Java - User Defined Classes

C&G criteria: 1.3.1, 1.4.1

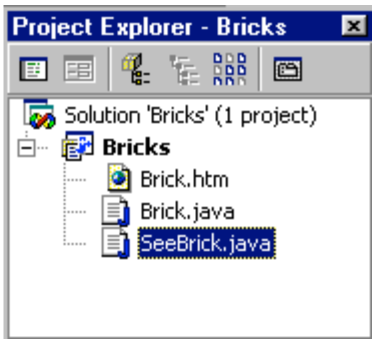
The time has come to create a Class from scratch rather than using instances of Classes like Button or Image that are defined within Java. The Class is often written as a separate Java file within the project folder. Like other all other Classes (except static Classes) they are referenced by creating 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



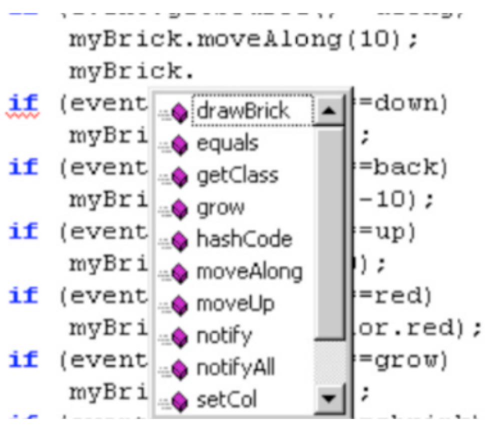
This program will have the 2 Classes described above. There is a chance that Brick might get used somewhere else so it is a separate Class file with its own code. The interface here is a way to test the Brick Class. It exists to pass instructions from the outside user to the Brick object and to display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

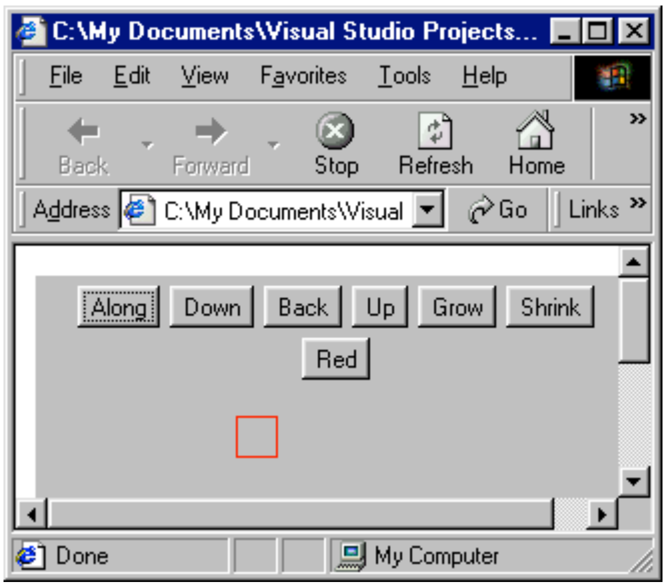
```
import java.awt.*; // we need to use g.drawRect so awt is imported
//the usual applet type imports can be imported by the interface Class

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;
        }
}
```



```
    myBrick.moveAlong(10);
    myBrick.
if (event           =down)
    myBri           ;
if (event           =back)
    myBri           -10);
if (event           =up)
    myBri           );
if (event           =red)
    myBri           or.red);
if (event           =grow)
    myBri           ;
```

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.


C:\My Documents\Visual Studio Projects...
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address C:\My Documents\Visual Go Links
Along Down Back Up Grow Shrink
Red
Done
My Computer

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.

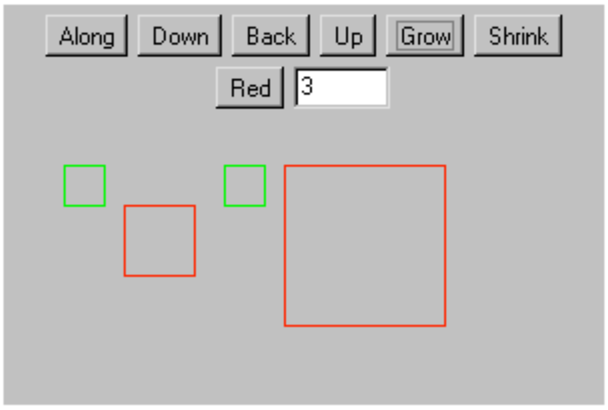
```
public class SeeBrick extends Applet implements ActionListener
{
        private Button along, down, back, up, red, grow,shrink;
        private Brick myBrick []= new Brick[4];
        public void init()
        {//button code as before
                for(int x=0;x<4;x++)
                {//this 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;
}
```



Keyboard Class

The Input and Output work showed the problems with capturing data from the keyboard when running Java as a console application. The code used to overcome these problems was organised into methods within a single Class. As the same problems will occur whenever keyboard input is required in a console application the key methods can be copied to a separate Keyboard Class. An object created from this Class can be used as required. The methods of that Class need never be fiddled with again. This is a good example of a Class that has real use in many applications.

Keyboard
getString getChar getInt

This Class has no variables outside of its methods.

The methods within Keyboard are of this form

```
import java.io.*;
public class Keyboard
{

        public String getString()
        {//80 byte buffer array for input
                byte buffer[]=new byte[80];
                String temp=null;
                try
                {
                                System.in.read(buffer);
                        temp=new String(buffer);//create String from buffer
                        temp=temp.trim();
                }
                catch(IOException e)
                {
                        System.out.println(e);
                }
                return temp;
        }
}
```

A new Class will be required to test out the Keyboard Class.

```
public class testKeyboard
{
        public static void main(String args[])
        throws java.io.IOException
        {//declare new Keyboard object
                Keyboard myKeyboard = new Keyboard();
                System.out.print("Enter a String ");
                String myString = myKeyboard.getString();
        //capture other data types
}
```