

Java - User Defined Classes

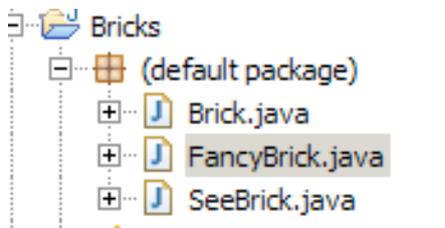
The time has come to create a Class from scratch rather than using instances of Classes like Button or Image that are defined within Java. The Class is often written as a separate Java file within the project folder. Like other all other Classes (except static Classes) they are referenced by creating objects from the Class by the keyword new. When an object is created it will show up in the IDE with the same drop downs for methods as the Classes that are provided by Java.

Brick

This Class is based on the g.drawRect() method that draws a simple rectangle on the screen. It will have various public methods that allow the rectangle object to be manipulated. The position, size and colour of the rectangle are private to that object although the public methods are quite likely to manipulate these private variables.

Brick
Length, width, xCoord, yCoord, bColor
drawBrick moveAlong moveUp setCol

SeeBrick
myBrick
init paint actionPerformed



This program will have the 2 Classes described above. There is a chance that Brick might get used somewhere else so it is a separate Class file with its own code. The interface here is a way to test the Brick Class. It exists to pass instructions from the outside user to the Brick object and to display the results. In this case there is also an html file to display the lot on a

web page.

When writing programs that have a user interface Class and 1 or more Classes that do things it is useful to write some of the doing Class then some of the interface Class. This enables the key doing Class to be tested as it is constructed. The interface class will pick up the public methods of the doing Class in the IDE as objects from that Class are referenced with the dot operator. The absence of expected methods in the object drop down is a clear sign to send in the marines.

Start by coding some of the Brick Class, note the use of public and private.

- Private – only exists within the Class
- Public – all Classes of the same package can use these methods or variables

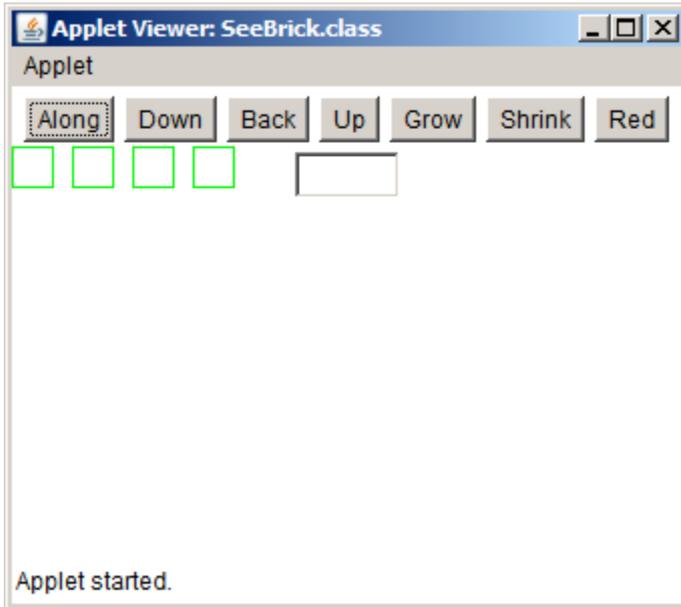
```

import java.awt.*; // we need to use g.drawRect so awt is imported
//the usual applet type imports can be imported by the interface Class

public class Brick //Capital letter for Class
{
    private int length,width,xCoord,yCoord; //these variables are only accessed by
Brick
    private Color bColour = Color.green; //they are private to Brick
    public Brick()
    { //this is the constructor of Brick – it sets default variables when a Brick is
created
        length=20;
        width = 10;
        xCoord =10;
        yCoord =10;
    }
    public Brick(int len,int wid, int x, int y)
    { // yes you can have more than 1 constructor
    // this is a public method to set dimensions from another Class
        length=len;
        width = wid;
        xCoord = x;
        yCoord = y;
    }
    public void drawBrick(Graphics g)
    { //draw a Brick on the user's screen
    //this 'returns' xCoord and yCoord to the user
        g.setColor(bColour);
        g.drawRect(xCoord,yCoord,length,length);
    }
    public void moveAlong(int dist)
    { //increase the private variable xCoord
        xCoord=xCoord + dist;
    }
    public void moveUp(int dist)
    { //increase the private variable yCoord
        yCoord = yCoord + dist;
    }
    public Color setCol(Color c)
    { //get a colour and return bColour to the Class that called this public method
        bColour=c;
        return bColour;
    }
}

```

All the public methods can be accessed from another Class if an object of the Class brick is created. The dot operator will show up the public methods.



The Brick Class cannot be tested without an appropriate interface. In this case various buttons will handle moving the Brick object.

Methods for Grow and Shrink can be added to Brick later.

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*; //for addActionListener

public class SeeBrick extends Applet implements ActionListener
{ //note implements to make it work
    private Button along, down, back, up, red, grow, shrink;
    private Brick myBrick; //declare the Brick like a Button object

    public void init()
    { //set up buttons
        along = new Button("Along");
        add(along);
        along.addActionListener(this);
        //add all the other Buttons in the same way
        //add the Brick object
        myBrick = new Brick();
    }
    public void actionPerformed(ActionEvent event)
    {
        if (event.getSource()==along)
            myBrick.moveAlong(10);
        //add other actions
        //repaint page with changes
        repaint();
    }
    public void paint(Graphics g)
    { //show brick on the applet
        myBrick.drawBrick(g);
    }
}
```

//In the above code the Brick can be moved down or back by sending negative numbers to the relevant methods in Brick.

Modifications

1. A grow method should be implemented in Brick to change the size of the Brick object. This is governed by the private variables length and width. Variables will have to be passed to the Brick object from the interface Class to say how much to increase or decrease the length and width variables by.
2. Only 1 Brick has been used. There is no limit to the number of objects created from a Class within another Class. Consider the number of Button objects that might be used. Create another Brick object, it will need its own name and have to be added separately with new. To prevent both Bricks being drawn in the same place have the 2nd brick be created using the constructor method that requires a position and size to be entered.

```
public Brick(int len,int wid, int x, int y) //in Brick
```

```
myBrick = new Brick(); //in seeBrick  
hisBrick = new Brick(30,30,30,30);
```

The paint() method will also have to be modified to draw both Brick objects.

```
public void paint(Graphics g)  
{//show brick  
    myBrick.drawBrick(g);  
    hisBrick.drawBrick(g);  
}
```

3. With more than 1 object derived from the same Class some method is required to decide which object is to be manipulated at any time. Luckily objects can be part of an array making it possible to reference 1 of an array of objects through a loop.

Creating an array of user defined objects is similar but not exactly the same as creating an array of control objects. The array has to be declared and a loop used to create instances of each object in the array with new.

```
public class SeeBrick extends Applet implements ActionListener  
{  
    private Button along, down, back, up, red, grow,shrink;  
    private Brick myBrick []= new Brick[4];  
    public void init()  
    {  
        //button code as before  
        for(int x=0;x<4;x++)  
        {  
            //this uses the value of x to spread the bricks out on the applet  
            myBrick[x]=new Brick(20,20,x*30,30);  
        }  
    }  
}
```

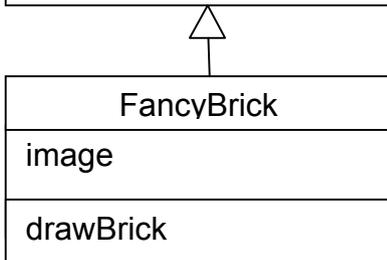
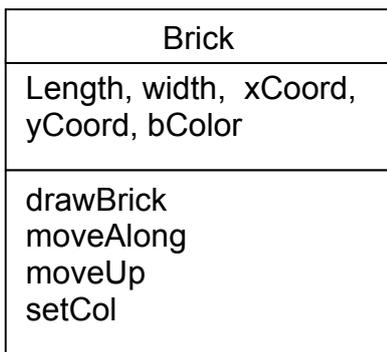
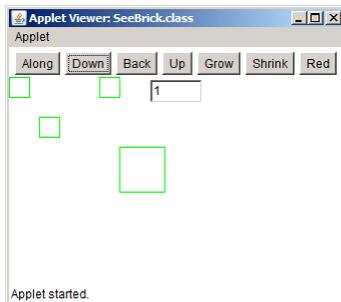
```
}
```

To reference each brick the array position must be used. To check the code so far works myBrick[0] can be referenced and the other 3 Bricks left alone.

```
if (event.getSource()==along)
    myBrick[0].moveAlong(10);
```

4. To choose which brick will be referenced by the interface at any time its array position can be used. This could be through input from a CheckboxGroup or by typing into a TextField. Any data entered into the TextField would have to be validated to prevent an array out of bounds error.

```
try
{ //iPos is an integer used to record the array position of myBrick[]
  iPos=Integer.parseInt(position.getText());
  if((iPos<0)||iPos>4) //data outside array boundary
    iPos=0;
}
catch(NumberFormatException e)
{ //not numeric
  iPos=0;
}
```



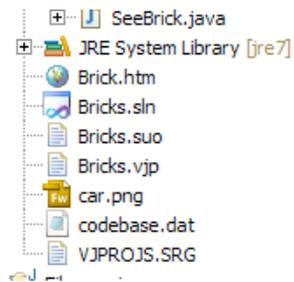
Loading images

The Brick draws a rectangle. If the method drawBrick() is changed other shapes or images can be drawn.

```

//imports also needed
public class FancyBrick extends Brick {
    private Image img = null;
    public FancyBrick()
    {
        super();
        img = Toolkit.getDefaultToolkit().getImage("car.png");
    }
    public FancyBrick(int len,int wid, int x, int y)
    {
        super();
        img = Toolkit.getDefaultToolkit().getImage("car.png");
    }
}
public void drawBrick(Graphics g)
{//should also use length and width for grow and shrink
    g.drawImage(img, xCoord, yCoord, null);
    //g.drawImage(img,xCoord, yCoord,length, width, null);
}
}

```



The image (car.png) must be in the same location as the Java files;

