a class definition can have access to or are able to change the value of variable (attributes) declared with the access modifier private. (information hiding)</p>
----------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Information hiding	<p>Information hiding is the principle OO concept of only allowing the methods defined within a class to have access to or change the values of the attributes of the class.</p> <p>Achieved by declaring the class attribute variables to using the access modifier private in front of their declarations.</p>
---------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

OO principles and concepts

public

This is an **access modifier** (a java keyword) which is put in front of your methods defined within your class. This allows other code to call these methods using an instance of an object and the dot “.” operator followed by the method name. (Calling methods in this way allows us to indirectly change or retrieve the values of private class variables).

You allow the public to have a way of accessing your attributes; you provide a **public interface** (set of publicly available methods) to access your attributes (class variables).

Note: the variables can NOT be accessed without going through the methods (E.g. in the Circle Class you use the methods setX, setY, setRad, setColour)

Generic Class Definition : We define our own type.

```
public class ClassType {  
  
    private int attribute1; // each created object has its own one  
  
    public ClassType( int num ) { // notice NO return type  
  
        attribute1 = num; // set attribute value on construction  
    }  
  
    public void setAttribute1( int num) { // sets the attribute  
  
        attribute1 = num;  
    }  
  
    public int getAttribute1( ) { // gets the attribute  
  
        return attribute1;  
    }  
  
} // end of ClassType definition
```

A **basic Class Definition** should have the following:

- ❑ Private class **variables** (or attributes) A copy of these is created **every time** you create an object using the Class.
- ❑ Methods or behaviours to access and change the class variables (attributes). Known as **getter** and **setter** methods.
- ❑ A special method called a class constructor method which is called when an object is being created. **If you do not define a class constructor method java automatically provides you with a default constructor method which takes no parameters.** *But it won't do this if you define at least one constructor

Using the Class definition to declare a class reference variable and create an object (or instance of the class). (Class is to object as Template is to Cookie)

```
ClassType Identifier = new ClassType ( parameter );
```

ClassType is the Class that we define (the one we write).

Identifier is the link reference variable name which will link to or newly created object.

new the java key word.

ClassType (**parameter**) is the class constructor method for this class

```
BasicCircle myCircle = new BasicCircle ( 5 );
```

This creates a BasicCircle object with an initial radius set at 5

Exercise 1

Using the information on the previous page, label the different parts of the line of code below and state overall what the line is doing.

```
Scanner sc = new Scanner(System.in) ;
```

Calling methods on a newly created object

```
Identifier.setterMethodName (parameter) ;
```

```
int x = Identifier.getterMethodName ( ) ;
```

You use the Identifier (the variable that stores the link to the object) followed by the “.” operator, followed by the setter or getter method name to access the private class variable.

```
myCircle.setRadius (10) ; //setter
```

```
int number = myCircle.getRadius () ; //getter
```

Exercise 2

Using the information on the previous page, label the different parts of the line of code below and state overall what the line is doing.

```
int v = sc.nextInt() ;
```

Creating an **instance** of the **ClassType**: Using our Class Type definition.

```
public class TestingClassType {    // just a basic java program

    public static void main (String [ ] args){

        ClassType    myObjRef = new ClassType(20);

        myObjRef.setAttribute1(15);

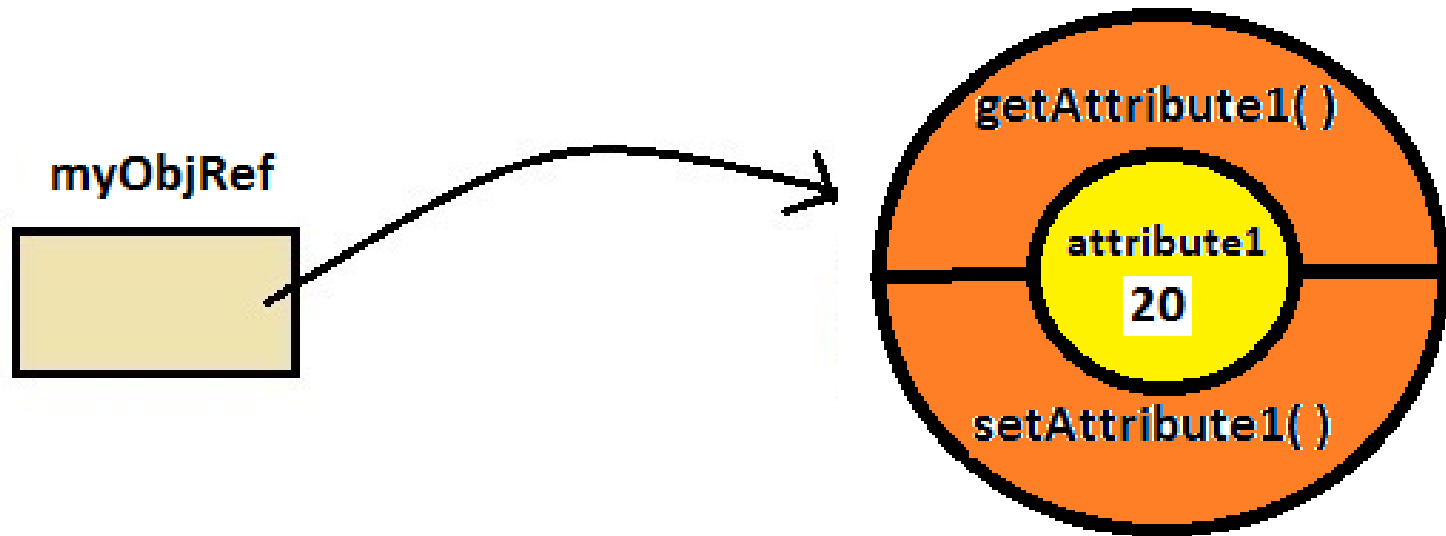
        int value = myObjRef.getAttribute1( );
        // get the object's attribute value

        System.out.println("Value of attribute1 :" + value);

        } // end of main method

} // end of TestingClassType
```


The Creation of an object in memory and having a connection to that object stored in **myObjRef**



```
ClassType    myObjRef = new ClassType(20);
```

Calling methods on a newly created object

```
myObjRef.setAttribute1(parameter) ;
```

```
int x = myObjRef.getAttribute1( ) ;
```

You use the **myObjRef** (the variable that stores the link to the object) followed by the “.” operator, followed by the setter or getter **method name** to access the private class variable.

Examples below:

```
myCircle.setRadius(10) ; //setter
```

```
int number = myCircle.getRadius() ; //getter
```

BasicCircle Class

```
public class BasicCircle
{
    private int radius = 1; // private Class variable, only methods inside the class can access this.
    /**
     * Circle constructor :
     * @param r the radius of the circle
     */
    public BasicCircle(int r) //Notice there is NO return type for a class constructor.
    {
        radius = r; // set the class attribute (variable) radius equal to r
    }
    /**
     * This provides the radius of the circle
     * @return the radius of the circle.
     */
    public int getRadius(){ // getter method
        return radius; // return the value of the class attribute radius
    }
    /**
     * This sets the radius of the circle
     * @param num the new value for the radius of the circle.
     */
    public void setRadius(int num){ // Setter method
        radius = num; // set the class attribute (variable) radius equal to num
    }
}
```

BasicCircleTester: Using the BasicCircle Class or (creating an instance of type BasicCircle) or (creating and object of type BasicCircle)

```
public class BasicCircleTester
{
    public static void main(String [] args){
        // 1) Declare a variable to be a link reference to a BasicCircle
        // 2) Create a BasicCircle object using the BasicCircle(5) class constructor
        //     and pass the reference to this newly created object to the myCircle variable
        BasicCircle myCircle; //Step (1)

        myCircle = new BasicCircle(5); // Step (2)

        //int x = myCircle.radius; // is an error as Radius is a private variable

        int number = myCircle.getRadius();// use getRadius to access radius

        System.out.println("The value of the Circle radius is " + number);
        // myCircle.radius = 10; // is an error as Radius is a private variable
        myCircle.setRadius(10); // use setRadius to change the radius value

        number = myCircle.getRadius(); // use getRadius again

        System.out.println("\n\n The value of the Circle radius is now " + number);
    }
}
```

Declaring a reference variable of type **BasicCircle** then creating a **BasicCircle object** and linking it to the reference variable

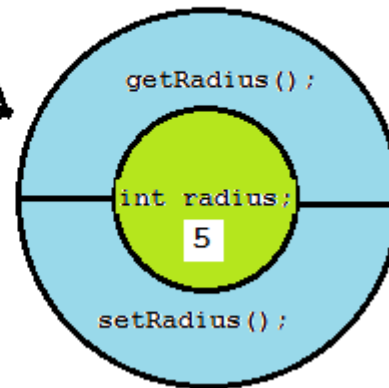
```
BasicCircle myCircle; //Step(1)declare reference variable  
//Step(2)Create object and step(2.1)allow myCircle to link  
//to the object  
myCircle = new BasicCircle(5);
```

Step (1)

myCircle



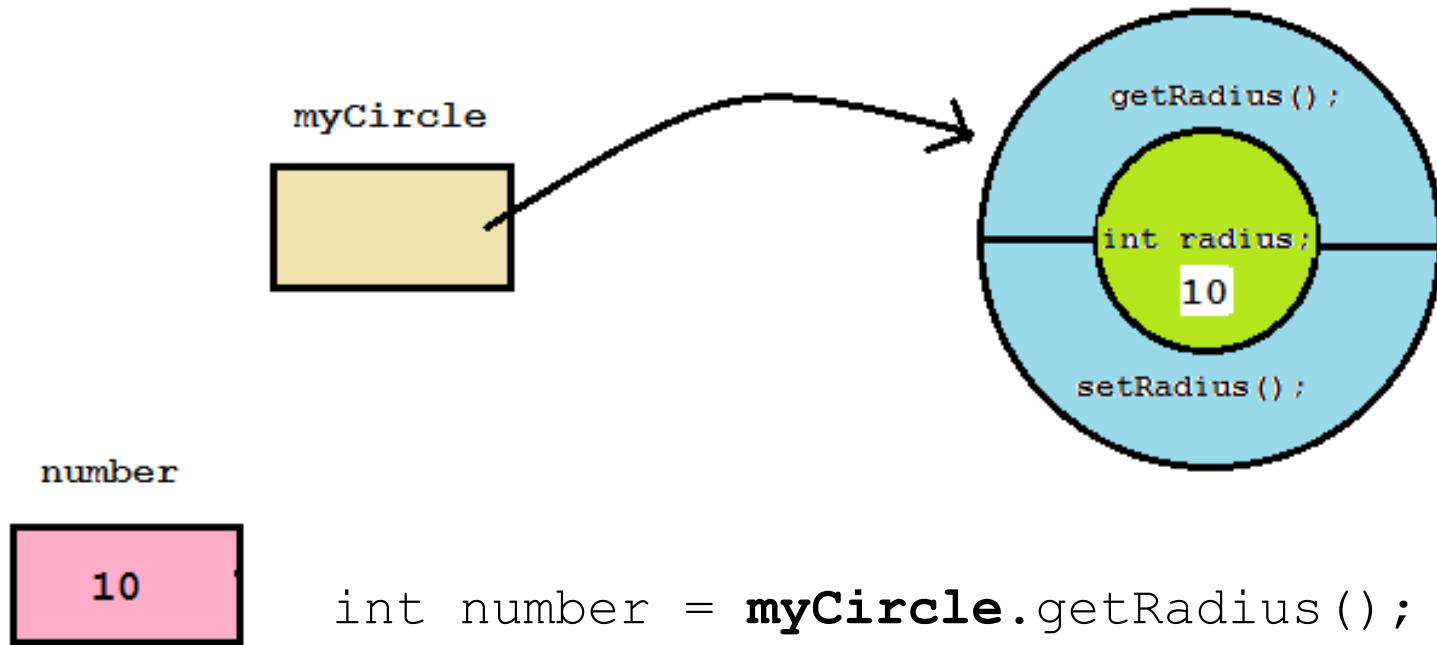
Step (2)



```
BasicCircle myCircle = new BasicCircle(5); // we could do all this on one line
```

Calling a public method **setRadius(10)** on the object **myCircle** in order **set** the value of the class variable **radius**

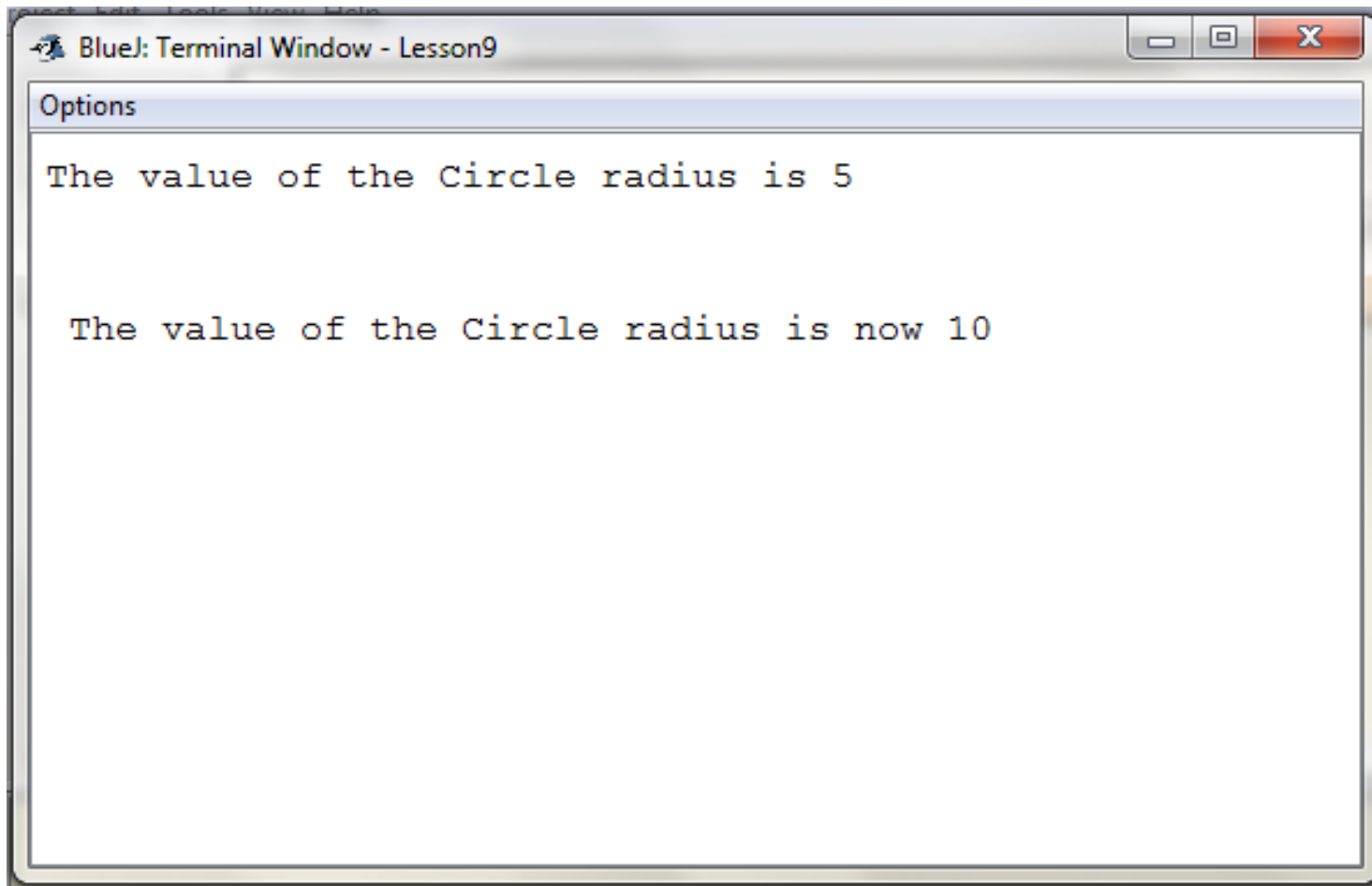
```
myCircle.setRadius(10); //change the radius to 10
```



The variable `number` will receive the value returned by the call to `getRadius()` on the object referenced by `myCircle`

`myCircle.radius = 10;` // Error, Wrong, why??

Output of **BasicCircleTester** : `setRadius` is used to change the value of the radius attribute of the object.



The screenshot shows a window titled "BlueJ: Terminal Window - Lesson9". Inside the window, there is a tab labeled "Options". The terminal output consists of two lines of text: "The value of the Circle radius is 5" followed by a blank line, and then "The value of the Circle radius is now 10".

```
Options  
  
The value of the Circle radius is 5  
  
The value of the Circle radius is now 10
```

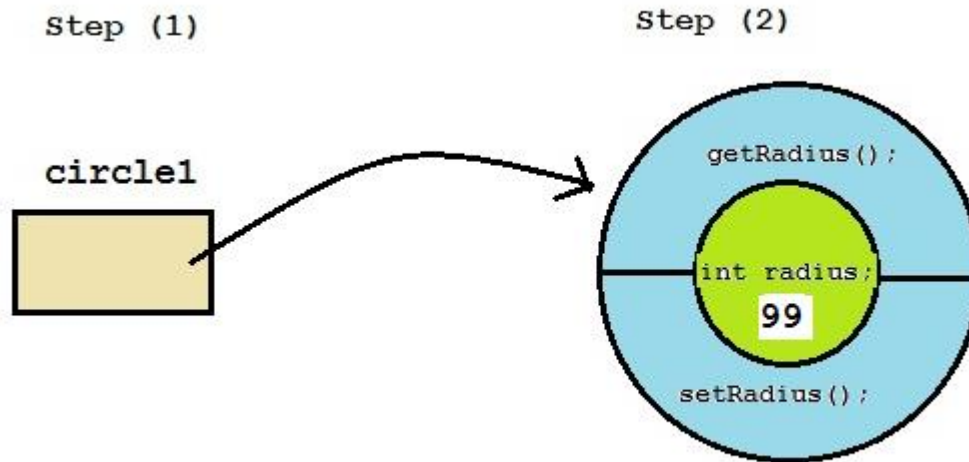
Why is it NOT correct to try and access the **radius** variable in the **myCircle** object?

- Remember that the class variable **radius** is declared using the access modifier **private**.
- When we try and access **radius** using the **dot** operator we will get an error because the access modifier **private** only permits the methods defined within the class to access radius.
- That is why we have **setRadius()** to set the value in the **radius** variable.
- And we have **getRadius()** to retrieve the value in the **radius** variable.

int x = myCircle.radius; //Error, Wrong, Why??

Three things need to happen for you to have a handle on an object.

```
BasicCircle circle1; //Step(1) declare reference variable  
circle1 = new BasicCircle(99); //Step(2) create the object  
//Step(2.1) the reference variable is linked to the object
```



```
circle1.setRadius(5); // Great method call!!
```

Couple of notes on writing Class constructor methods

```
public BasicCircle(int a) //NO return type for class constructor.
{
    x = a; // set the class attribute x to the value of a
}
```

- ☐ The Class constructor method name **is the same as the name of the Class** itself.
- ☐ The constructor method **never** returns a value or type although it appears to do so when you use it to create an object.
- ☐ The constructor needs to be accessible from outside the class. (**public**)
- ☐ You can have multiple versions of the class constructor method (**method overloading**) and when you are creating a new object you can pick one from those which are available.
- ☐ There are other rules concerning the class constructor but those listed above are the primary rules. (especially when it comes to OO inheritance)

Explicitly referring to class instance variable within a class using the key word **this** synonym.

```
public BasicCircle(int x) //NO return type for class constructor.  
{  
    this.x = x; //set class attribute x equal to formal variable x  
}
```

The key word **this** followed by the dot “.” operator allows you to specify which variable you are referring to explicitly. In the example above the key word **this** is used to tell the difference between the constructor **formal parameters x** and the **class instance variables x**.

this.x *refers to the class instance variable x*

Write a Class to represent a Dog (10 minutes)



Write your own class to represent a dog and name the class **Dog**

The **Dog** class should have class variables as follows

String **breed**; // the breed of the dog e.g. terrier.

boolean **isDangerous**; // to indicate whether or not the dog is dangerous.

int **age**; // the age of the dog

The class should have one constructor

- 1) A constructor which takes parameters to give values to the class attributes.
- 2) The class should have getter and setter methods for each of the class attributes
- 3) The class should have a separate method named **bark**. This method should simply print out the sentence “Woof! The dog has barked” when the method bark is called.
- 4) Think about using this new class definition to create two different dogs and get both of your dogs to bark for you.

Write a Class to represent a Point (x,y) (7 minutes)

Write your own class to represent a point in space and name it **Point**

The class should have class variables as follows

```
int x; // the x coordinate  
int y; // the y coordinate
```

The class should have one constructor

- 1) A constructor which takes parameters to give values to the class attributes.
- 2) The class should have getter and setter methods for each of the class attributes
- 3) The class should have a separate method named **printPoint**. This method should print out both attributes to the screen e.g. "Point coordinates are (5,10)"
- 4) Write code to create two **Point** objects using the **Point** class and call the **printPoint** method on both of these objects to print the coordinates.

Constructor method Overloading.

Notice: there are **two versions** of the **Circle constructor** method with different signatures, defined with the same Circle class. You can do this with other methods too as long as the formal parameters in the method signatures are different in types and or number of parameters.

```
public Circle(int a,int b) //NO return type for class constructor.
{
    x = a; // set the class attribute x equal to a
    y = b; // set the class attribute y equal to b
    circleCounter++; // counts number of circles created.
}
```

```
public Circle(int a,int b, int rad, String col)
{
    x = a; // set the class attribute x equal to a
    y = b; // set the class attribute y equal to b
    radius = rad; // set class attribute radius equal to rad
    colour = col; // set class attribute colour equal to col
    circleCounter++; // counts number of circles created.
}
```

Couple of notes on writing Class constructor methods

```
public Circle(int a,int b) //NO return type for class constructor.
{
    x = a; // set the class attribute x to the value of a
    y = b; // set the class attribute y to the value of b
    circleCounter++; // counts number of circles created.
}
```

- ☐ The Class constructor method name **is the same as the name of the Class** itself.
- ☐ The constructor method **never** returns a value or type although it appears to do so when you use it to create an object.
- ☐ The constructor needs to be accessible from outside the class. (**public**)
- ☐ You can have multiple versions of the class constructor method (**method overloading**) and when you are creating a new object you can pick one from those which are available.
- ☐ There are other rules concerning the class constructor but those listed above are the primary rules.

Basic principles and concepts used in object oriented programming.

Class Scope	This is when a variable (attribute) is declared just inside the first line of a class after the first brace {. The scope of this variable covers the whole class which includes any methods inside the class. (NB: revise scope rules).
Data encapsulation	This OO concept is achieved by bundling the methods (behaviour) and the variable (attributes) of a class representation together in the class.
private	This is an access modifier (a java keyword) which is put in front of your class variables declarations (variables declared just inside your class). Only methods defined inside a class definition can have access to or are able to change the value of variable (attributes) declared with the access modifier private.
Information hiding	Information hiding is the principle OO concept of only allowing the methods defined within a class to have access to or change the values of the attributes of the class. Achieved by declaring the class attribute variables to using the access modifier private in front of their declarations.
Instance / object	This is comes about when you use a class definition or specifically a call to the constructor in a class definition to create a concrete object in memory which has its own attribute data and its own methods encapsulated into a bundle (the object). Remember that you can use the class definition repeatedly to create many objects in the same way as you can use a template to create many cookies. ("template is to cookie as class is to object"). Remember also that you have to have links to these objects using object reference variables.

Basic principles and concepts used in object oriented programming.

public	This is an access modifier (a java keyword) which is put in front of your methods defined within your class. This allows other code to call these methods using an instance of an object and the dot “.” operator followed by the method name. (Calling methods in this way allows us to indirectly change or retrieve the values of private class variables). You allow the public to have a way of accessing your attributes; you provide a public interface (set of publicly available methods) to access your attributes (class variables).
Inheritance	When a class definition reuses methods and attributes from another class definition by inheriting that classes definition. Java allows the programmer to reuse class definitions and extend the functionality in them by allowing one class (subclass) to inherit from another class (parent or superclass). This is a very important ability because it allows to programmer to develop and extend software which is dependent on previously proven super classes.
Overriding Methods	When a method is overridden in a subclass, and that method is invoked (called) on an instance (object) of the subclass, it is the overridden version of the method that is executed. The subclass method must have the same name and parameter types as the superclass method it overrides. The superclass method is still there but it is hidden or overridden by the subclass method.