Contents

0 Introduction **1**

1 Building the View for Tricolor **2**

2 Our Panels Need to Know their Colors **2**

3 Inserting ColorPanels into the Tricolor App **5**

4 Le Carte **6**

5 It's time to build the controller! **10**

6 The Color Menu and the Controller **10**

7 The Position Menu and Its Controller **17**

8 All Code Shown **19**

8.1 Tricolor.java 19

8.2 ColorPanel.java 21

8.3 QuitListener.java 21

8.4 ColorMenuItem.java 22

8.5 ColorMenuItemListener.java 22

8.6 PositionMenuItem.java 23

8.7 PositionMenuItemListener.java 23

9 See it All **25**

0 Introduction

This chapter consists of a case study. We are going to get a fully functioning application which will draw in its content pane called `Tricolor`. It will be menu-driven and fully graphical.

1 Building the View for Tricolor

The `Tricolor` application will feature three menus: File, Color and Position. The File menu will have one item, Quit, which will quit the application. The color menu will have three colors: red, green and blue. The Position menu will have three positions, left middle and right.

The content pane will contain three panels, a left, middle and right panel. When a panel is selected, the panel will be painted the color of the item in the Color menu. All panels will be white when the app starts.

We begin by creating a graphical shell as follows.

```
import javax.swing.JFrame;

public class Tricolor extends JFrame implements Runnable
{
    public Tricolor()
    {
        super("Tricolor");
    }
    public void run()
    {
        setSize(500,500);
        setVisible(true);
    }
    public static void main(String[] args)
    {
        Tricolor t = new Tricolor();
        javax.swing.SwingUtilities.invokeLater(t);
    }
}
```

If you compile and run this you should see an empty window with a title in the title bar as shown in figure 0.

2 Our Panels Need to Know their Colors

Now it is time to make some design decisions. We need three panels that know their colors. To this end, we create a new class called `ColorPanel`, which extends `JPanel` and which knows its color. It will paint itself its color.

From this point, we know that we will need to change the color of these panels. Since graphical elements tend to be big, we favor mutability here and allow changes of state. We will allow this panel to change its color.

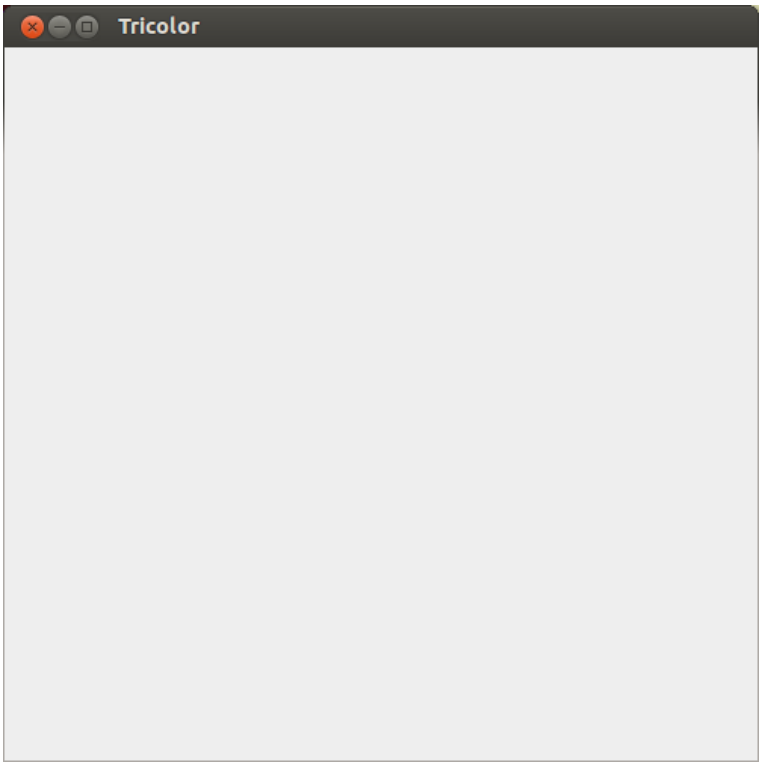

Figure 0: Empty Tricolor Window

```
import javax.swing.JPanel;
public class ColorPanel extends JPanel
{
    private Color color;
}
```

Let us now make this panel white in the constructor. Also, let us implement a method `setColor` that sets the color of this panel. The color of this panel will be controlled by the items in the Color menu. As a result, we know now that it must be mutable. We insert the method `setColor` so the panel's color can change in response to the selection of a menu item.

```
import javax.swing.JPanel;
import java.awt.Color;
import java.awt.Graphics;
public class ColorPanel extends JPanel
{
    private Color color;
    public ColorPanel()
    {
        color = Color.WHITE;
    }
    public void setColor(Color c)
    {
        color = c;
    }
}
```

Finally, we will tell the panel to paint itself its color. While we are here, let's create a `toString()` method in case we want to check anything along the way. Note the use of the `@Override` annotation to get the compiler to check that we are overriding correctly.

```
import javax.swing.JPanel;
import java.awt.Color;
import java.awt.Graphics;
public class ColorPanel extends JPanel
{
    private Color color;
    public ColorPanel()
    {
        color = Color.WHITE;
    }
    public void setColor(Color c)
    {
```

```
        color = c;
    }
    @Override
    public void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        g.setColor(color);
        g.fillRect(0,0,getWidth(), getHeight());
    }
    @Override
    public String toString()
    {
        return "I am a color panel of color" + color;
    }
}
```

3 Inserting ColorPanels into the Tricolor App

The `Tricolor` app will now have four state variables. One will be for each of the three panels. The fourth will be for a variable that points at the current panel. By default, and before the Position menu is created, we will make the current panel point at the left panel.

```
import javax.swing.JFrame;
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import java.awt.Color;
import java.awt.Container;
import java.awt.GridLayout;

public class Tricolor extends JFrame implements Runnable
{
    private ColorPanel leftPanel;
    private ColorPanel middlePanel;
    private ColorPanel rightPanel;
    private ColorPanel currentPanel;

    public Tricolor()
    {
        super("Tricolor");
        leftPanel = new ColorPanel();
        middlePanel = new ColorPanel();
        rightPanel = new ColorPanel();
```

```
        currentPanel = leftPanel;
    }
    public void run()
    {
        setSize(500,500);
        Container c = getContentPane();
        c.setLayout(new GridLayout(1,3));
        c.add(leftPanel);
        c.add(middlePanel);
        c.add(rightPanel);
        setVisible(true);
    }
    public static void main(String[] args)
    {
        Tricolor t = new Tricolor();
        javax.swing.SwingUtilities.invokeLater(t);
    }
}
```

If you have doubt as to whether the `ColorPanel`s got inserted, just go into the `ColorPanel` constructor and temporarily set it to be green. Then it will be obvious. The `Tricolor` app now knows four panels: the three in the content pane, plus the one to be colored. We show the results of compilation in Figure 1.

4 Le Carte

It's time to begin making the menus. We will produce the menu bar and the menus. This is akin to carpentry. We must make and nail in the menu bar, a container in which the menus live. Then we install the menus. So let us begin. We will create a method called `makeMenus` to keep `run` from becoming a run-on method. So, add the call `makeMenus()` to your `run` method. Your `run` method should look like this. As long as you place this before `setVisible`, it does not matter greatly where you put it.

```
    public void run()
    {
        setSize(500,500);
        makeMenus();
        Container c = getContentPane();
        c.setLayout(new GridLayout(1,3));
        c.add(leftPanel);
        c.add(middlePanel);
        c.add(rightPanel);
```

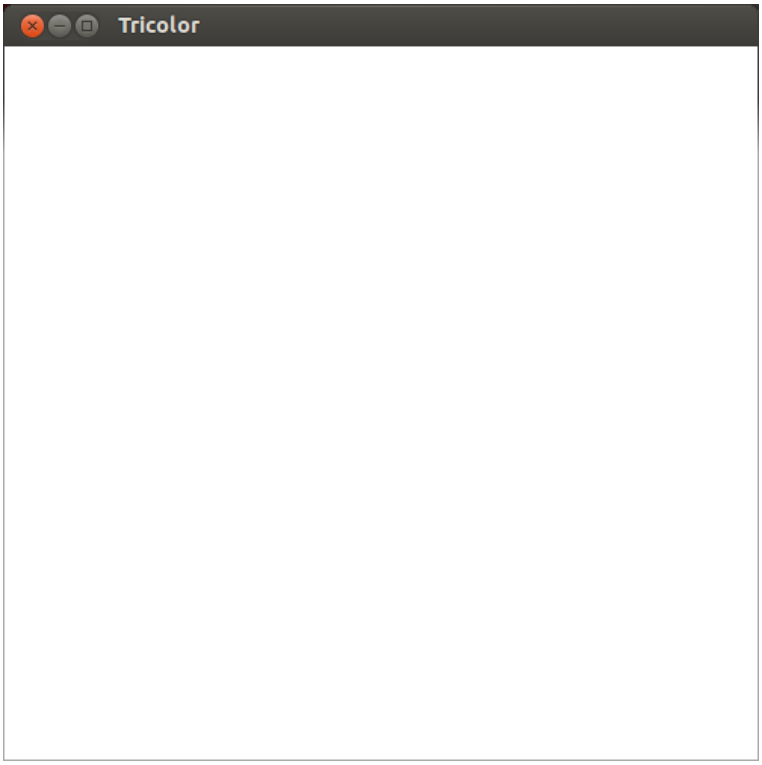

Figure 1: Tricolor window with the ColorPanels added. Note that the content pane is now white.

```
        setVisible(true);
    }
```

Now we will make the menu bar and nail it in. Place this code in `makeMenus`.

```
    JMenuBar mbar = new JMenuBar();
    setJMenuBar(mbar);
```

The `JMenuBar` is a container that holds widgets. In particular it holds `JMenus`. We make these and add them using ... duh ... `add`.

```
    private void makeMenus()
    {
        JMenuBar mbar = new JMenuBar();
        setJMenuBar(mbar);
        JMenu fileMenu = new JMenu("File");
        mbar.add(fileMenu);
        JMenu colorMenu = new JMenu("Color");
        mbar.add(colorMenu);
        JMenu positionMenu = new JMenu("Position");
        mbar.add(positionMenu);
    }
```

Run this. You will see menu headers living in the menu bar. Before we begin generating menu items, we need to *think*. You will now see an app with three menus and its three panels put into place. It is shown in Figure 2.

Now we begin with the Color menu. We will create menu items that are smart enough to *know their colors*. Be warned that this class will undergo an evolution as we proceed.

When designing a class ask: What does the class need to know? This tells us what its state must be. Then ask: what does the class need to do? This tells us what methods we need to write.

Note we created a `ColorPanel` that knows its color. We will now create a `ColorMenuItem` that knows its color. Since this color will be permanently assigned to this menu item, we will mark it `final`.

Note that because there is no canonical association between colors and their names, our constructor must allow us to pass both the color and its name. Since we can pass the color name off to the parent constructor, we never bother to store it as a state variable. We can do this if the need arises. But it will never do so in this example.

```
import javax.swing.JMenuItem;
import java.awt.Color;
```

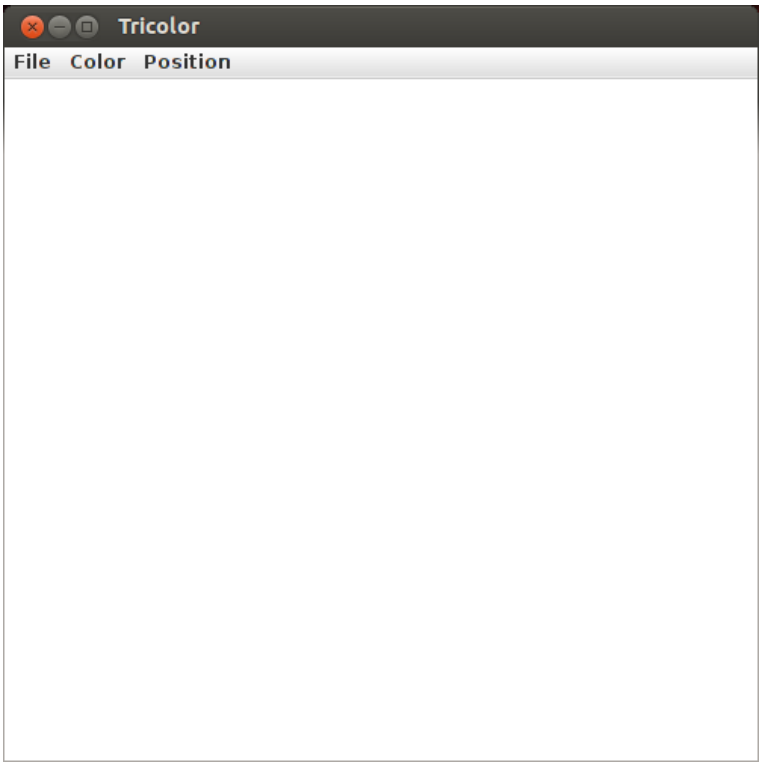

Figure 2: The Tricolor window is now festooned with menus

```
public class ColorMenuItem extends JMenuItem
{
    private final Color color;
    public ColorMenuItem(Color _color, String colorName)
    {
        super(colorName);
        color = _color;
    }
}
```

5 It's time to build the controller!

We begin with the low-hanging fruit. Our sole item in the File menu is a quit item. So let us make that. Just add these two lines into the `makeMenus()` method just after you create and add the File menu.

```
        JMenuItem quitItem = new JMenuItem("Quit");
        fileMenu.add(quitItem);
```

To make this live we will need an action listener that shuts the app down when its `actionPerformed` code gets called. We will call this class `QuitListener`. In the file `QuitListener.java`, place the following code.

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class QuitListener implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        System.exit(0);
    }
}
```

This small class is all we need. Now we attach an instance of it to `quitItem` as follows.

```
         quitItem.addActionListener(new QuitListener());
```

Insert this line of code in `makeMenus()` just after you add the `quitItem` to the File menu. Now run the code. Pull down the `File` menu and choose `Quit`. The application will quit! We have the first element of the controller. Figure 3 shows the quit menu item being selected.

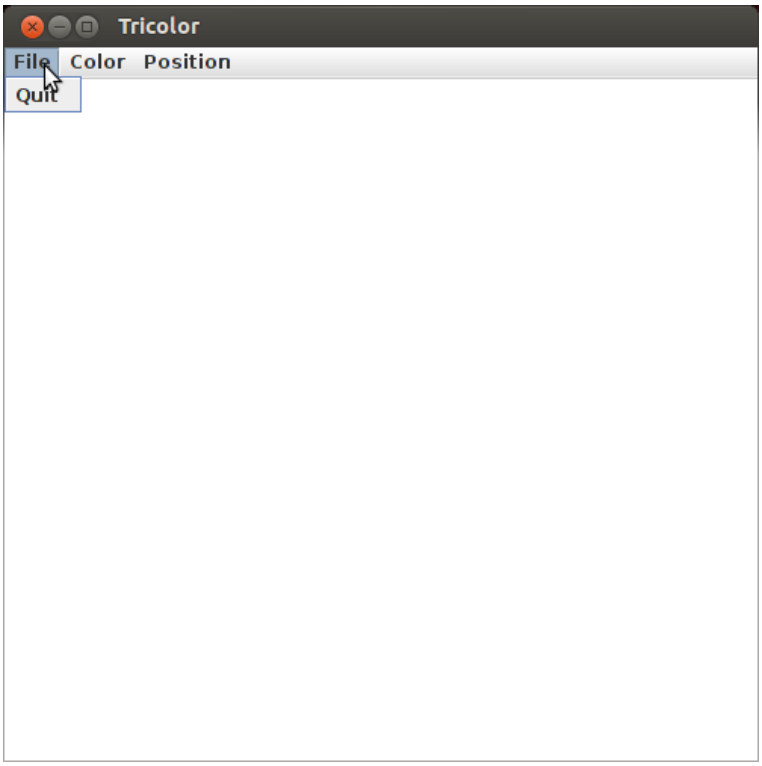

Figure 3: Tricolor window with quit menu item being selected.

6 The Color Menu and the Controller

Next we will take care of the Color menu. You might think we are going to do this.

1. Create the three menu items.
2. Create a listener class
3. If the red menu item is chosen, turn the panel red.
4. If the green menu item is chosen, turn the panel green.
5. If the blue menu item is chosen, turn the panel blue.

But that is *not* going to happen. Every time we want to add a new color, we are forced to do a lot of work. Another way to proceed might be to create separate listeners for each item. Then, when that item is chosen, that item's color is used. All of the listeners would look alike. This is a violation of the 11th commandment: *Thou shalt not maintain duplicate code!* What, then, is the best course of action?

Remember: *The power to delegate is the power to accomplish!* We will make the menu items responsible for knowing their colors and for having a listener that causes their colors to be used.

Previously we created `ColorMenuItem.java`.

```
import javax.swing.JMenuItem;
import java.awt.Color;

public class ColorMenuItem extends JMenuItem
{
    private final Color color;
    public ColorMenuItem(Color _color, String colorName)
    {
        super(colorName);
        color = _color;
    }
}
```

We can populate the Color menu with items that are instances of this class. Add these lines to `makeMenus` after you add the Color menu.

```
    colormenu.add(new ColorMenuItem(Color.RED, "red"));
    colormenu.add(new ColorMenuItem(Color.GREEN, "green"));
    colormenu.add(new ColorMenuItem(Color.BLUE, "blue"));
```

Once you do this, run your program and see that the menu items are now present. Note that their controllers are not yet written, so they do not yet work.

Next, we must begin to write a controller for a color menu item. Begin by creating this new class, `ColorMenuItemListener.java`. Our color menu items must know their colors, so we included that as a state variable. We also include a constructor to initialize the state variable. So our class compiles we put in the `actionPerformed` method required by the `ActionListener` interface. We put in some code that prints to `stdout` so we can test if the listener is properly attached when the time comes.

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.Color;

public class ColorMenuItemListener implements ActionListener
{
    private final Color color;
    public ColorMenuItemListener(Color _color)
    {
        color = _color;
    }
    public void actionPerformed(ActionEvent e)
    {
        //tell the current panel to change color
        System.out.println("fooment");
    }
}
```

Now that we have a listener, we can go back to our `ColorMenuItem` class and attach a listener as follows in the last line of the constructor.

```
import javax.swing.JMenuItem;
import java.awt.Color;

public class ColorMenuItem extends JMenuItem
{
    private final Color color;
    public ColorMenuItem(Color _color, String _colorName)
    {
        super(_colorName);
        color = _color;
        addActionListener(new ColorMenuItemListener(color));
    }
}
```

Now compile all and run. When you select a `ColorMenuItem`, you will see `"fooment"` put to `stdout`. But life is not that simple.

Next, we need to think about the code needed to make the listener work. We must think about the lines of communication necessary to accomplish this. The listener must know the current panel and tell it to change its color. There is a problem: `ColorMenuItemListener` has no way to communicate back to the application object, `Tricolor`. The `ColorMenuItem` is the intermediary: it needs to know of `Tricolor` so it can pass it on to its listener.

How do we set that up? We add a state variable that is a `Tricolor`. We will make this `final` since there will be only one `Tricolor` as the program runs. So Let us modify the menu item class first. We insert a new state variable and initialize it in the constructor.

```
import javax.swing.JMenuItem;
import java.awt.Color;

public class ColorMenuItem extends JMenuItem
{
    private final Color color;
    private final Tricolor tc;
    public ColorMenuItem(Color _color, String _colorName, Tricolor _tc)
    {
        super(_colorName);
        color = _color;
        tc = _tc;
        addActionListener(new ColorMenuItemListener(tc));
    }
}
```

Compile and see brokenness. Look in `makeMenus()` to find the offending code. We have changed the constructor for the `ColorMenuItem` and must therefore fix the constructor calls in the calling routine `makeMenus()` in `Tricolor`. Here is how they currently look

```
    colormenu.add(new ColorMenuItem(Color.RED, "red"));
    colormenu.add(new ColorMenuItem(Color.GREEN, "green"));
    colormenu.add(new ColorMenuItem(Color.BLUE, "blue"));
```

Modify them to this. You are now passing an instance of the `Tricolor` class to each of them.

```
    colormenu.add(new ColorMenuItem(Color.RED, "red", this));
    colormenu.add(new ColorMenuItem(Color.GREEN, "green", this));
    colormenu.add(new ColorMenuItem(Color.BLUE, "blue", this));
```

We are not done yet. Now we must modify the constructor of the `ColorMenuItemListener` to accept a reference to a `Tricolor` as follows.

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.Color;

public class ColorMenuItemListener implements ActionListener
{
    private final Tricolor tc;
    private final Color color;
    public ColorMenuItemListener(Tricolor _tc, Color _color)
    {
        tc  = _tc;
        color = _color;
    }
    public void actionPerformed(ActionEvent e)
    {
        //tell the current panel to change color
        //System.out.println("fooment");
    }
}
```

and then change the constructor calls to this class back in `ColorMenuItem`. We just modify the line where we add the action listener.

```
addActionListener(new ColorMenuItemListener(tc, color));
```

We need in the `actionPeformed` method to be able to get the current panel and change its color. We need the following accessor in `Tricolor`.

```
    public ColorPanel getCurrentPanel()
    {
        return currentPanel;
    }
```

To tell the current panel to change to our color, we use

```
tc.getCurrrentPanel().setColor(color);
```

Place this code in the listener to see the following

```
    tc.getCurrentPanel().setColor(color);
    tc.repaint();
```

We use the `repaint()` to refresh the window and have our change take effect. Use a call to `repaint()` whenever you want the graphics to update.

We now show the finished appearance of the two classes. First `ColorMenuItem.java` is shown here.

```
import javax.swing.JMenuItem;
import java.awt.Color;

public class ColorMenuItem extends JMenuItem
{
    private final Color color;
    private final Tricolor tc;
    public ColorMenuItem(Color _color, String _colorName, Tricolor _tc)
    {
        super(_colorName);
        color = _color;
        tc = _tc;
        addActionListener(new ColorMenuItemListener(tc, color));
    }
}
```

Now here is the listener class.

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.Color;

public class ColorMenuItemListener implements ActionListener
{
    private final Tricolor tc;
    private final Color color;
    public ColorMenuItemListener(Tricolor _tc, Color _color)
    {
        tc  = _tc;
        color = _color;
    }
    public void actionPerformed(ActionEvent e)
    {
        //Tell the current panel to change
        //color and refresh the whole thing.
        tc.getCurrentPanel().setColor(color);
        tc.repaint();
    }
}
```

Now run this. You will see that the Color menu now functions.

Programming Exercises

1. Add a new menu item for the color white. How much code do you have to add?
2. Add a new menu item called `tarheel` which uses color `0xabcdef`.

7 The Position Menu and Its Controller

We now go to work on the Position menu. Again we will be subclassing `JMenuItem`. We will make the menu item know two things: which panel belongs to it and we will make it know the application so it can communicate with it.

Let us sketch in `PositionMenuItem.java`.

```
import javax.swing.JMenuItem;

public class PositionMenuItem extends JMenuItem
{
    private final Tricolor tc;
    private final ColorPanel attachedPanel;
}
```

We now create a constructor. We will pass a string along to label the menu item called `pos`.

```
import javax.swing.JMenuItem;

public class PositionMenuItem extends JMenuItem
{
    public final Tricolor tc;
    public final ColorPanel attachedPanel;
    public PositionMenuItem(Tricolor _tc, ColorPanel _attachedPanel,
       String pos)
    {
        super(pos);
        tc = _tc;
        attachedPanel = _attachedPanel;
    }
}
```

Now we create the listener class. This will need to know the application and the attached panel. We comment in a procedure for the action listener to carry out.

```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class PositionMenuItemListener implements ActionListener
{
    private final Tricolor tc;
    private final ColorPanel attachedPanel;
    public PositionMenuItemListener(Tricolor _tc, ColorPanel _attachedPanel)
    {
        tc = _tc;
        attachedPanel = _attachedPanel;
    }
    public void actionPerformed(ActionEvent e)
    {
        //set the current panel to the selected value
    }
}
```

We need the ability to set the current panel. To to this add this accessor method to `Tricolor`

```
public void setCurrentPanel(ColorPanel c)
{
    currentPanel = c;
}
```

```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class PositionMenuItemListener implements ActionListener
{
    private final Tricolor tc;
    private final ColorPanel attachedPanel;
    public PositionMenuItemListener(Tricolor _tc, ColorPanel _attachedPanel)
    {
        tc = _tc;
        attachedPanel = _attachedPanel;
    }
    public void actionPerformed(ActionEvent e)
    {
        //set the current panel to the selected value
        tc.setCurrentPanel(attachedPanel);
    }
}
```

We are now ready to make the menu items. Notice that we do not re-

paint because a change of panel does not cause any graphics to need updating. The panel to be colored changed, and it will change the next time we select a `ColorMenuItem`.

Now let us add our items to the position menu.

```
positionMenu.add(new PositionMenuItem(this, leftPanel, "left"));
positionMenu.add(new PositionMenuItem(this, middlePanel, "middle"));
positionMenu.add(new PositionMenuItem(this, rightPanel, "right"));
```

Now everything is going to work. Below, we see all of the classes in their entirety.

8 All Code Shown

This shows all of the classes.

8.1 Tricolor.java

This is the main application class.

```
import javax.swing.JFrame;
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import java.awt.Color;
import java.awt.Container;
import java.awt.GridLayout;

public class Tricolor extends JFrame implements Runnable
{
    ColorPanel leftPanel;
    ColorPanel rightPanel;
    ColorPanel middlePanel;
    ColorPanel currentPanel;

    public Tricolor()
    {
        super("Tricolor");
        leftPanel = new ColorPanel();
        middlePanel = new ColorPanel();
        rightPanel = new ColorPanel();
        currentPanel = leftPanel;
    }
    public ColorPanel getCurrentPanel()
```

```
{
    return currentPanel;
}
public void setCurrentPanel(ColorPanel c)
{
    currentPanel = c;
}
public void run()
{
    setSize(500,500);
    makeMenus();
    //install Color Panels
    Container c = getContentPane();
    c.setLayout(new GridLayout(1,3));
    c.add(leftPanel);
    c.add(middlePanel);
    c.add(rightPanel);
    setVisible(true);
}
private void makeMenus()
{
    //make and install menu bar
    JMenuBar mbar = new JMenuBar();
    setJMenuBar(mbar);
    //File menu
    JMenu fileMenu = new JMenu("File");
    mbar.add(fileMenu);
    JMenuItem quitItem = new JMenuItem("Quit");
    fileMenu.add(quitItem);
    quitItem.addActionListener(new QuitListener());
    // Color Menu
    JMenu colorMenu = new JMenu("Color");
    mbar.add(colorMenu);
    colorMenu.add(new ColorMenuItem(Color.RED, "red", this));
    colorMenu.add(new ColorMenuItem(Color.GREEN, "green", this));
    colorMenu.add(new ColorMenuItem(Color.BLUE, "blue", this));
    JMenu positionMenu = new JMenu("Position");
    // Position Menu
    mbar.add(positionMenu);
    positionMenu.add(new PositionMenuItem(this, leftPanel, "left"));
    positionMenu.add(new PositionMenuItem(this, middlePanel, "middle"));
    positionMenu.add(new PositionMenuItem(this, rightPanel, "right"));
}
public static void main(String[] args)
{
    Tricolor t = new Tricolor();
```

```
        javax.swing.SwingUtilities.invokeLater(t);
    }
}
```

8.2 ColorPanel.java

This is the class for the three panels that fill the content pane.

```
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;

public class ColorPanel extends JPanel
{
    Color color;
    public ColorPanel()
    {
        super();
        color = Color.WHITE;
    }
    @Override
    public void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        g.setColor(color);
        g.fillRect(0,0,getWidth(), getHeight());
    }
    public void setColor(Color c)
    {
        color = c;
    }
}
```

8.3 QuitListener.java

This is the part of the controller that quits when the quit item is selected from the File menu.

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class QuitListener implements ActionListener
{
    public void actionPerformed(ActionEvent e)
```

```
    {
        System.exit(0);
    }
}
```

8.4 ColorMenuItem.java

This is the class for the menu items in the Color menu.

```
import javax.swing.JMenuItem;
import java.awt.Color;


public class ColorMenuItem extends JMenuItem
{
    private final Color color;
    private final Tricolor tc;
    public ColorMenuItem(Color _color, String _colorName, Tricolor _tc)
    {
        super(_colorName);
        color = _color;
        tc = _tc;
        addActionListener(new ColorMenuItemListener(tc, color));
    }
}
```

8.5 ColorMenuItemListener.java

This is the part of the controller that handles menu selections from the Color menu.

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.Color;

public class ColorMenuItemListener implements ActionListener
{
    private final Tricolor tc;
    private final Color color;
    public ColorMenuItemListener(Tricolor _tc, Color _color)
    {
        tc  = _tc;
        color = _color;
    }
```

```
    public void actionPerformed(ActionEvent e)
    {
        //tell the current panel to change color
        //System.out.println("fooment");
        tc.getCurrentPanel().setColor(color);
        tc.repaint();
    }
}
```

8.6 PositionMenuItem.java

This is the class for menu items determining which panel is to be colored.

```
import javax.swing.JMenuItem;

public class PositionMenuItem extends JMenuItem
{
    public final Tricolor tc;
    public final ColorPanel attachedPanel;
    public PositionMenuItem(Tricolor _tc, ColorPanel _attachedPanel,
        String pos)
    {
        super(pos);
        tc = _tc;
        attachedPanel = _attachedPanel;
        addActionListener(new PositionMenuItemListener(tc, attachedPanel));
    }
}
```

8.7 PositionMenuItemListener.java

This is the controller for the position menu items.

```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class PositionMenuItemListener implements ActionListener
{
    private final Tricolor tc;
    private final ColorPanel attachedPanel;
    public PositionMenuItemListener(Tricolor _tc, ColorPanel _attachedPanel)
    {
        tc = _tc;
        attachedPanel = _attachedPanel;
```

```
    }
    public void actionPerformed(ActionEvent e)
    {
        //set the current panel to the selected value
        tc.setCurrentPanel(attachedPanel);
    }
}
```

Programming Exercises

1. Add a fourth panel to your app. How much code do you need to do this?
2. Modify this code to display four squares in a 2×2 arrangement.

9 See it All

Here we will show screenshots of various things happening.

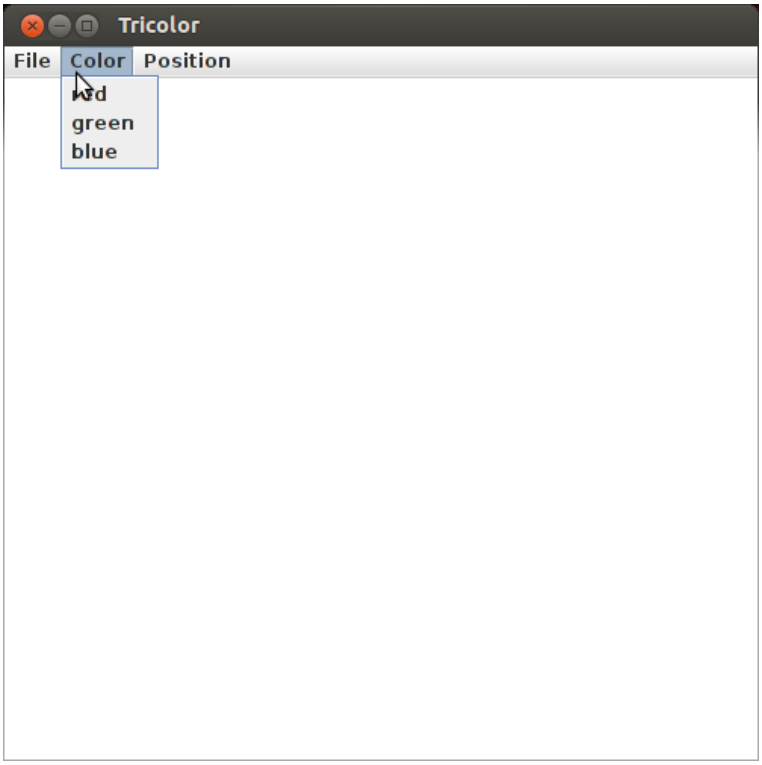

Figure 4: This shows menu items appearing in the color menu.

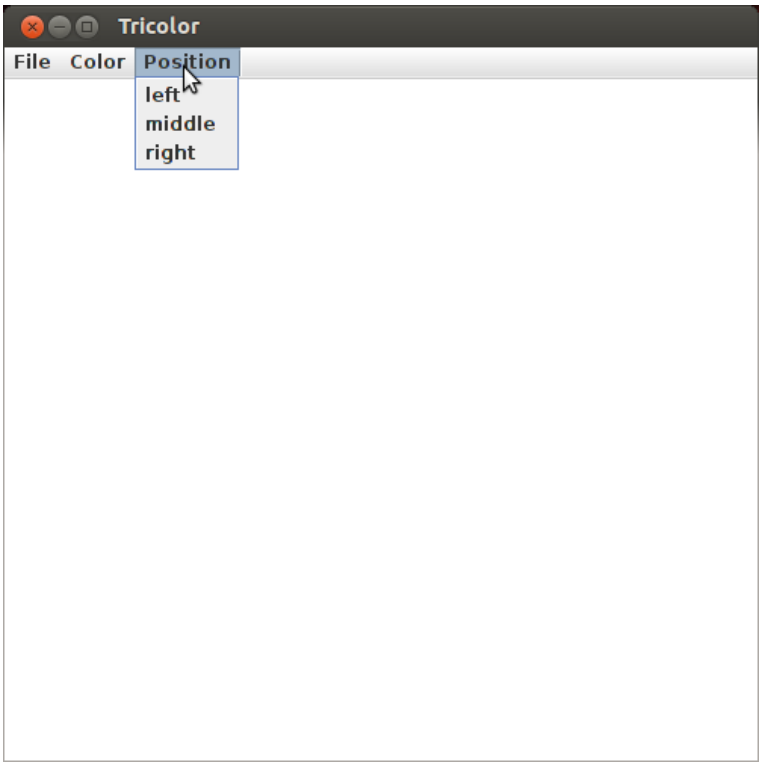

Figure 5: This shows menu items appearing in the position menu.

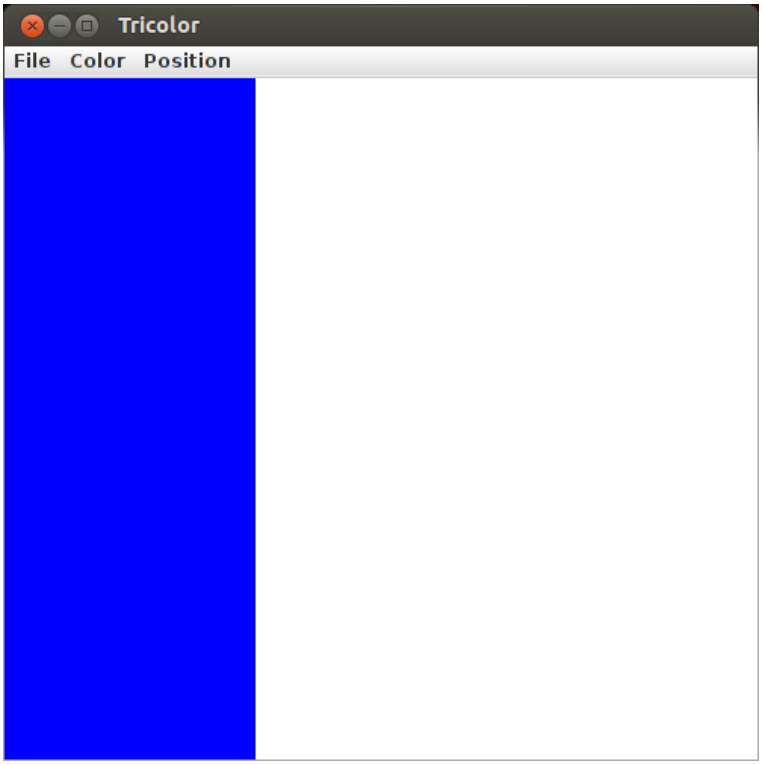

Figure 6: This shows the action of selecting the blue color menu item. Notice that the default position is left.

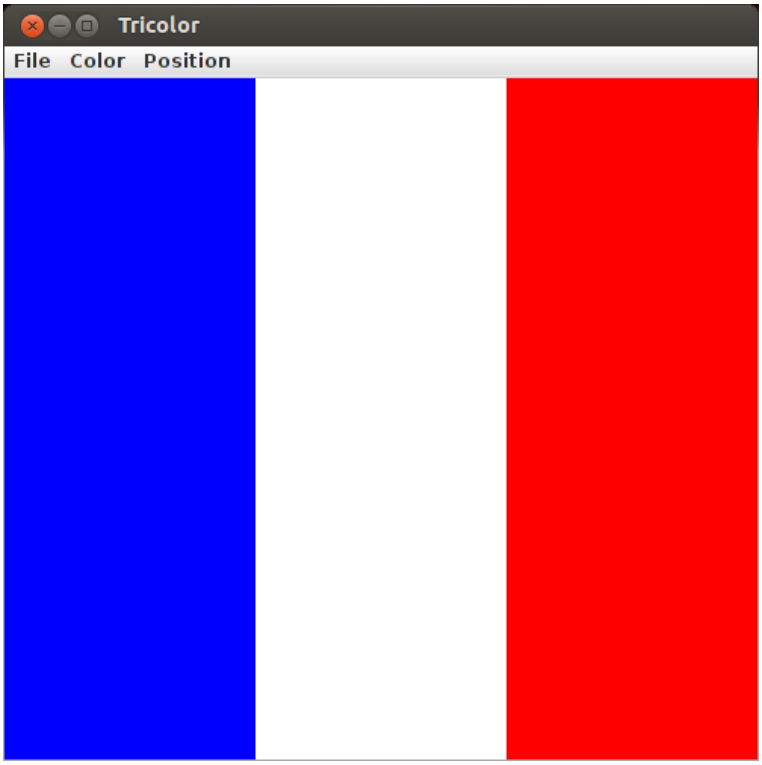

Figure 7: This shows what happens after you select the right position menu item and you select red.