Adding Color

- Color a JFrame as follows:

  JFrame_Object.getContentPane().setBackground(Color);

  If this is inside a constructor for the JFrame, then the expression simplifies to

  getContentPane().setBackground(Color);

  or the equivalent

  this.getContentPane().setBackground(Color);

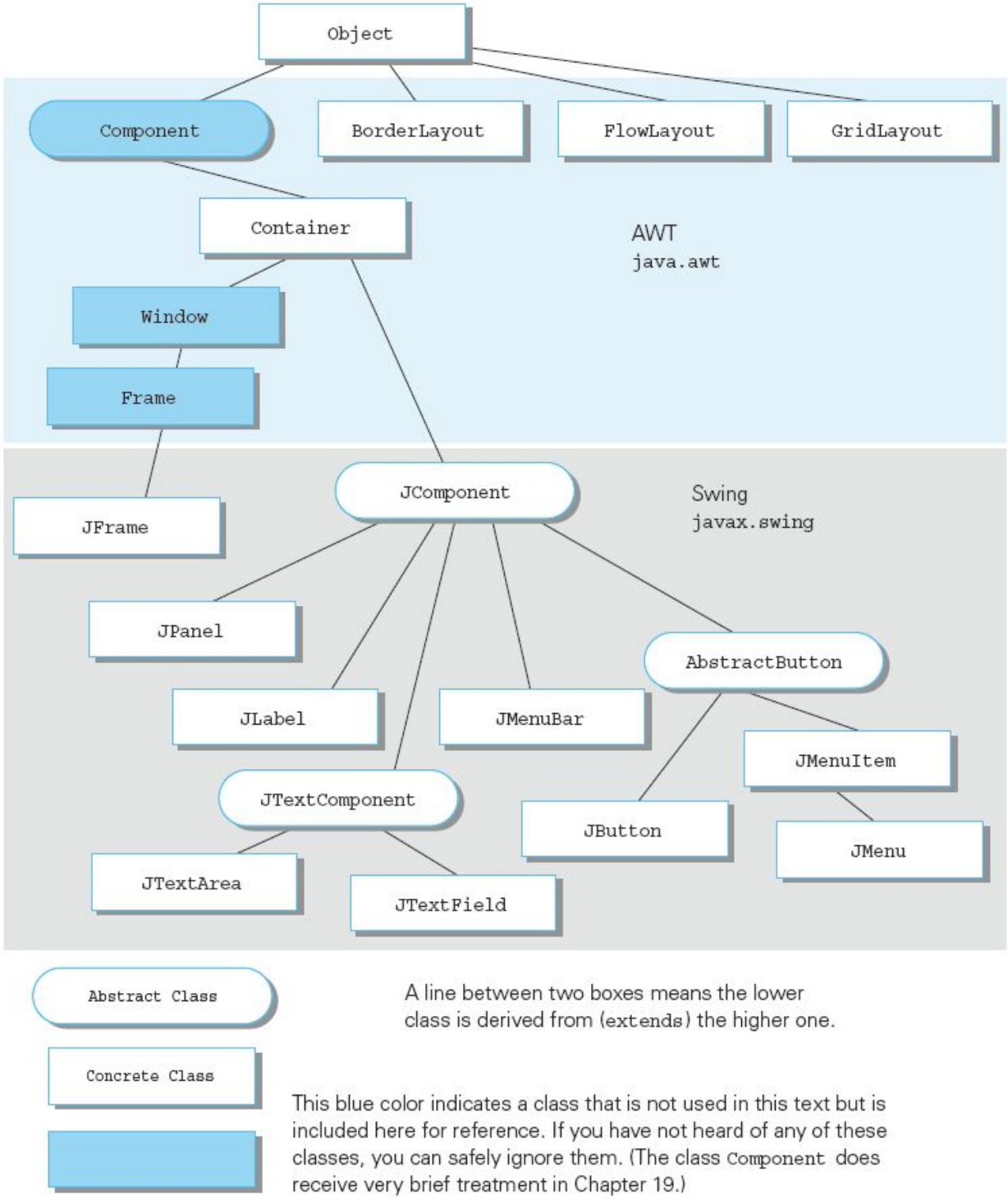
- Color a button, label, or any other component (which is not a JFrame) as follows:

  Component_Object.setBackground(Color);

The Container Class: A container class is any descendent class of the class Container. The class called Container is in the java.awt package. Any descendent class of the class Container can have components added to it.

- The class JFrame is a descendent class of the class Container, so any descendent class of the class JFrame can serve as a container to hold labels, buttons, panels, or other components.
- The class JPanel is a descendent of the class Container, and any object of the class JPanel can serve as a container to hold labels, buttons, other panels, or other components.

Hierarchy of Swing and AWT Classes



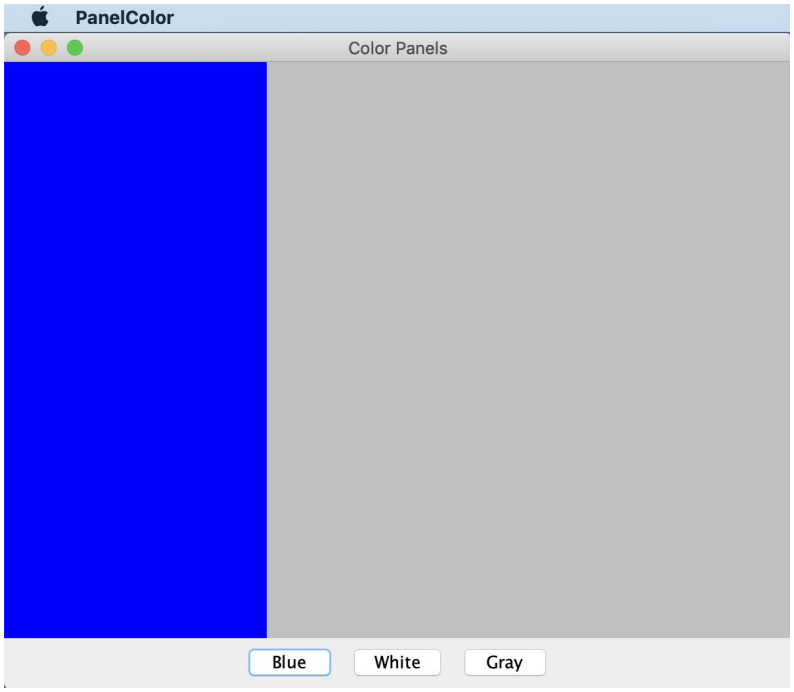
/* Sample code with Panels, Layout, and Buttons. The Panel changes the color on the click of the button. */

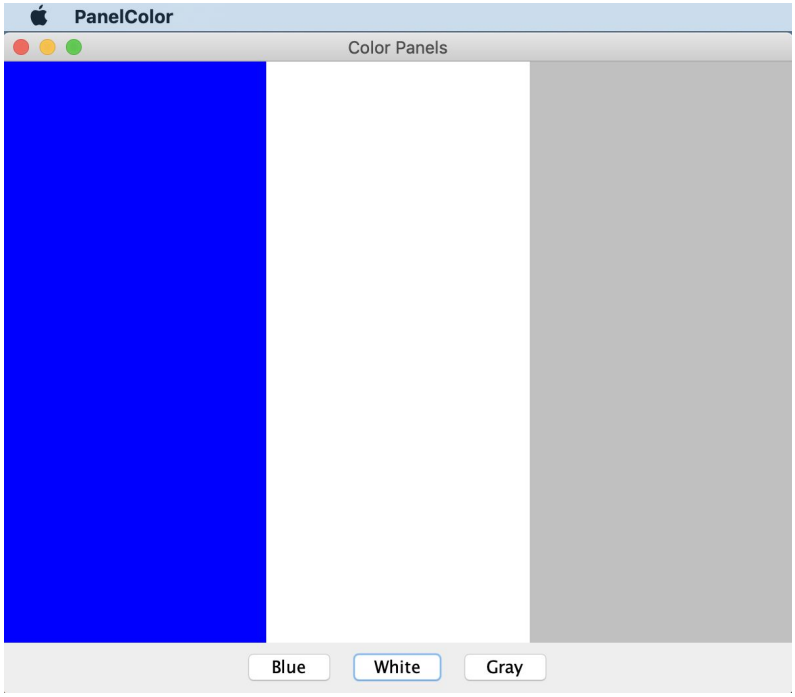
```
import java.awt.*;
import javax.swing.*;
```

```
import java.awt.event.*;

public class PanelColor extends JFrame implements ActionListener
{
   public static final int WIDTH=600;
   public static final int HEIGHT=500;

   private JPanel BluePanel;
   private JPanel WhitePanel;
   private JPanel GrayPanel;

   public static void main(String[] args)
   {
         PanelColor pc=new PanelColor();
         pc.setVisible(true);
   }

   public PanelColor()
   {
         super("Color Panels");
         setSize(WIDTH,HEIGHT);
         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         setLayout(new BorderLayout());

         JPanel BigPanel=new JPanel();
         BigPanel.setLayout(new GridLayout(1,3));

         BluePanel=new JPanel();
         BluePanel.setBackground(Color.LIGHT_GRAY);
         BigPanel.add(BluePanel);

         WhitePanel=new JPanel();
         WhitePanel.setBackground(Color.LIGHT_GRAY);
         BigPanel.add(WhitePanel);

         GrayPanel=new JPanel();
         GrayPanel.setBackground(Color.LIGHT_GRAY);
         BigPanel.add(GrayPanel);

         add(BigPanel,BorderLayout.CENTER);

         JPanel ButtonPanel=new JPanel();
         ButtonPanel.setLayout(new FlowLayout());
```

```
        JButton blueButton=new JButton("Blue");
        blueButton.setBackground(Color.BLUE);
        blueButton.addActionListener(this);
        ButtonPanel.add(blueButton);

        JButton whiteButton=new JButton("White");
        whiteButton.setBackground(Color.WHITE);
        whiteButton.addActionListener(this);
        ButtonPanel.add(whiteButton);

        JButton grayButton=new JButton("Gray");
        grayButton.setBackground(Color.LIGHT_GRAY);
        grayButton.addActionListener(this);
        ButtonPanel.add(grayButton);

        add(ButtonPanel,BorderLayout.SOUTH);

    }

    public void actionPerformed(ActionEvent e)
    {
        String buttonString=e.getActionCommand();

        if(buttonString.equals("Blue"))
            BluePanel.setBackground(Color.BLUE);
        else if(buttonString.equals("White"))
            WhitePanel.setBackground(Color.WHITE);
        else if(buttonString.equals("Gray"))
            GrayPanel.setBackground(Color.GRAY);
        else
            System.out.println("Error");

    }
}
```

//Sample Outputs


PanelColor
Color Panels
Blue
White
Gray


PanelColor
Color Panels
Blue
White
Gray

Menus

A menu is an object of the class **JMenu**. A choice on a menu is an object of the class **JMenuItem**. Menus are collected together in a menu bar (or menu bars). A menu bar is an object of the class **JMenuBar**.

Events and listeners for menu items are handled in exactly the same way as they are for buttons.

JMenuItem ---> JMenu ----> JMenuBar

The class **JMenu** is a descendent of the **JMenuItem** class. So, every **JMenu** object is also a **JMenuItem** object. Thus, a **JMenu** can be a menu item in another menu. This means that you can nest menus.

Adding Menus to a JFrame

- **Creating Menu Items**

  A menu item is an object of the class JMenuItem. Create a new menu item in the usual way, as illustrated by the following example. The string in the argument position is the displayed text for the menu item.

  JMenuItem redChoice = new JMenuItem("Red");

- **Adding Menu Item Listeners**

  Events and listeners for menu items are handled in the exact same way as they are for buttons: Menu items fire action events that are received by objects of the class ActionListener.

  Syntax

  JMenu_Item_Name.addActionListener(Action_Listener);

  Example

  redChoice.addActionListener(this);

- **Creating a Menu**

  A menu is an object of the class JMenu. Create a new menu in the usual way, as illustrated by the following example. The string argument is the displayed text that identifies the menu.

JMenu colorMenu = new JMenu("Add Colors");

- **Adding Menu Items to a Menu**

  Use the method add to add menu items to a menu.

  Syntax

  JMenu_Name.add(JMenu_Item);

  Example (colorMenu is an Object of the Class JMenu)

  colorMenu.add(redChoice);

- **Creating a Menu Bar**

  A menu bar is an object of the class **JMenuBar**. Create a new menu bar in the usual way, as illustrated by the following example:

  JMenuBar bar = new JMenuBar();

- **Adding a Menu to a Menu Bar**

  Add a menu to a menu bar using the method add as follows:

  Syntax

  JMenu_Bar_Name.add(JMenu_Name);

  Example (bar is an Object of the Class JMenubar)

  bar.add(colorMenu);

- **Adding a Menu Bar to a Frame**

  There are two different ways to add a menu bar to a JFrame. You can use the method add to add the menu bar to a JFrame (or to any other container). Another common way of adding a menu bar to a JFrame is to use the method setJMenuBar as follows:
  Syntax

  setJMenuBar(JMenu_Bar_Name);

Example

setJMenuBar(bar);

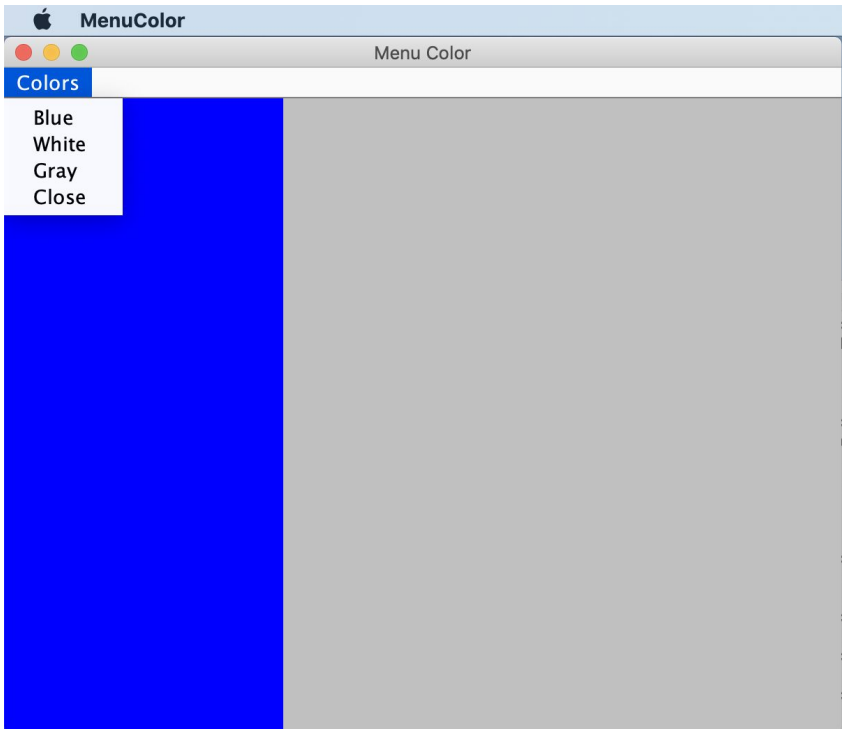
//Sample Code

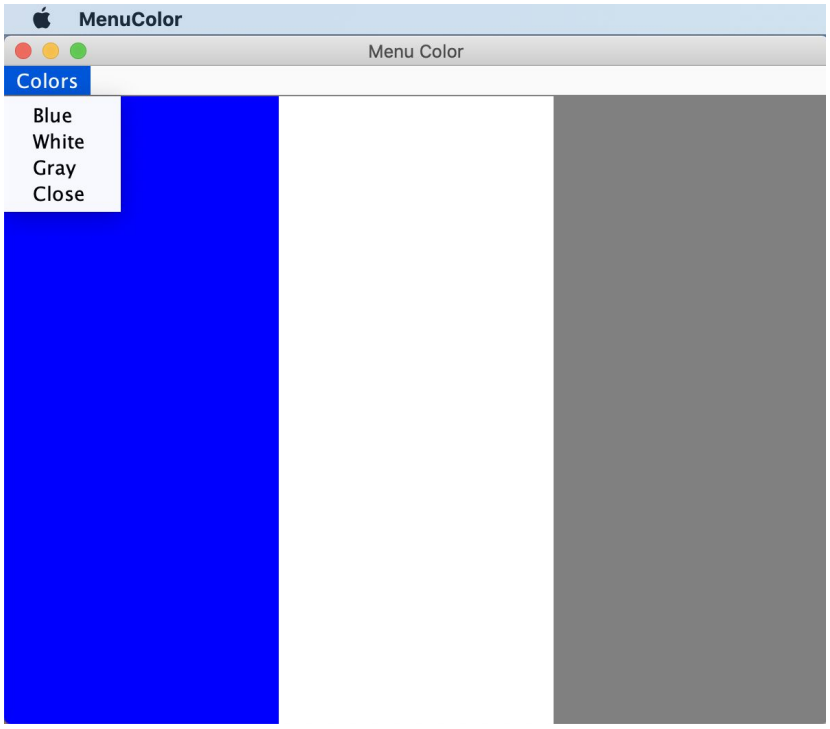
```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class MenuColor extends JFrame implements ActionListener
{
   public static final int WIDTH=600;
   public static final int HEIGHT=500;

   private JPanel BluePanel;
   private JPanel WhitePanel;
   private JPanel GrayPanel;

   public static void main(String[] args)
   {
         MenuColor pc=new MenuColor();
         pc.setVisible(true);
   }

   public MenuColor()
   {
         super("Menu Color");
         setSize(WIDTH,HEIGHT);
         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         setLayout(new BorderLayout());

         JPanel BigPanel=new JPanel();
         BigPanel.setLayout(new GridLayout(1,3));

         BluePanel=new JPanel();
         BluePanel.setBackground(Color.LIGHT_GRAY);
         BigPanel.add(BluePanel);

         WhitePanel=new JPanel();
         WhitePanel.setBackground(Color.LIGHT_GRAY);
         BigPanel.add(WhitePanel);
```

```
        GrayPanel=new JPanel();
        GrayPanel.setBackground(Color.LIGHT_GRAY);
        BigPanel.add(GrayPanel);

        add(BigPanel,BorderLayout.CENTER);

        JMenu ColorMenu=new JMenu("Colors");


        JMenuItem bluechoice=new JMenuItem("Blue");
        bluechoice.addActionListener(this);
        ColorMenu.add(bluechoice);

        JMenuItem whitechoice=new JMenuItem("White");

        whitechoice.addActionListener(this);
        ColorMenu.add(whitechoice);

        JMenuItem graychoice=new JMenuItem("Gray");
        graychoice.addActionListener(this);
        ColorMenu.add(graychoice);

        JMenuItem closechoice=new JMenuItem("Close");
        closechoice.addActionListener(this);
        ColorMenu.add(closechoice);

        JMenuBar bar=new JMenuBar();
        bar.add(ColorMenu);
        setJMenuBar(bar);



}

public void actionPerformed(ActionEvent e)
{
        String buttonString=e.getActionCommand();

        if(buttonString.equals("Blue"))
                BluePanel.setBackground(Color.BLUE);
        else if(buttonString.equals("White"))
                WhitePanel.setBackground(Color.WHITE);
```

```
        else if(buttonString.equals("Gray"))
                GrayPanel.setBackground(Color.GRAY);
        else if(buttonString.equals("Close"))
                System.exit(0);
        else
                System.out.println("Error");

    }
}
```

//Sample Output





Some Methods in the Class AbstractButton

The abstract class AbstractButton is in the javax.swing package.

All of these methods are inherited by both of the classes **JButton** and **JMenuItem.**

- **public void setBackground(Color theColor)**

  Sets the background color of this component.

- **public void addActionListener(ActionListener listener)**

  Adds an ActionListener.

- **public void removeActionListener(ActionListener listener)**

  Removes an ActionListener.

- **public void setActionCommand(String actionCommand)**

  Sets the action command.

- **public String getActionCommand()**

Returns the action command for this component.

- **public void setText(String text)**

  Makes text the only text on this component.

- **public String getText()**

  Returns the text written on the component, such as the text on a button or the string for a menu item.

- **public void setPreferredSize(Dimension preferredSize)**

  Sets the preferred size of the button or label. Note that this is only a suggestion to the layout manager. The layout manager is not required to use the preferred size. The following special case will work for most simple situations. The int values give the width and height in pixels.

  public void setPreferredSize(new Dimension(int width, int height))

- **public void setMaximumSize(Dimension maximumSize)**

  Sets the maximum size of the button or label. Note that this is only a suggestion to the layout manager. The layout manager is not required to respect this maximum size. The following special case will work for most simple situations. The int values give the width and height in pixels.

  public void setMaximumSize(new Dimension(int width, int height))

- **public void setMinimumSize(Dimension minimumSize)**

  Sets the minimum size of the button or label. Note that this is only a suggestion to the layout manager. The layout manager is not required to respect this minimum size.

  The following will work for most simple situations. The int values give the width and height in pixels.

  public void setMinimumSize(new Dimension(int width, int height))

The Dimension Class

Objects of the class Dimension are used with buttons, menu items, and other objects to specify a size. The Dimension class is in the package java.awt. The parameters in the following constructor are pixels.

Constructor

Dimension(int width, int height)

Example

aButton.setPreferredSize(new Dimension(30, 50));

setActionCommand and getActionCommand

Every button and every menu item has a string associated with it that is known as the action command for that button or menu item. When the button or menu item is clicked, it fires an action event e. The following invocation returns the action command for the button or menu item that fired e:

e.getActionCommand( )

The method actionPerformed typically uses this action command string to decide which button or menu item was clicked.

The default action command for a button or menu item is the string written on it, but if you want, you can change the action command with an invocation of the method **setActionCommand**. For example, the menu item chooseNext created by the following code will display the string "Next" when it is a menu choice, but will have the string "Next Menu Item" as its action command.

Example

```
JMenuItem chooseNext = new JMenuItem("Next");
chooseNext.setActionCommand("Next Menu Item");
```