CS620c
Introducing Objected Oriented Inheritance.

Joe Duffin

Iontas Building Room 1.45

Email: Joseph.Duffin@nuim.ie

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
{
        ….
        ….
}
```

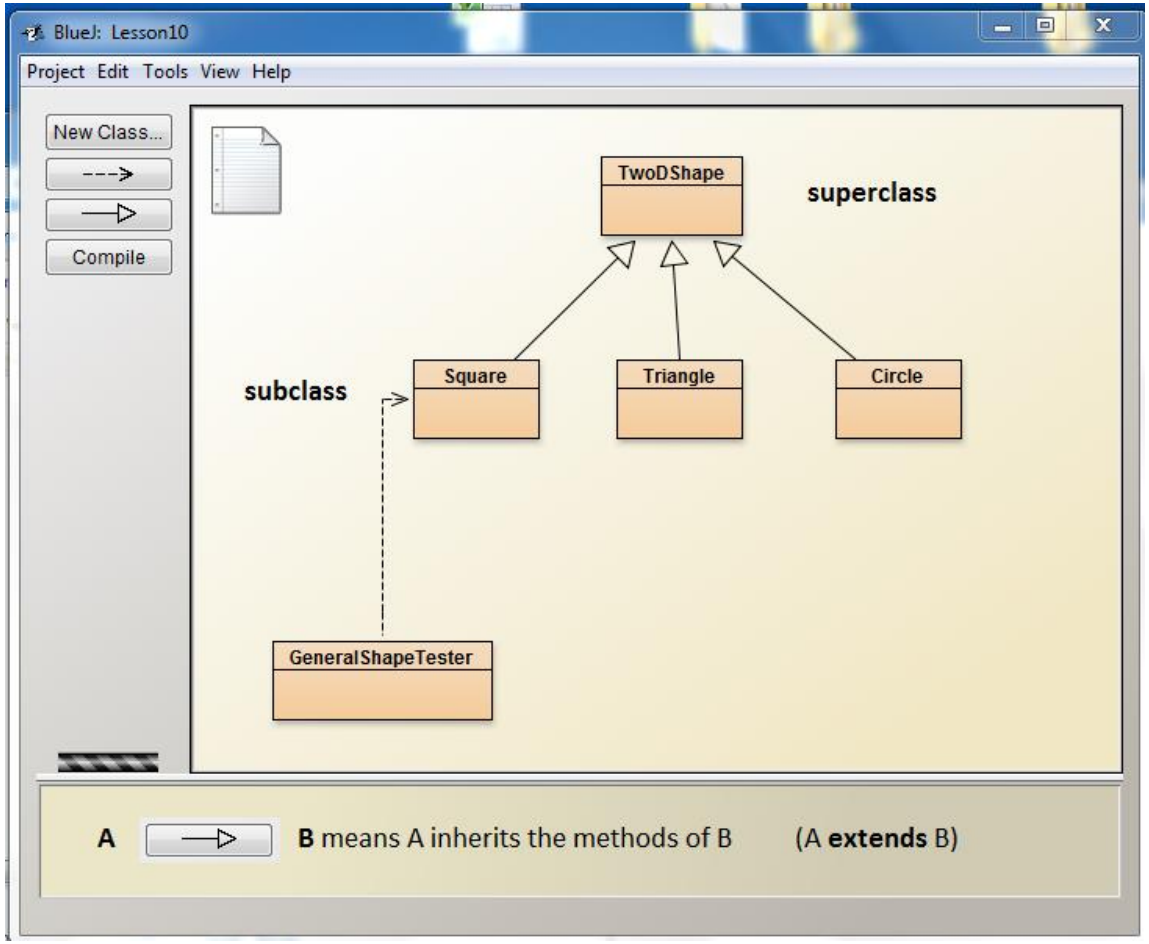
The class **Square** can **inherit** data and behaviour from the class **TwoDShape**

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



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 can be viewed as a 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; the superclass aka the parent class

```java
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
    }
}
```

The Square Class **inherits** from the TwoDShape Class

```java
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
    }
}
```

`super( );`

The java word **super( )** is stands for the constructor of the parent class in an inheritance relationship.

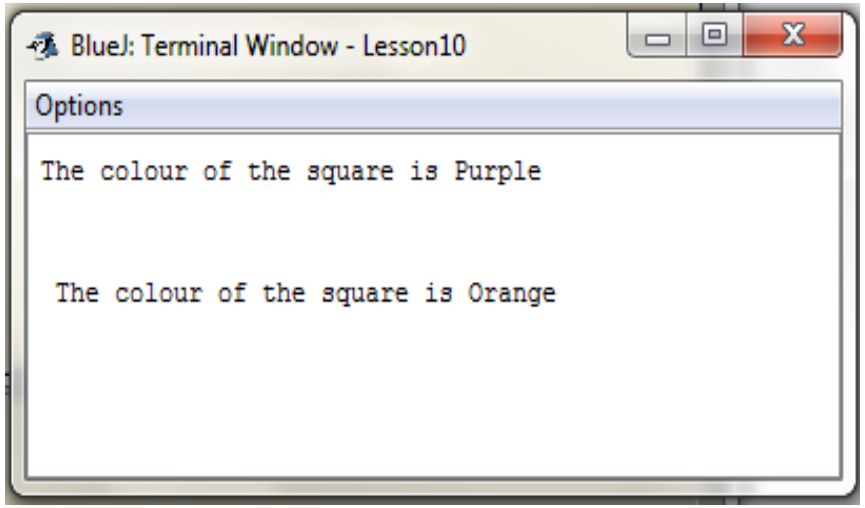
super( ) on its own invokes the default constructor.

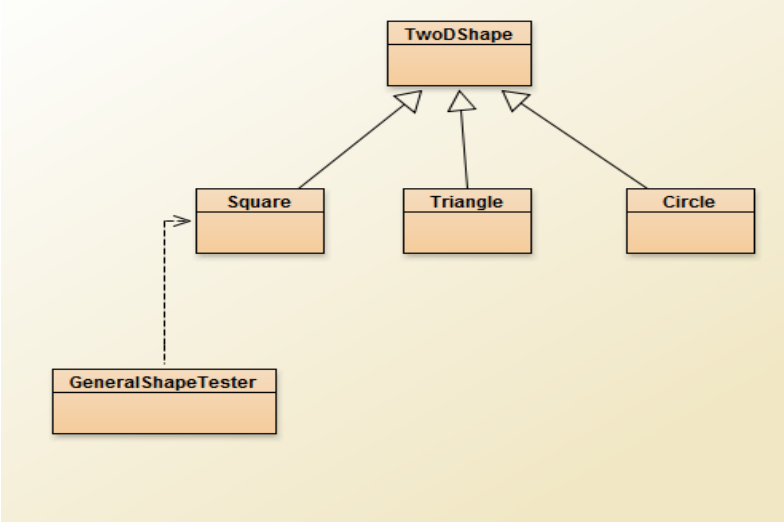
super (parameter1, parameter2) calls the parent constructor with two parameters.

It depends on the design of the constructors available in the parent class (the class inherited from)

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());

    }

}
```

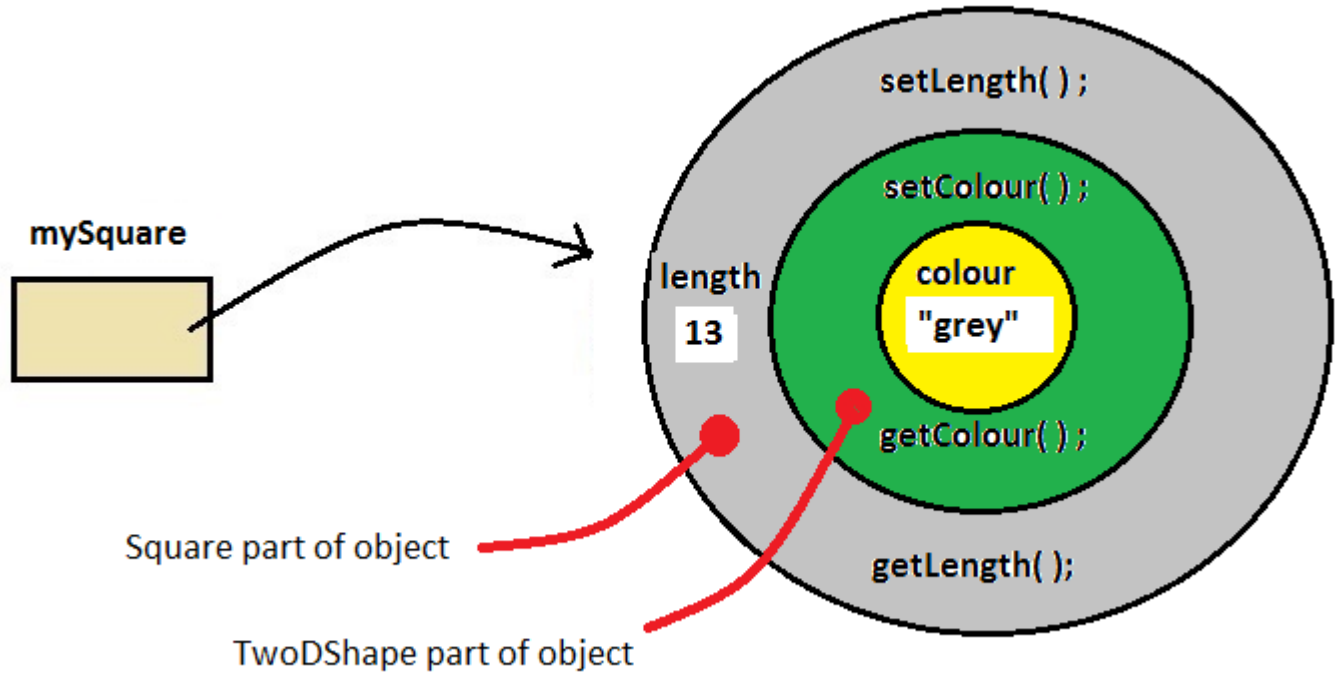




What happens when you create an instance of the subclass Square? What if we use the constructor below?

```
public Square(double num)
{
 // The compiler adds right here, an implicit call to the
 // parent's class default constructor TwoDShape( ) which
 // does not take a parameter and sets the colour attribute
 // to "grey" ( Examine the constructor code to see this)
 // So it actually creates a TwoDShape object part first
 // When that object part is created this Square constructor
 // is used to create the rest of the object
 // That is why the memory set aside for the object
 // mySquare looks like an "onion". It has an inner TwoDShape
 // part which is created first and an outer Square part
 // which was created second. See diagram on the next page.

  length = num; // set length attribute to the value of num
}

     // created using constructor above
     Square mySquare = new Square(13);
```

```
Square mySquare = new Square(13);
```



An object of type Square with its **TwoDShape** part and its **Square** part