CS620c Introduction to Objected Oriented Programming
Supplementary
OO continued, creating arrays of objects, passing an object reference to a method, and introducing OO inheritance.

Joe Duffin

Eolas Building Room 1.45

Email: Joseph.Duffin@nuim.ie

So far you have covered the first principles of Object Oriented programming!

- Introducing the first principles of Object Oriented Programming. √

Designing your own OO class – **variables and methods** - **encapsulation**
private class variables – **information hiding**
public interface to the class – **class methods** – **information hiding**
Class **constructor** methods
Class constructor **method overloading**
Creating an object using the **class constructor method**.
Calling an object's method using:

objectReferenceVariable.methodName( )

- Further Object Oriented Programming, creating arrays of objects.

- Introducing basic inheritance.

Repeated : 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.

Repeat: 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	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	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**.**

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)

Declaring a Class reference variable and creating an 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

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.

Calling methods on a newly created object

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

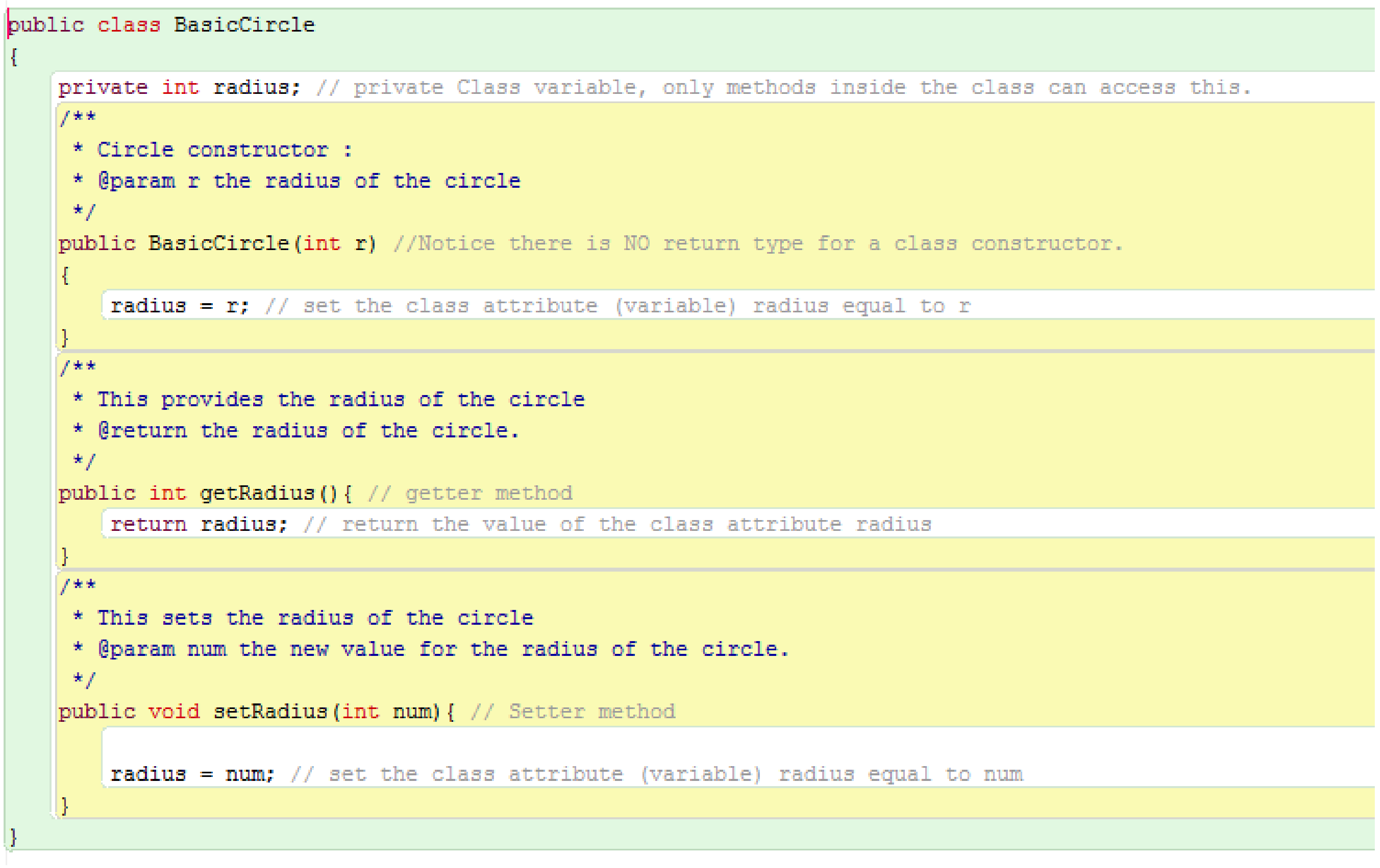
```
int x = Identifier.getterMethodName( );
```

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
```

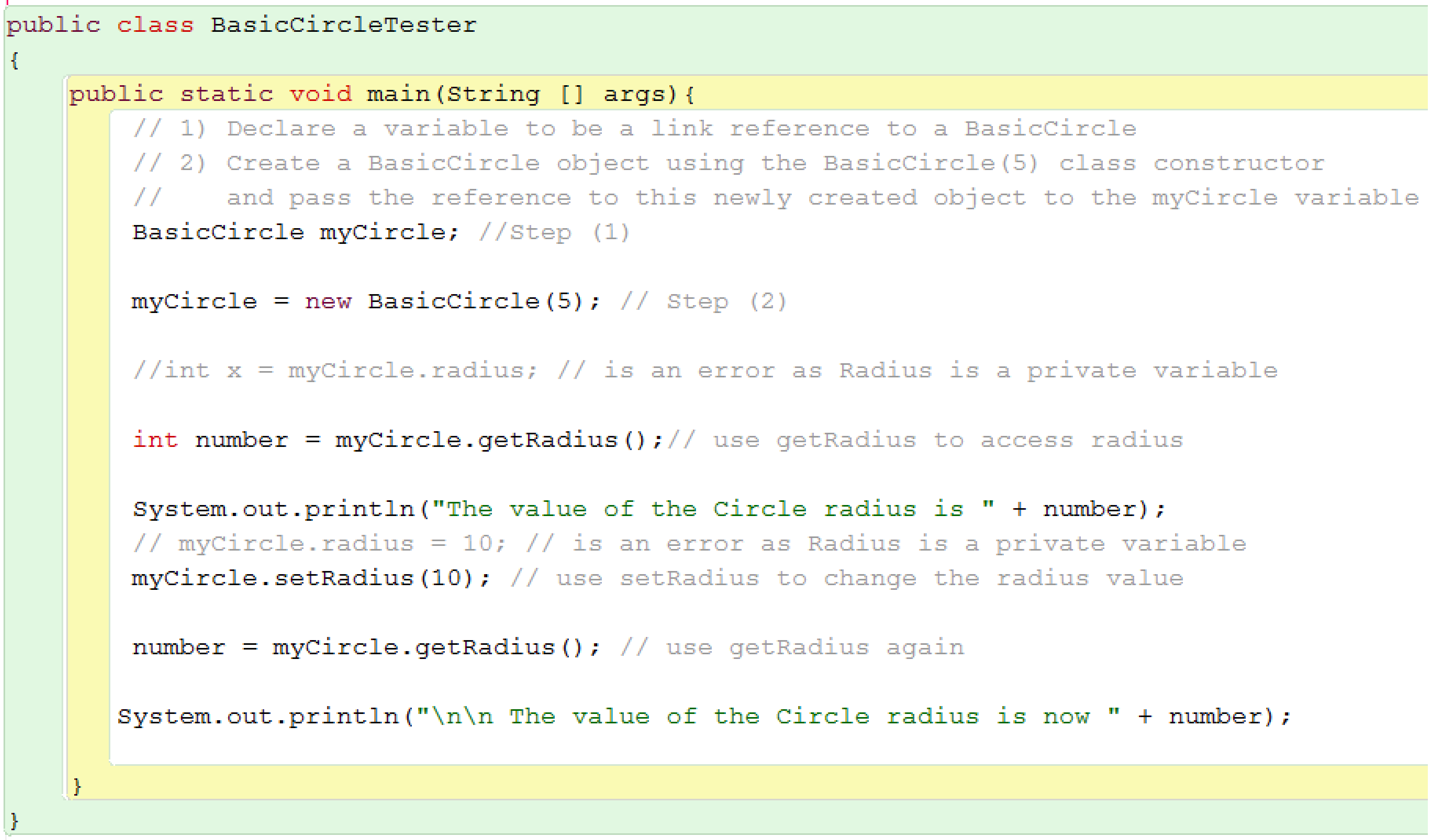
```
int number = myCircle.getRadius(); //getter
```

BasicCircle Class: with a single class attribute



```
public class BasicCircle
{
    private int radius; // 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)

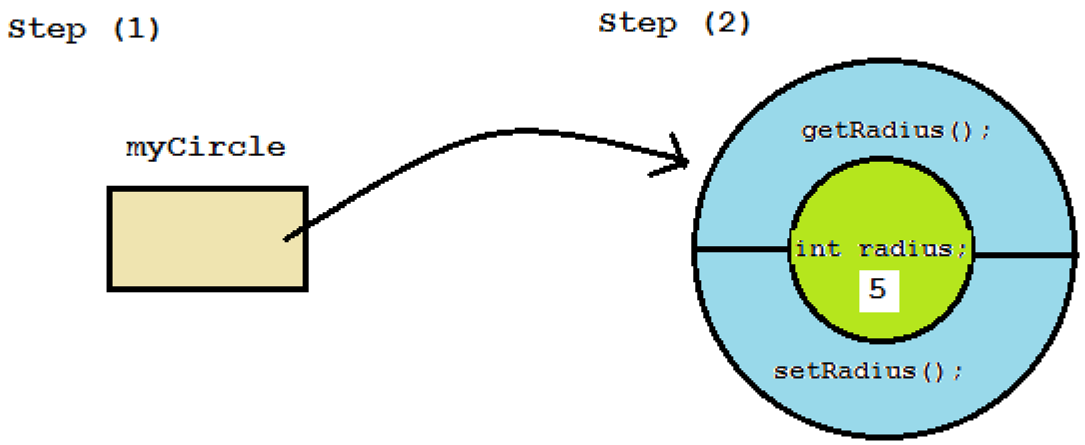


```
public class BasicCircleTester
{
    public static void main(String [] args){
        // 1) Declare a variable to be a link reference to a BasicCircle
        // 2) Create a BasicCircle object using the BasicCircle(5) class constructor
        //    and pass the reference to this newly created object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

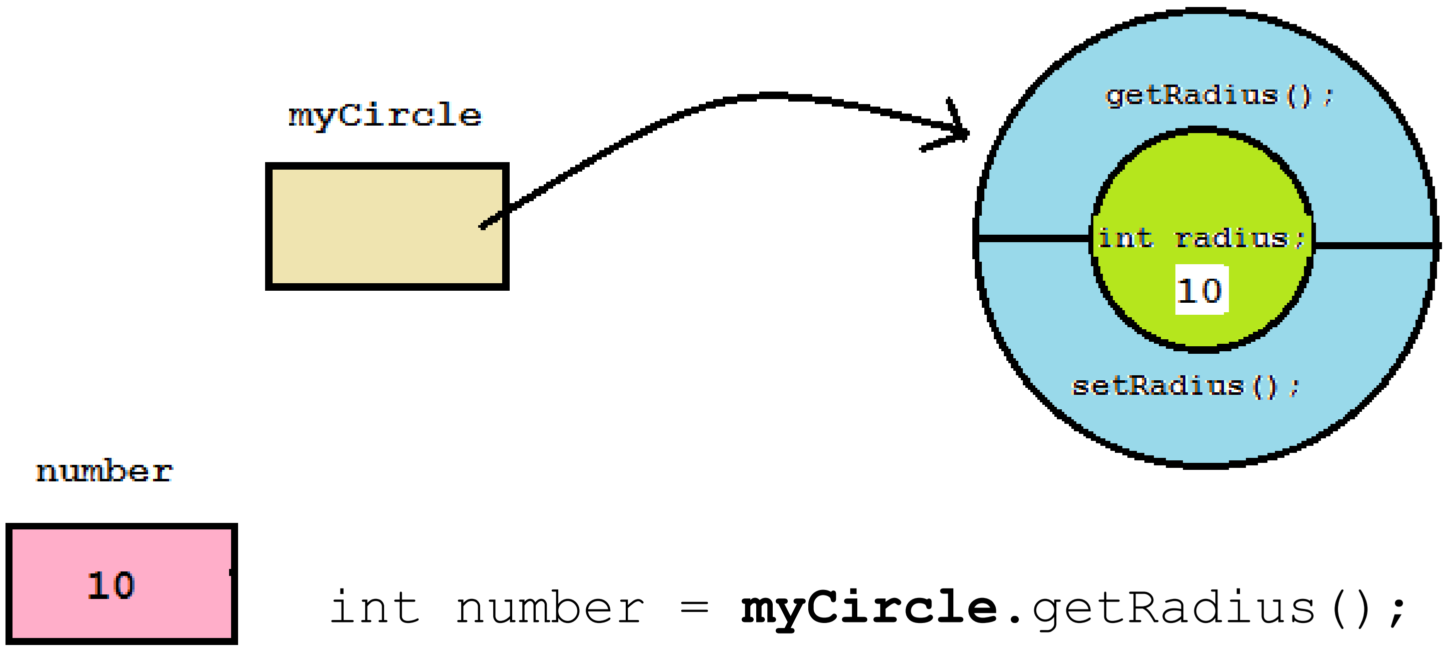
```
myCircle = new BasicCircle(5); //Step(2) create object
```



```
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**

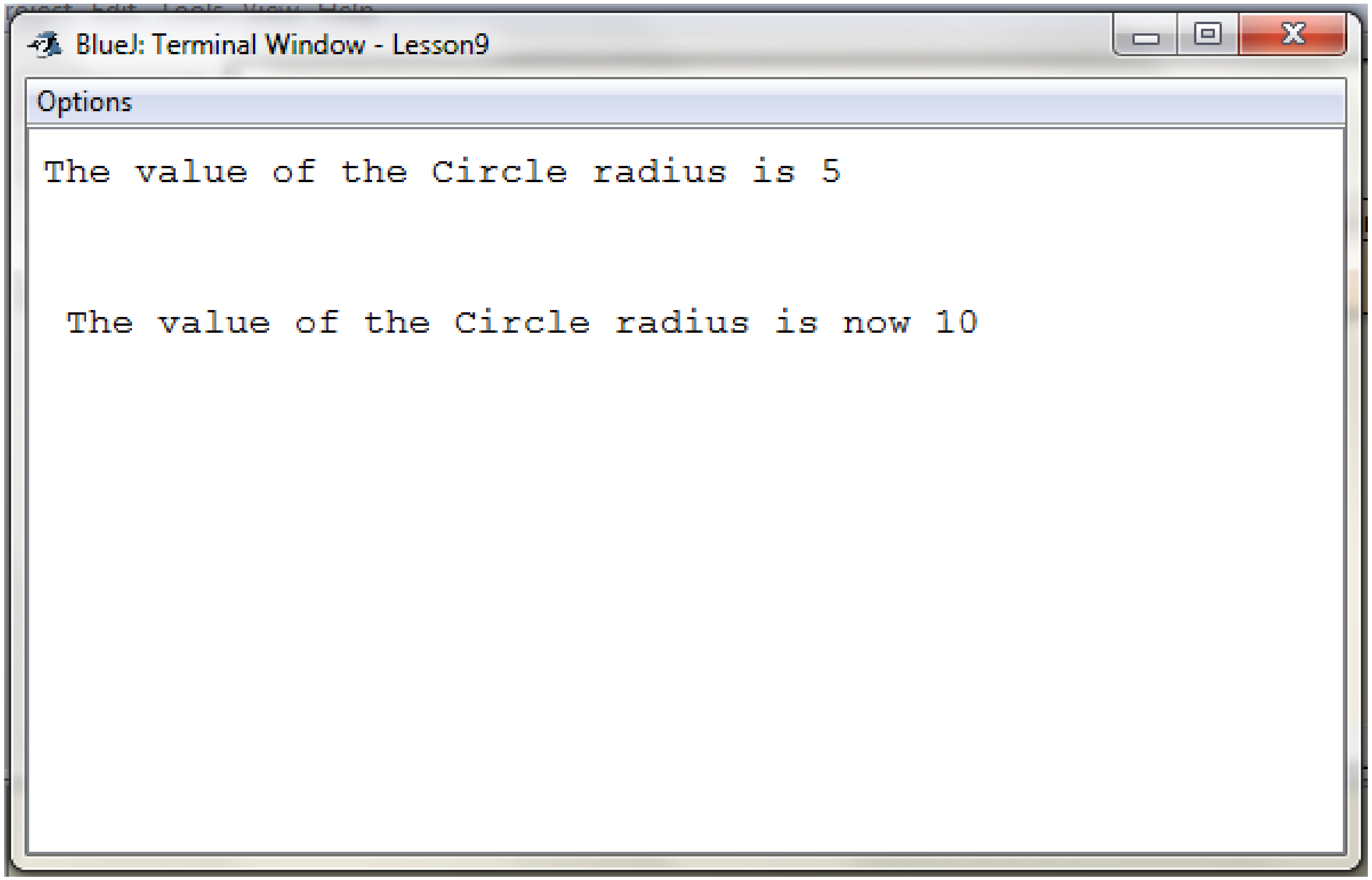
```
myCircle.setRadius(10); //change the radius to 10
```



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

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

Output of BasicCircleTest : setRadius is used to change the value of the radius attribute of the object.



```
The value of the Circle radius is 5



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

Why is it incorrect 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.

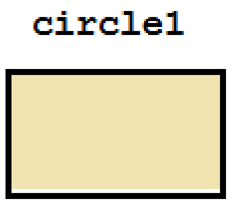
- That is why we have **setRadius()** to set the value in the radius variable.

- And we have **getRadius()** to retrieve the value in the radius variable.

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

What is wrong here?

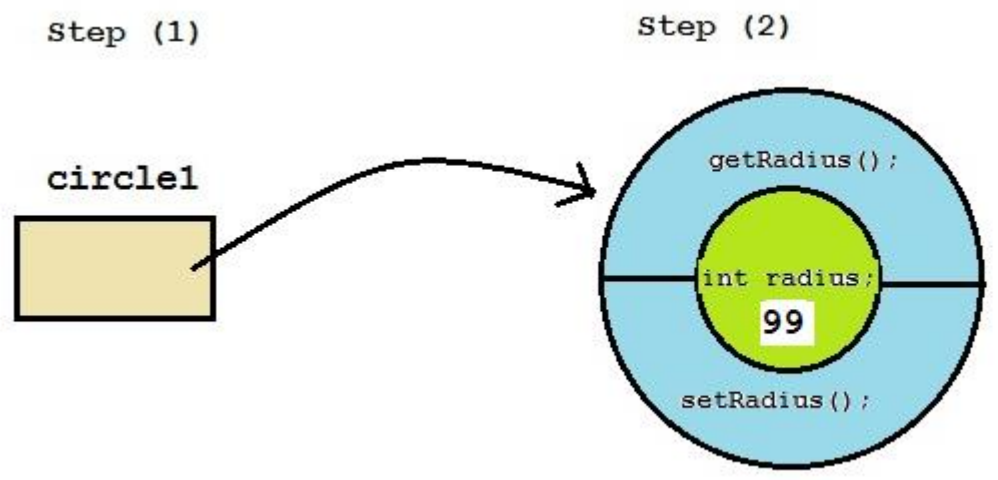
```
BasicCircle circle1; //Step(1)declare reference variable
```



```
circle1.setRadius(5); //?????
```

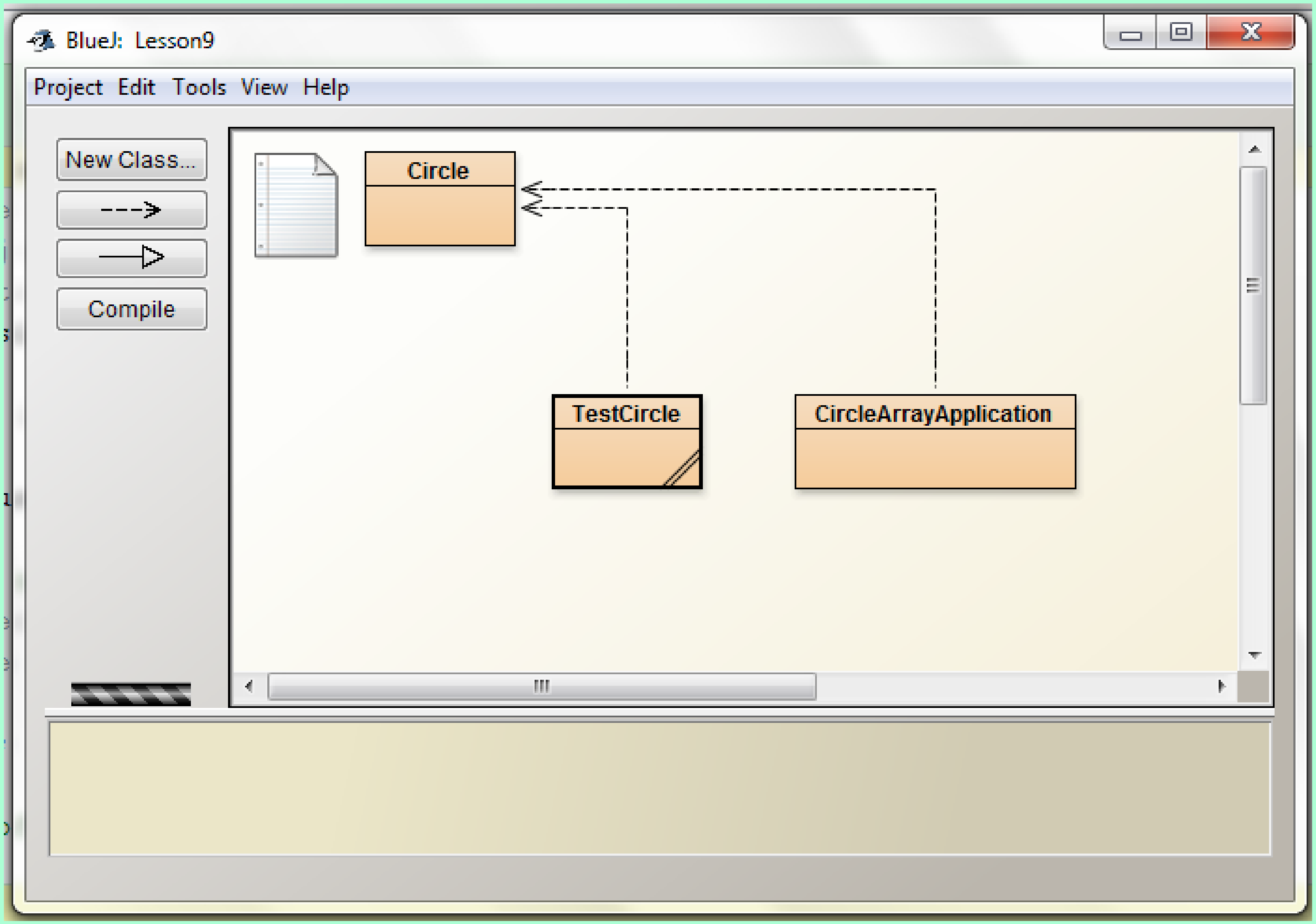
You need to do two things.

```
BasicCircle circle1; //Step(1)declare reference variable

circle1 = new BasicCircle(99); //Step(2)create the object
```

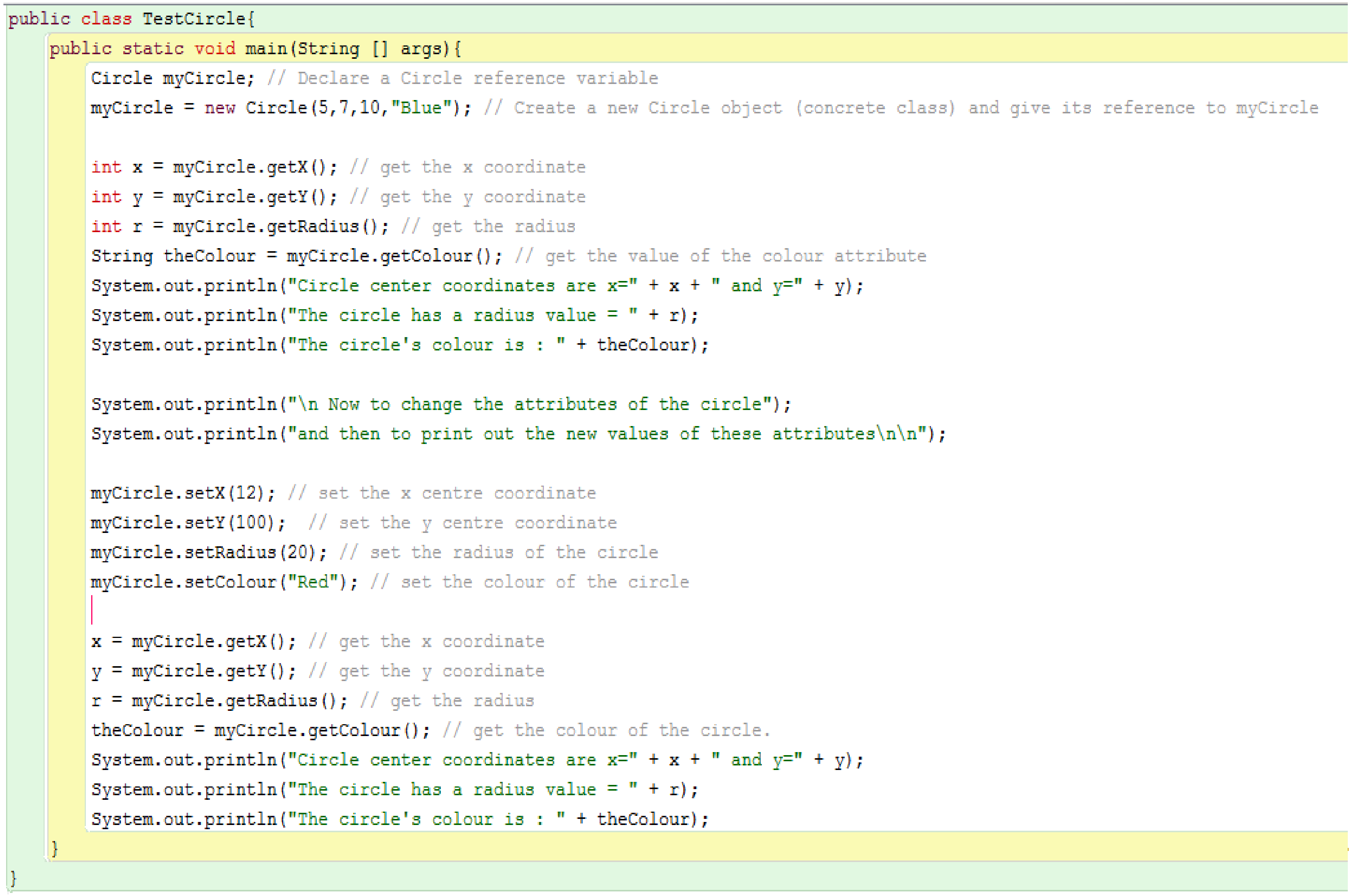


```
circle1.setRadius(5); // Great method call!!
```

Improved **Circle** Class with two classes **TestCircle** and **CircleArrayApplication** which creates an array of objects of **Circle Type**

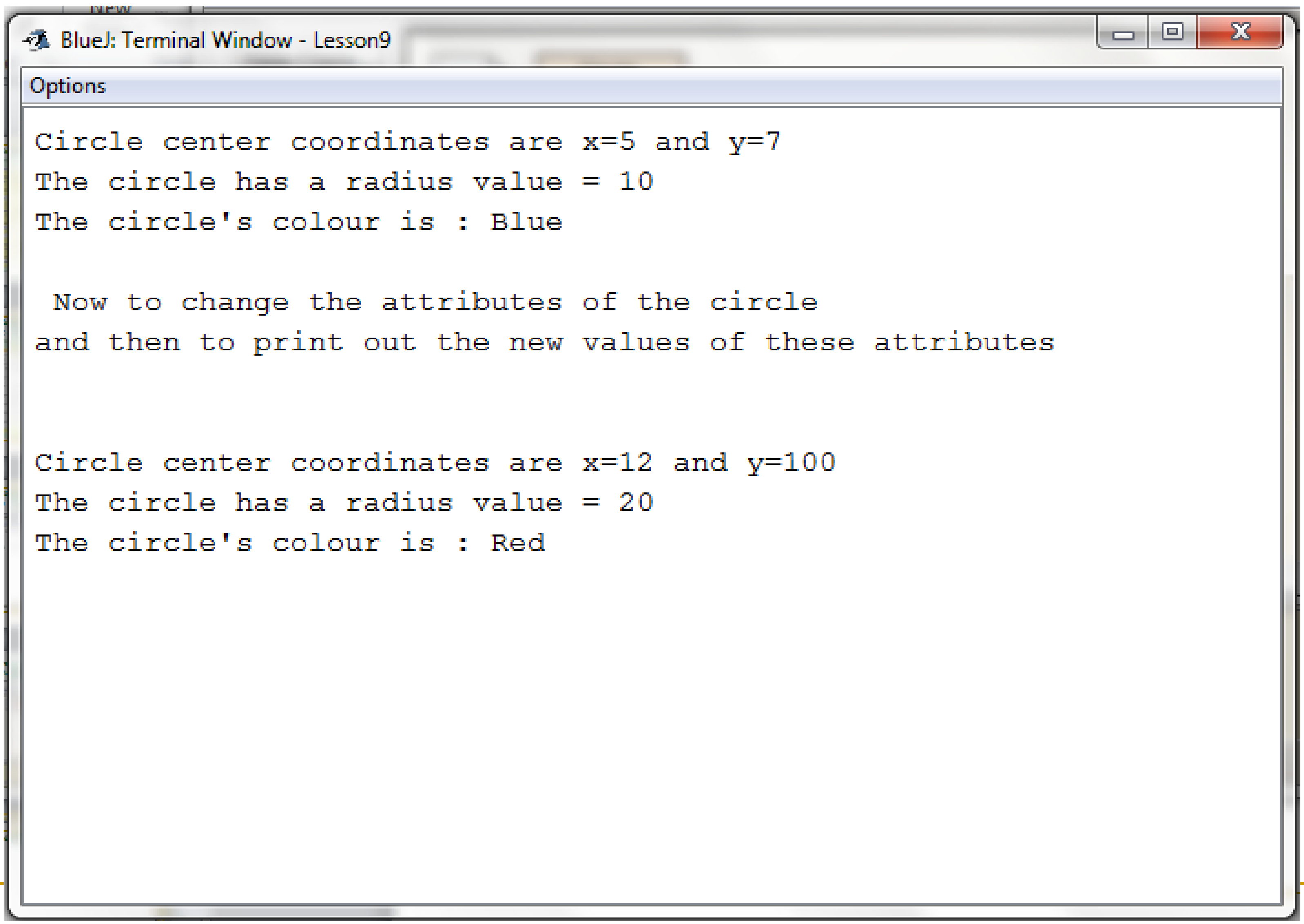


TestCircle: creates a single Circle object



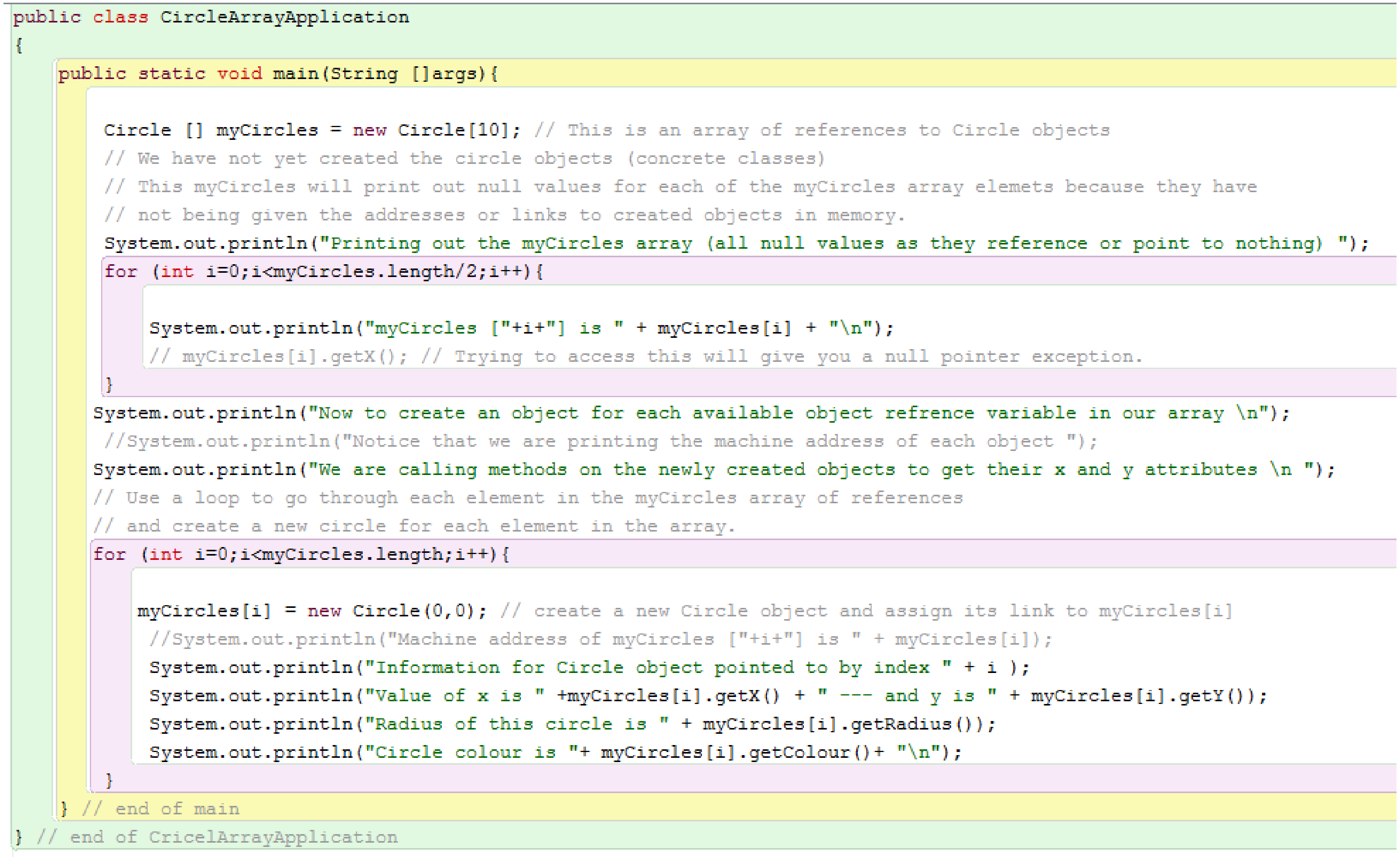
```
public class TestCircle{
    public static void main(String [] args){
        Circle myCircle; // Declare a Circle reference variable
        myCircle = new Circle(5,7,10,"Blue"); // Create a new Circle object (concrete class) and give its reference to myCircle

        int x = myCircle.getX(); // get the x coordinate
        int y = myCircle.getY(); // get the y coordinate
        int r = myCircle.getRadius(); // get the radius
        String theColour = myCircle.getColour(); // get the value of the colour attribute
        System.out.println("Circle center coordinates are x=" + x + " and y=" + y);
        System.out.println("The circle has a radius value = " + r);
        System.out.println("The circle's colour is : " + theColour);

        System.out.println("\n Now to change the attributes of the circle");
        System.out.println("and then to print out the new values of these attributes\n\n");

        myCircle.setX(12); // set the x centre coordinate
        myCircle.setY(100);  // set the y centre coordinate
        myCircle.setRadius(20); // set the radius of the circle
        myCircle.setColour("Red"); // set the colour of the circle

        x = myCircle.getX(); // get the x coordinate
        y = myCircle.getY(); // get the y coordinate
        r = myCircle.getRadius(); // get the radius
        theColour = myCircle.getColour(); // get the colour of the circle.
        System.out.println("Circle center coordinates are x=" + x + " and y=" + y);
        System.out.println("The circle has a radius value = " + r);
        System.out.println("The circle's colour is : " + theColour);
    }
}
```

Output of TestCircle



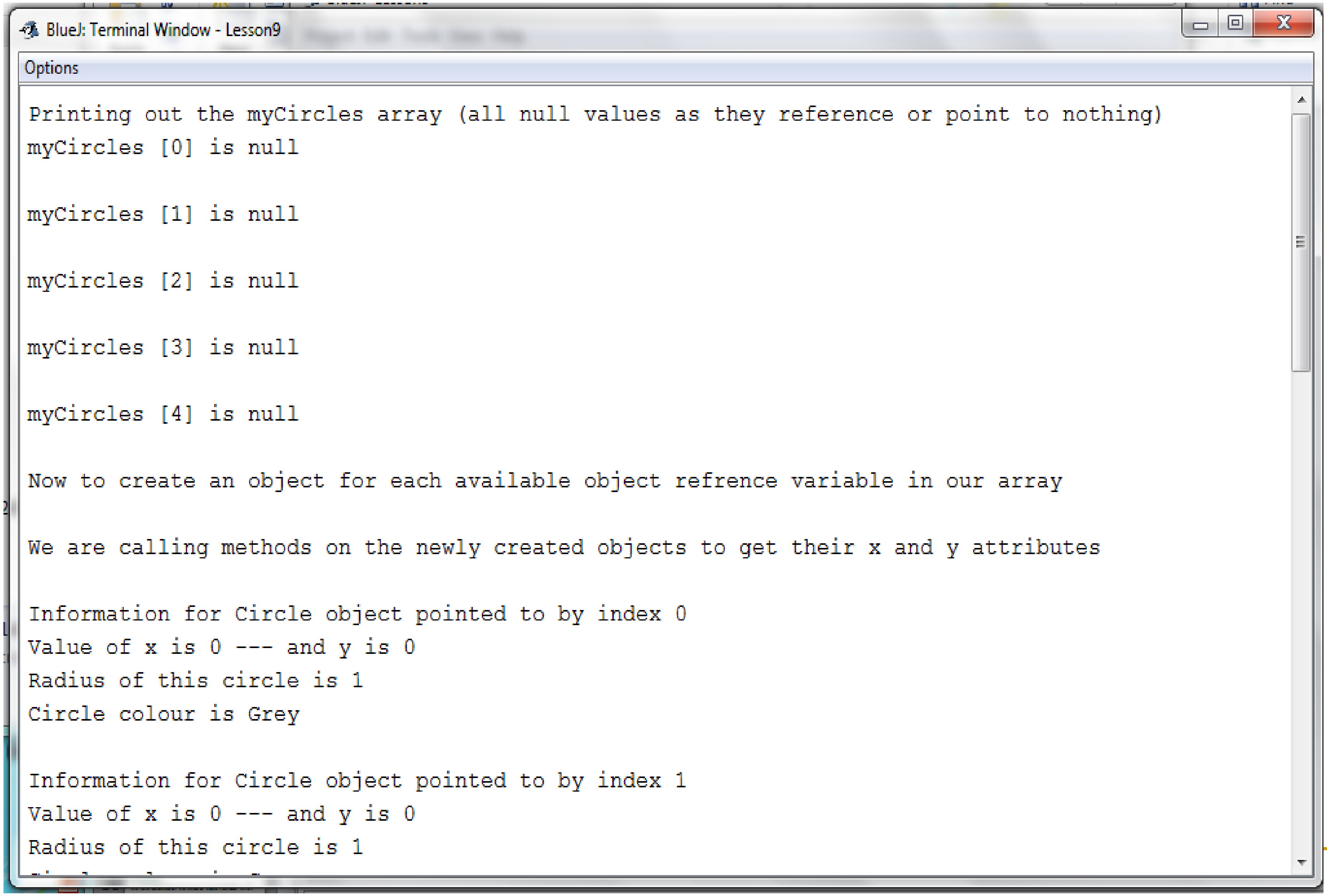
```
Circle center coordinates are x=5 and y=7
The circle has a radius value = 10
The circle's colour is : Blue

 Now to change the attributes of the circle
and then to print out the new values of these attributes


Circle center coordinates are x=12 and y=100
The circle has a radius value = 20
The circle's colour is : Red
```

CircleArrayApplication: creates an array of circle objects



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

        Circle [] myCircles = new Circle[10]; // This is an array of references to Circle objects
        // We have not yet created the circle objects (concrete classes)
        // This myCircles will print out null values for each of the myCircles array elemets because they have
        // not being given the addresses or links to created objects in memory.
        System.out.println("Printing out the myCircles array (all null values as they reference or point to nothing) ");
        for (int i=0;i<myCircles.length/2;i++){

            System.out.println("myCircles ["+i+"] is " + myCircles[i] + "\n");
            // myCircles[i].getX(); // Trying to access this will give you a null pointer exception.
        }
       System.out.println("Now to create an object for each available object refrence variable in our array \n");
        //System.out.println("Notice that we are printing the machine address of each object ");
       System.out.println("We are calling methods on the newly created objects to get their x and y attributes \n ");
       // Use a loop to go through each element in the myCircles array of references
       // and create a new circle for each element in the array.
       for (int i=0;i<myCircles.length;i++){

           myCircles[i] = new Circle(0,0); // create a new Circle object and assign its link to myCircles[i]
            //System.out.println("Machine address of myCircles ["+i+"] is " + myCircles[i]);
            System.out.println("Information for Circle object pointed to by index " + i );
            System.out.println("Value of x is " +myCircles[i].getX() + " --- and y is " + myCircles[i].getY());
            System.out.println("Radius of this circle is " + myCircles[i].getRadius());
            System.out.println("Circle colour is "+ myCircles[i].getColour()+ "\n");
        }
    } // end of main
} // end of CricelArrayApplication
```

Output of CircleArrayApplication

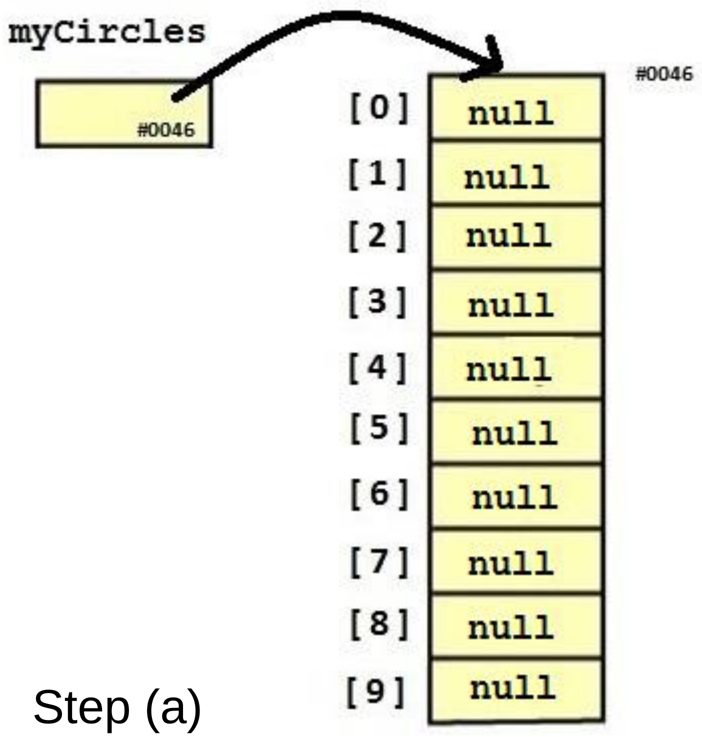


```
Printing out the myCircles array (all null values as they reference or point to nothing)
myCircles [0] is null

myCircles [1] is null

myCircles [2] is null

myCircles [3] is null

myCircles [4] is null

Now to create an object for each available object refrence variable in our array

We are calling methods on the newly created objects to get their x and y attributes

Information for Circle object pointed to by index 0
Value of x is 0 --- and y is 0
Radius of this circle is 1
Circle colour is Grey

Information for Circle object pointed to by index 1
Value of x is 0 --- and y is 0
Radius of this circle is 1
```

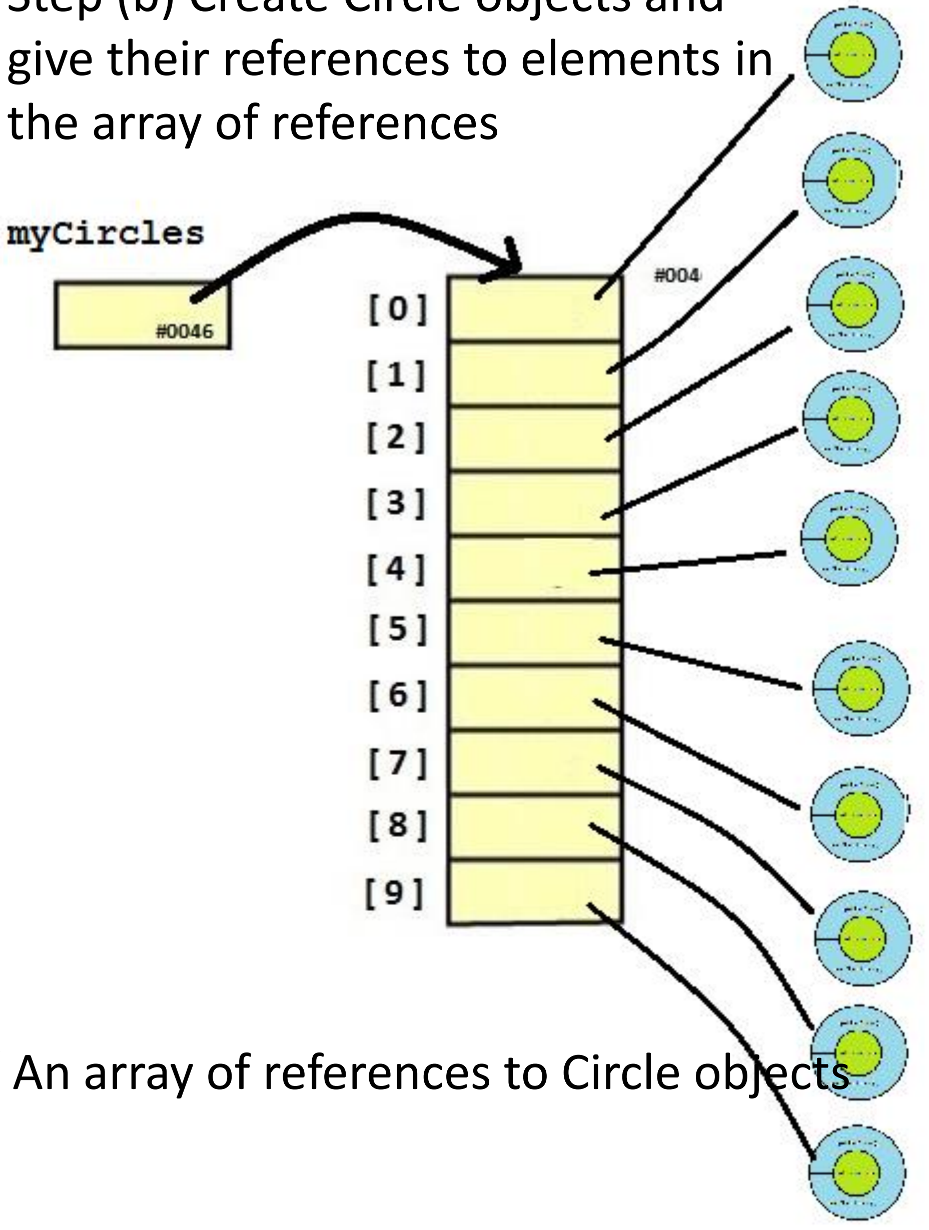
Creating an array of Circle objects

Step (a) Declare an array of Circle references

Step (b) Create Circle objects and give their references to elements in the array of references



Step (a)



Circle [] myCircles = **new Circle**[10]; // An array of references to Circle objects

Creating objects. Step (b)

Use a loop to create a new Circle object linked to the myCircles reference array. Each of the objects is created with the Circle constructor method which takes only two parameters.

```
// loop through all myCircle array elements

for (int i=0;i < myCircles.length; i++)
{

   // create a new circle object using the Circle
   // constructor method and put its link reference
   // in to the current (ith) element of the
   // array of references using myCircles[i]

      myCircles[i] = new Circle(0,0);
}
```

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.
- The are other rules concerning the class constructor but those listed above are the primary rules.

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

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

   circleCounter++; // counts number of circles created.
}
```

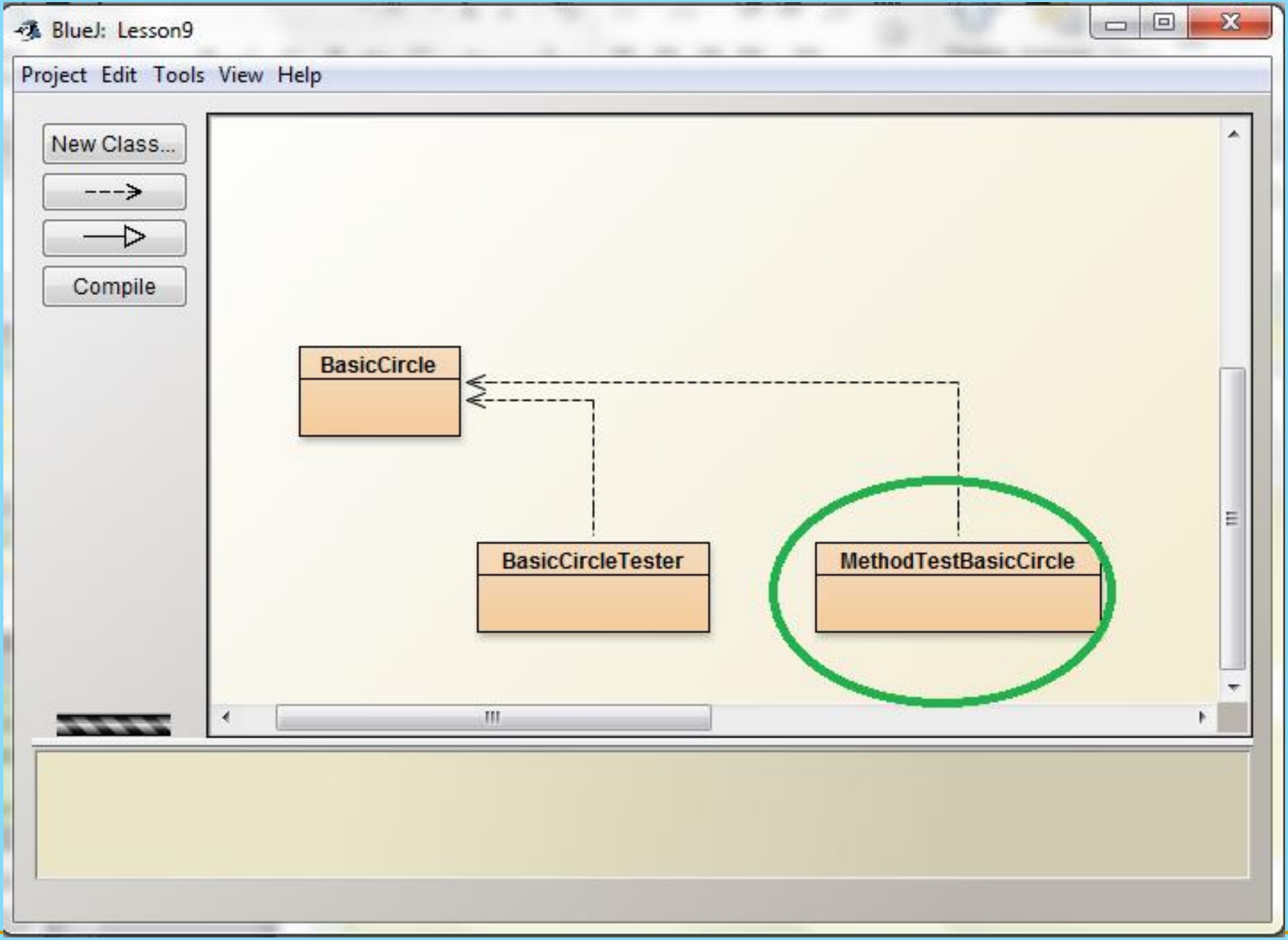
The key word **this** followed by the dot "." operator allows you to specify which variable you are referring to explicitly. In the example above the key word **this** is used to tell the difference between the constructor **formal parameters x and y** and the **class instance variables x and y.**

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

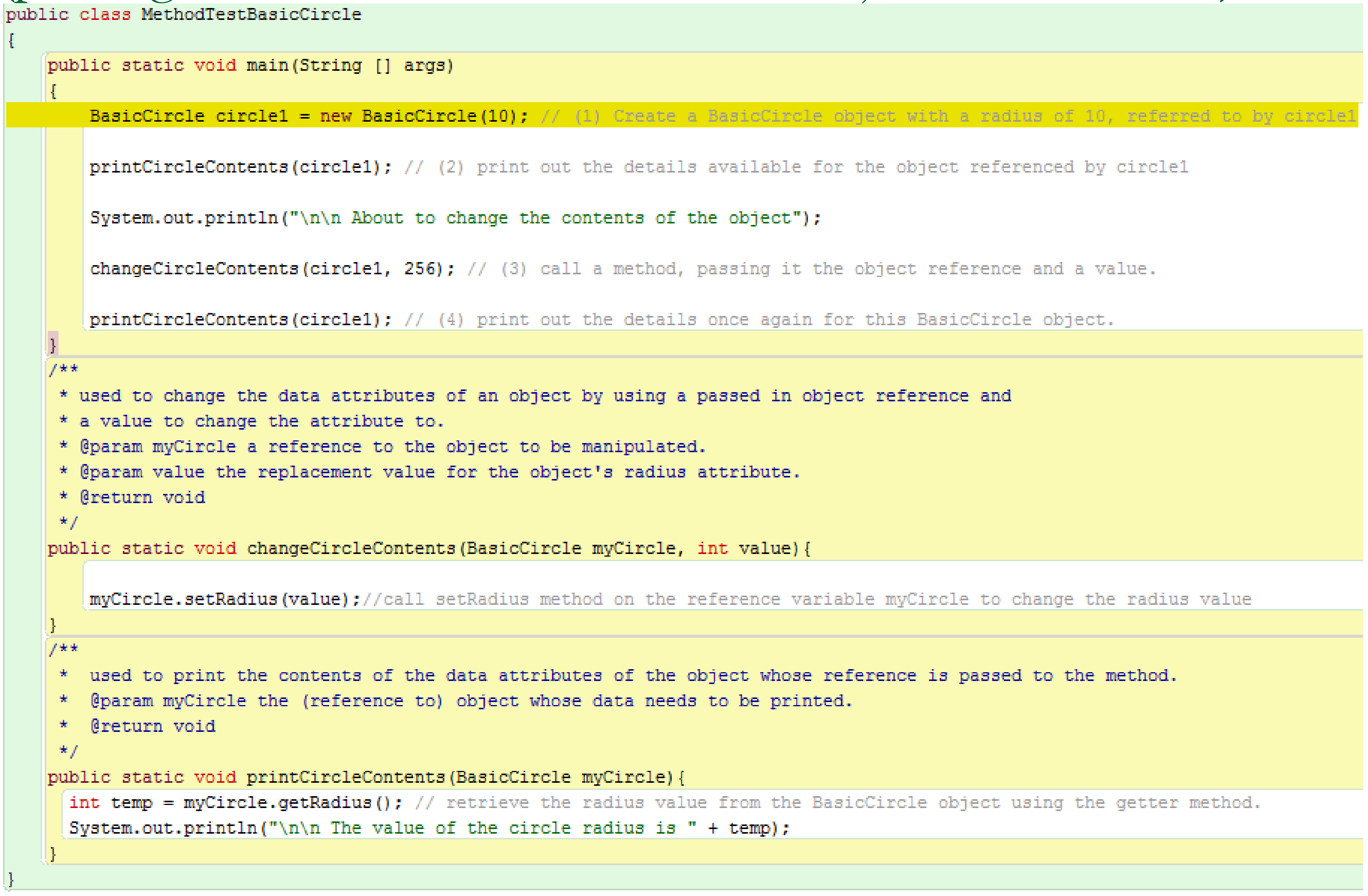
Calling methods on individual objects in the array of object references.

```
myCircles[i].setRadius(5);


myCircles[i].setColour("Blue");


int num1 = myCircles[i].getRadius();


int num2  = myCircles[i].getX();


int num2 = myCircles[i].getY();


String colour = myCircles[i].getColour();
```

Passing a BasicCircle object reference to a method.

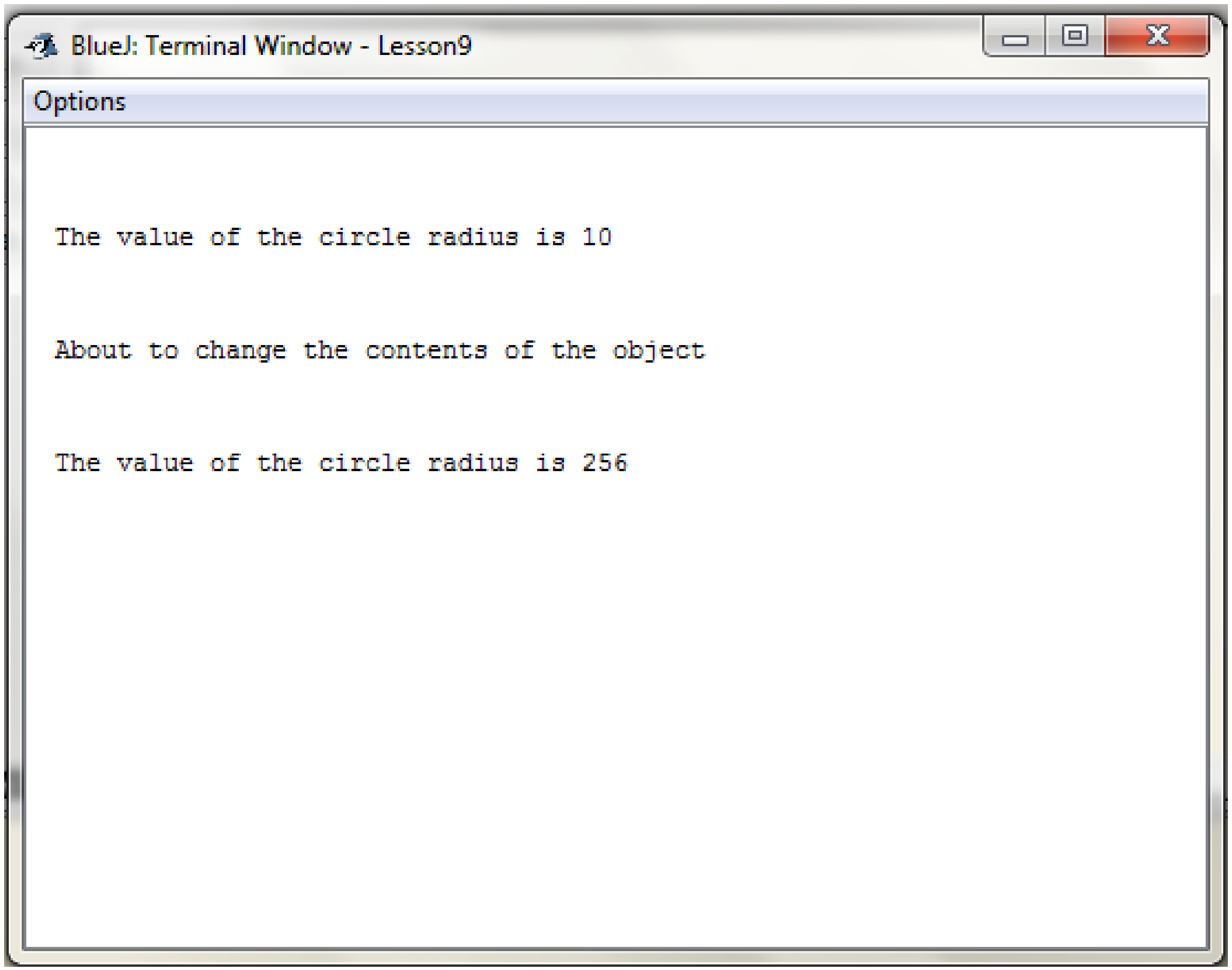


MethodTestBasicCircle: Passing a BasicCircle to a method (passing a reference to a BasicCircle object to a method)



```java
public class MethodTestBasicCircle
{
    public static void main(String [] args)
    {
        BasicCircle circle1 = new BasicCircle(10); // (1) Create a BasicCircle object with a radius of 10, referred to by circle1

        printCircleContents(circle1); // (2) print out the details available for the object referenced by circle1

        System.out.println("\n\n About to change the contents of the object");

        changeCircleContents(circle1, 256); // (3) call a method, passing it the object reference and a value.

        printCircleContents(circle1); // (4) print out the details once again for this BasicCircle object.
    }
    /**
     * used to change the data attributes of an object by using a passed in object reference and
     * a value to change the attribute to.
     * @param myCircle a reference to the object to be manipulated.
     * @param value the replacement value for the object's radius attribute.
     * @return void
     */
    public static void changeCircleContents(BasicCircle myCircle, int value){

        myCircle.setRadius(value);//call setRadius method on the reference variable myCircle to change the radius value
    }
    /**
     *  used to print the contents of the data attributes of the object whose reference is passed to the method.
     *  @param myCircle the (reference to) object whose data needs to be printed.
     *  @return void
     */
    public static void printCircleContents(BasicCircle myCircle){
      int temp = myCircle.getRadius(); // retrieve the radius value from the BasicCircle object using the getter method.
      System.out.println("\n\n The value of the circle radius is " + temp);
    }
}
```

Output of running main of class MethodTestBasicCircle



```
The value of the circle radius is 10

About to change the contents of the object

The value of the circle radius is 256
```

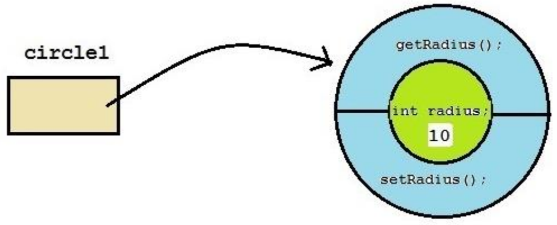
Passing a reference to a BasicCircle object to a method

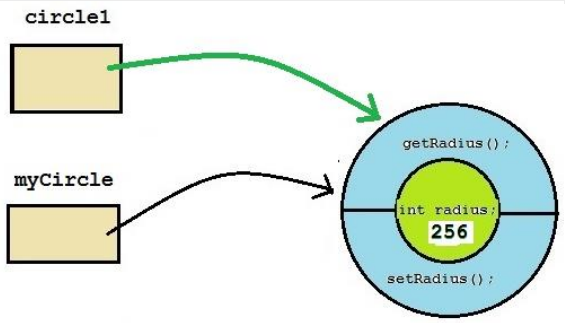
We have seen before how to pass an array reference to a method in order to change the contents of its array elements.
Passing an object to a method by a reference to it, uses exactly the same principles.

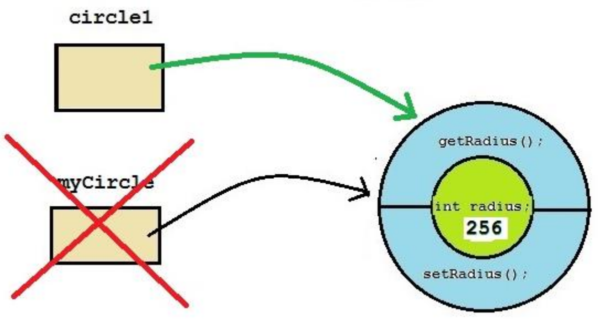
```
BasicCircle circle1 = new BasicCircle(10);
```

```
changeCircleContents( circle1, 256);
```

```
public static void changeCircleContents(BasicCircle myCircle, int value)
{
        myCircle.setRadius(value);
}
```







a) Create a **BasicCircle object** referenced by **circle1**

b) Call method **changeCircleContents** passing it the value of **circle1** and a value to change the **radius. Rem.** circle1 reference value is copied to myCircle

c) When the method is finished, variable **myCircle** is destroyed but **circle1** is still pointing to the object.

Inheritance: when a class reuses methods and attributes from another class 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 a another class (parent or superclass).

This is very important because it allows to programmer to develop software which is dependent on previously proven super classes.

```
public class Square extends TwoDShape
{
        ….
        ….
}
```

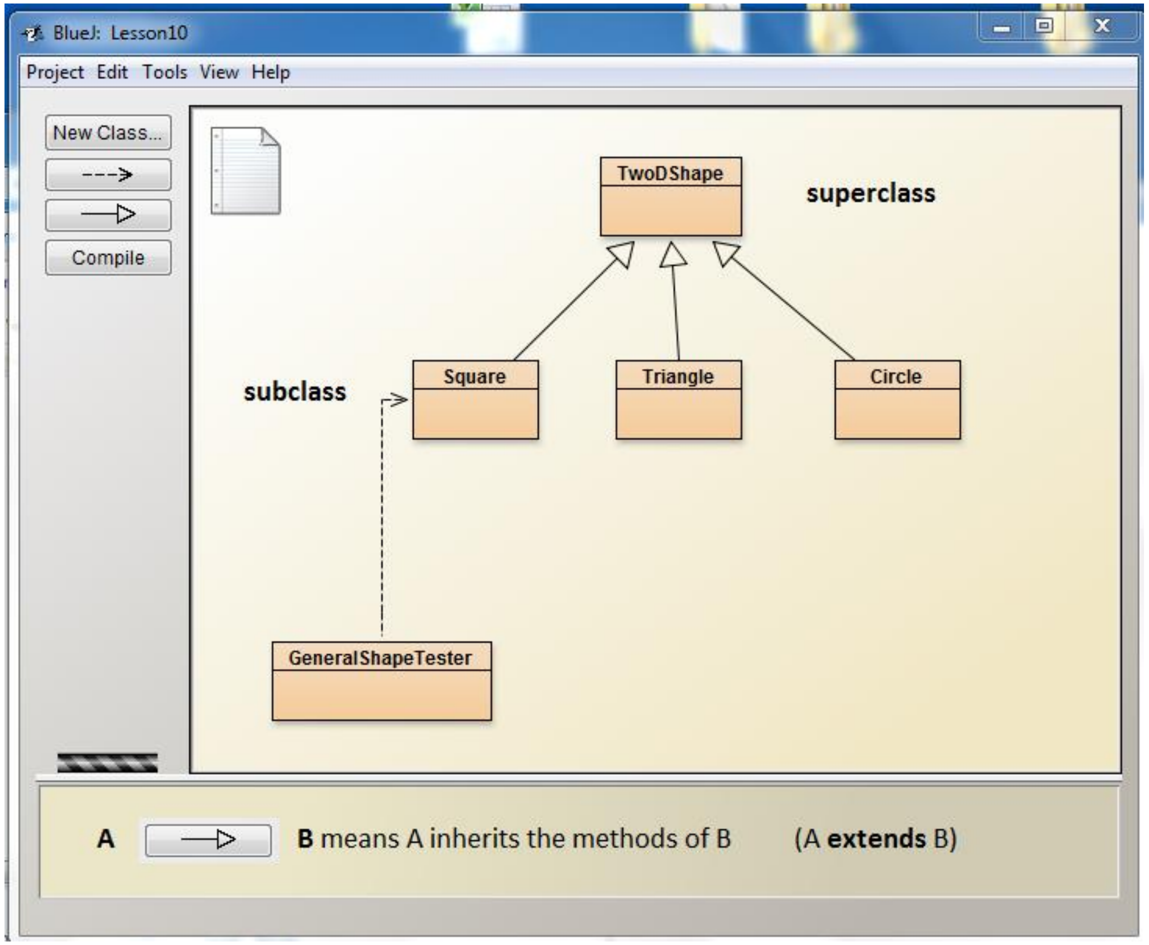
The class **Square** can **inherit** data and behaviour from the class **TwoDShape**

How does java indicate one class inherits from another?

```
public class Square extends TwoDShape
{
          ….
          ….
}
```

- Java uses the key word **extends** to indicate that a subclass inherits from a superclass.
- In the example above Square is the subclass and TwoDShape is the superclass.
- The first line of the Square declaration is the same as before but the key word **extends** is added followed by the name of the class to be inherited from. (TwoDShape).

Inheritance Class hierarchy showing the super class TwoDShape and three sub classes, Square, Triangle and Circle



A → **B** means A inherits the methods of B (A **extends** B)

The sub classes **Square**, **Triangle** and **Circle** are said to **inherit** the **methods** and have access to the **variables** of the **superclass** TwoDShape.

You can say that Square "is-a" TwoDShape, Triangle "is-a" TwoDShape and Circle "is-a" TwoDShape.

The class **GeneralShapeTester** creates instances of these subclasses to illustrate how methods are inherited.

The class Square has been redesigned to have a constructor which uses the constructor of the superclass.

- A subclass ***extends*** a superclass.

- A subclass **Inherits** all *public* Instance variables and methods of the superclass, but does **not** Inherit the *private* Instance variables and methods of the superclass,

- Inherited methods *can* be overridden; instance variables *cannot* be overridden (although they can be *redefined* in the subclass, but that's not the same thing, and there's almost never a need to do it)

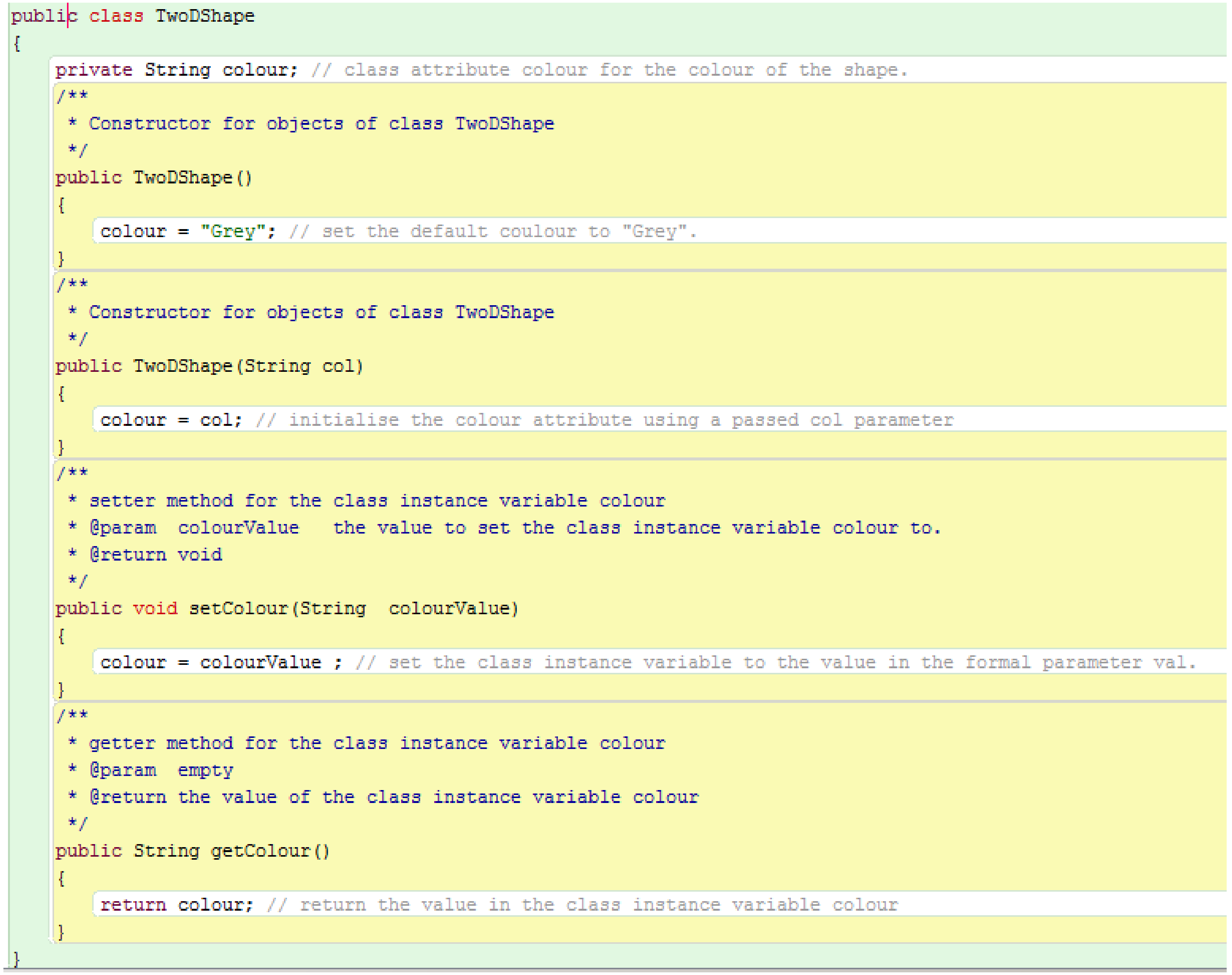
- Use the **IS-A** test to verify that your inheritance hierarchy is valid. If X ***extends*** Y, then X ***IS-A*** Y must make sense. (Square extends TwoDShape so Square **IS-A** TwoDShape is valid statement)

- When a method is overridden in a subclass, and that method is Invoked on an instance of the subclass, it is the overridden version of the method that is activated.

- If class B **extends** A, and C **extends** B, class B **IS-A** class A, and class C **IS-A** class B, and class C also **IS-A** class A.

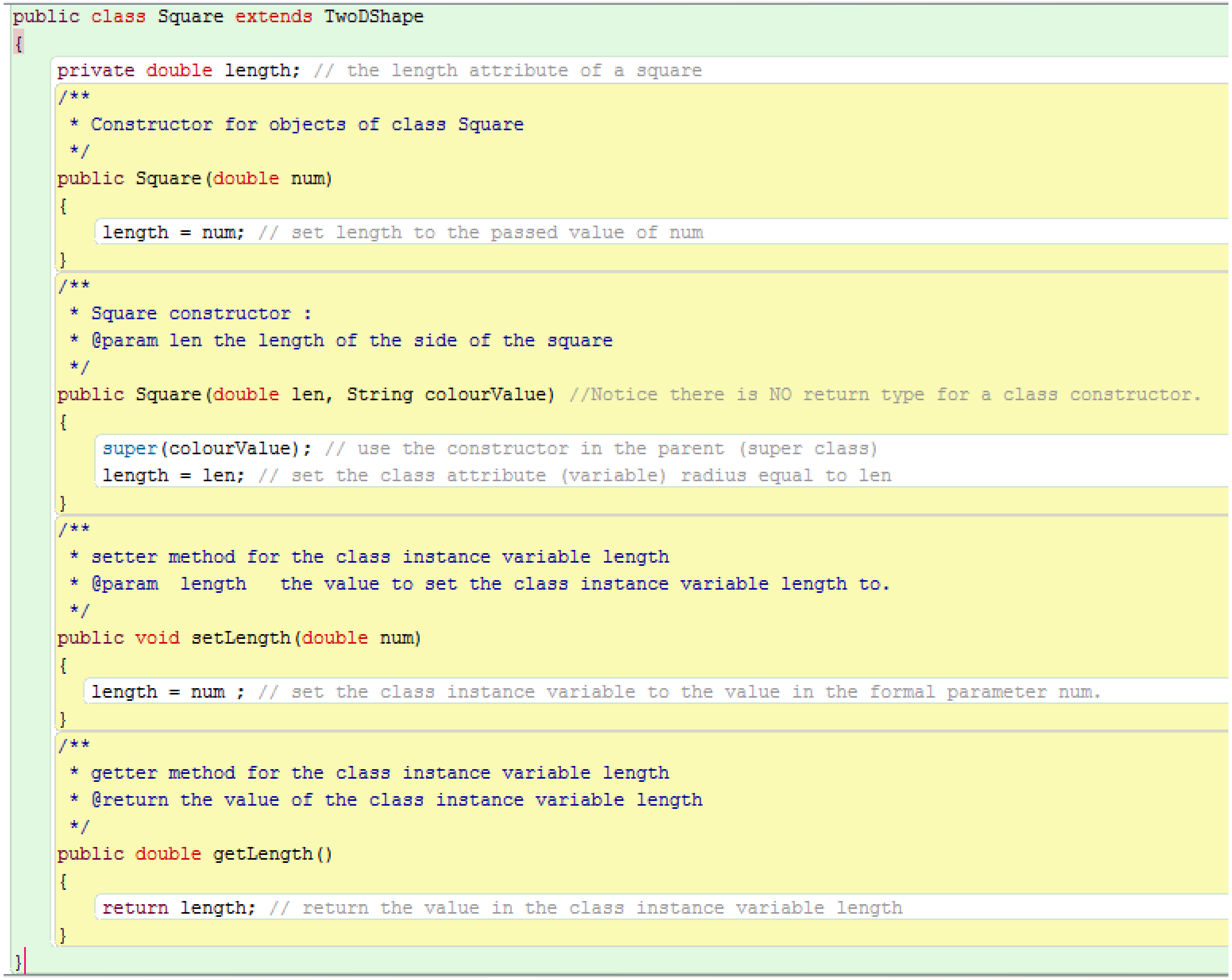
See Chapter 7 pages 181-191 “Head First Java” 2nd edition

TwoDShape; superclass



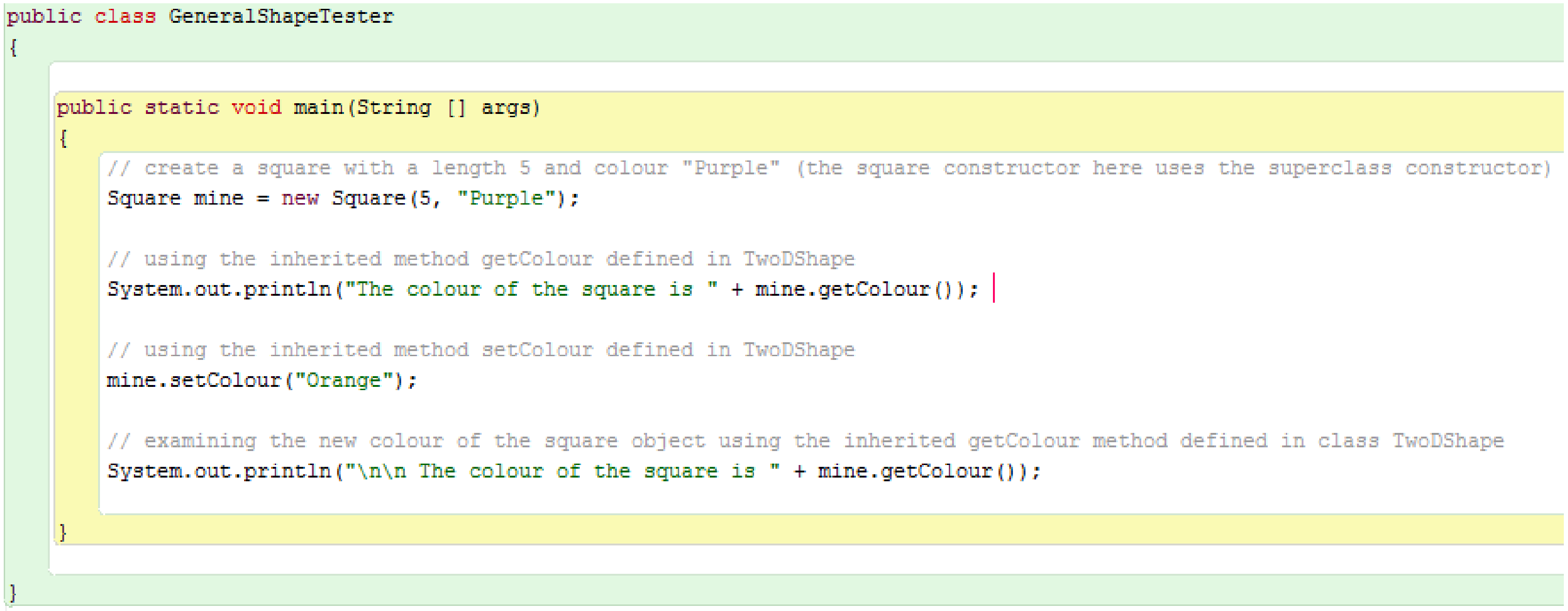
```
public class TwoDShape
{
    private String colour; // class attribute colour for the colour of the shape.
    /**
     * Constructor for objects of class TwoDShape
     */
    public TwoDShape()
    {
        colour = "Grey"; // set the default coulour to "Grey".
    }
    /**
     * Constructor for objects of class TwoDShape
     */
    public TwoDShape(String col)
    {
        colour = col; // initialise the colour attribute using a passed col parameter
    }
    /**
     * setter method for the class instance variable colour
     * @param  colourValue   the value to set the class instance variable colour to.
     * @return void
     */
    public void setColour(String  colourValue)
    {
        colour = colourValue ; // set the class instance variable to the value in the formal parameter val.
    }
    /**
     * getter method for the class instance variable colour
     * @param  empty
     * @return the value of the class instance variable colour
     */
    public String getColour()
    {
        return colour; // return the value in the class instance variable colour
    }
}
```

Square Class which inherits from TwoDShape

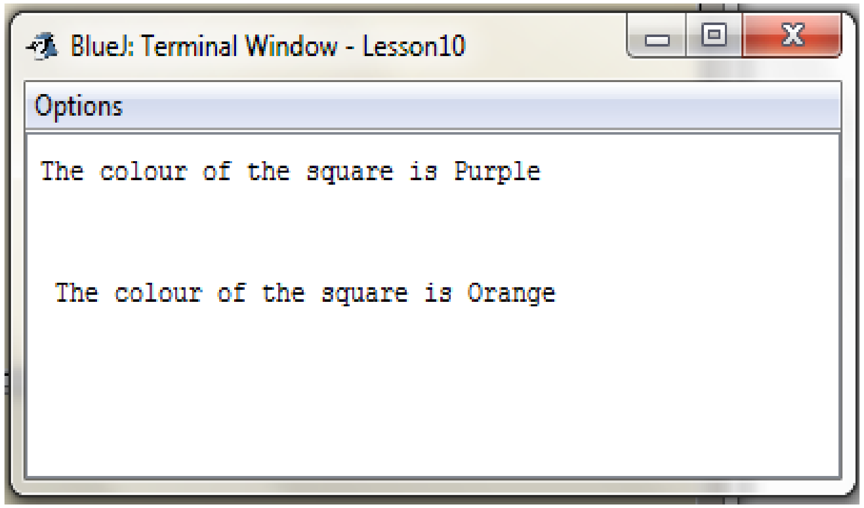


```
public class Square extends TwoDShape
{
    private double length; // the length attribute of a square
    /**
     * Constructor for objects of class Square
     */
    public Square(double num)
    {
        length = num; // set length to the passed value of num
    }
    /**
     * Square constructor :
     * @param len the length of the side of the square
     */
    public Square(double len, String colourValue) //Notice there is NO return type for a class constructor.
    {
        super(colourValue); // use the constructor in the parent (super class)
        length = len; // set the class attribute (variable) radius equal to len
    }
    /**
     * setter method for the class instance variable length
     * @param  length   the value to set the class instance variable length to.
     */
    public void setLength(double num)
    {
       length = num ; // set the class instance variable to the value in the formal parameter num.
    }
    /**
     * getter method for the class instance variable length
     * @return the value of the class instance variable length
     */
    public double getLength()
    {
        return length; // return the value in the class instance variable length
    }
}
```

GeneralShapeTester; creates an object of Square type.



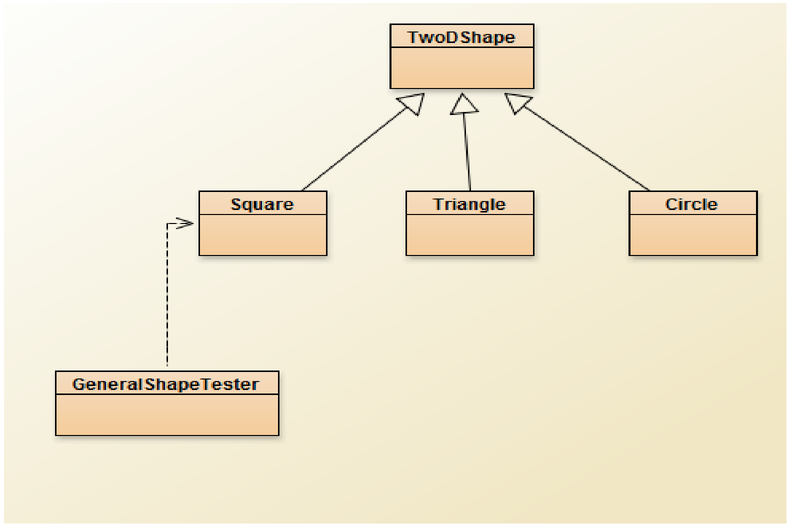
```
public class GeneralShapeTester
{

    public static void main(String [] args)
    {
        // create a square with a length 5 and colour "Purple" (the square constructor here uses the superclass constructor)
        Square mine = new Square(5, "Purple");

        // using the inherited method getColour defined in TwoDShape
        System.out.println("The colour of the square is " + mine.getColour());

        // using the inherited method setColour defined in TwoDShape
        mine.setColour("Orange");

        // examining the new colour of the square object using the inherited getColour method defined in class TwoDShape
        System.out.println("\n\n The colour of the square is " + mine.getColour());

    }

}
```



BlueJ: Terminal Window - Lesson10

Options

```
The colour of the square is Purple


 The colour of the square is Orange
```



Conclusion

The basic principles of object oriented design were presented and illustrated with the BasicCircle and Circle examples.

We looked at how to create an array of objects building on our previous knowledge of arrays (declaring an array reference variable and creating an array linked to this reference variable)

We had a brief introduction to object oriented inheritance. There are many other object oriented concepts and subtleties which we will describe in the coming weeks and we will explore and use them for the remainder of the course.