Inheritance

Inheritance is the capability of a class to use the properties and methods of another class while adding its own functionality.

Programming In A Graphical Environment

Java is specifically designed to program internet/network based applets and applications in a graphical environment. What is the difference between an applet and an application? Applets are designed to run within a web page while applications run within their own windows.

The Java language provides an extensive collection of *graphical **u**ser **i**nterface* (GUI) components. Inheritance plays a big role in Java programming because all of these components are specifically designed to be extended.

Abstract Windowing Toolkit (AWT) **(Outsource: 12-4)**

The **AWT** is a collection of graphical user interface (GUI) components that were implemented using native-platform versions of the components. The AWT is a part of the core of the Java API (Application Programming Interface).

Swing **(Outsource: 12-5)**

The Java Foundation Classes (JFC) includes the **Swing** classes, which define a complete set of GUI components for JFC applications. An extension to the original Abstract Windowing Toolkit, the JFC includes the Swing classes, pluggable look and feel designs, and the Java Accessibility API, which are all implemented without native code (code that refers to the functions of a specific operating system or is compiled for a specific processor). The JFC components include windows and frames, panels and panes, dialog boxes, menus and toolbars, buttons, sliders, combo boxes, text components, tables, list components, and trees.

Swing widgets provide more sophisticated GUI components than the earlier Abstract Window Toolkit. Since they are written in pure Java, they run the same on all platforms, unlike the AWT which is tied to the underlying platform's windowing system. Swing supports pluggable look and feel – not by using the native platform's facilities, but by roughly emulating them. This means you can get any supported look and feel on any platform. The disadvantage of lightweight components is slower execution. The advantage is uniform behavior on all platforms.

Here is a list of a few of the components located in the AWT and Swing packages:

AWT Component	**Swing Component**	**Purpose**
Frame	JFrame	A top-level window with a title and a border.
Component	JComponent	A *component* is an object

		having a graphical representation that can be displayed on the screen and that can interact with the user.
Panel	JPanel	A panel provides space in which an application can attach any other component, including other panels.
Button	JButton	An implementation of a "push" button.
TextField	JTextField	A component that allows the editing of a single line of text.

JFrame class (Outsource: 12-22)

A graphical user interface has to start with a top-level container. It provides a home for the other components of the interface, and dictates the overall feel of the application. The `JFrame` class is used to create a simple top-level window for a Java application.

Method Summary (JFrame clsss)	
`Container`	`getContentPane()` Returns the `contentPane` object for this frame.
`Graphics`	`getGraphics()` Creates a graphics context for this component.
`void`	`pack()` Causes this Window to be sized to fit the preferred size and layouts of its subcomponents.
`void`	`paint(Graphics g)` Paints this component.
`void`	`repaint()` Repaints this component.
`void`	`setBackground(Color c)` Sets the background color of this component.
`void`	`setContentPane(Container contentPane)` Sets the `contentPane` property. This method is called by the constructor.
`void`	`setDefaultCloseOperation(int operation)` Sets the operation that will happen by default when the user initiates a "close" on this frame. You must specify one of the following choices: • **`DO_NOTHING_ON_CLOSE`** (defined in `WindowConstants`): Don't do anything; require the program to handle the operation in the `windowClosing` method of a registered `WindowListener` object. • **`HIDE_ON_CLOSE`** (defined in `WindowConstants`): Automatically hide the

	frame after invoking any registered `WindowListener` objects. • **`DISPOSE_ON_CLOSE`** (defined in `WindowConstants`): Automatically hide and dispose the frame after invoking any registered `WindowListener` objects. • **`EXIT_ON_CLOSE`** (defined in `JFrame`): Exit the application using the `System exit` method. Use this only in applications. The value is set to **`HIDE_ON_CLOSE`** by default.
`void`	**`setForeground`**`(Color c)` Sets the foreground color of this component.
void	**`setIconImage`**`(Image image)` Sets the image to be displayed as the icon for this window.
void	**`setJMenuBar`**`(JMenuBar menubar)` Sets the menubar for this frame.
void	**`setLayout`**`(LayoutManager manager)` Sets the `LayoutManager`.
void	**`setResizable`**`(boolean resizable)` Sets whether this frame is resizable by the user.
void	**`setSize`**`(int width, int height)` Resizes this component so that it has width `width` and height `height`.
void	**`setTitle`**`(String title)` Sets the title for this frame to the specified string.
void	**`setVisible`**`(boolean b)` Shows or hides this `Window` depending on the value of parameter `b`.

An instance of a `JFrame` object can be instantiated and displayed in just a few lines of code.

```
public class MyFrameExample
{
   public static void main (String[] args) {
      JFrame frame = new JFrame("MyFrame");
      frame.setVisible(true);
   }
}
```

We use the *main* method to instantiate an instance of `MyFrame` and to make it appear on the screen. The purpose of the constructor is to initialize MyFrame. Executing this program will result in a window appearing on the screen

```
public class MyFrame extends JFrame
{
   public static void main (String[] args) {
      JFrame frame = new MyFrame(); //MyFrame IS a JFrame
      frame.setVisible(true);
   }

   public MyFrame() {
      super("MyFrame");
      // or alternatively we could call setTitle("MyFrame");
      setSize(400, 300);
   }
}
```

You can override any method in any class just by declaring a new method with the same method profile in the derived class. The paint method of the `JFrame` class is defined as **public void paint(Graphics g).**

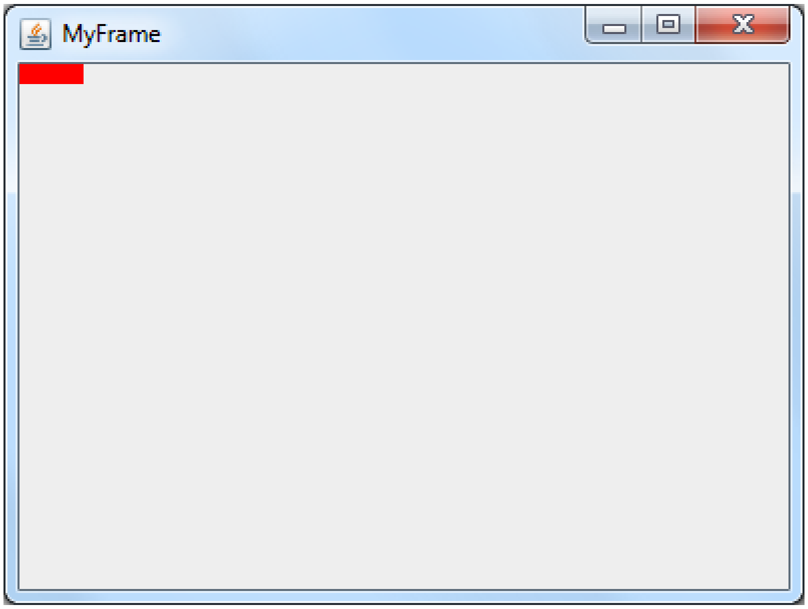
Overriding the paint method results in the following code:

```
public class MyFrame extends JFrame
{
   public static void main (String[] args) {
      JFrame frame = new MyFrame();
      frame.setVisible(true);
   }

   public MyFrame() {
      super("MyFrame");
      setSize(400, 300);
   }

   public void paint(Graphics g) {
      super.paint(g);
      g.setColor(Color.red);
      g.fillRect(0, 0, 40, 40);
   }
}
```

Running this program results in the appearance of the following window:



The `fillRect` method of the `Graphics` class draws a filled rectangle using the current drawing color (which can be changed by calling `setColor`). `fillRect` requires four parameters – the `x` and `y` coordinate for the upper left corner and the `width` and `height`, i.e. `g.fillRect(x, y, width, height)`. The `x` and `y` coordinates are relative to the component being drawn on, in this case the `JFrame`. It is very apparent that the rectangle being drawn is not a rectangle 40 pixels wide by 40 pixels high. That is because part of the rectangle is being hidden by the title bar and the frame border. All drawing on a `JFrame` is relative to the frame not relative to the client area (the part of the window that we should be drawing in).

What we should be doing is drawing on the client area of the `JFrame` which is called the content pane in Java. The simplest way to accomplish that is to utilize another GUI class called a `JPanel`.

Content Panes

Top-level containers (JFrame and JApplet) have several layers (*panes*): *root*, *content*, *layered*, and *glass*. Programs normally reference only the content pane. There are two programming idioms for using the content pane: (1) using the pre-assigned pane, or (2) building your own pane.

Idiom 1: Use the existing content pane

Every frame (or window) has a preconstructed content pane of class Container. You can get this pane and add the components to it. For example,

```
class MyFrame extends JFrame {

    MyFrame()  {     // constructor
        Container content = getContentPane();    // Use the default content pane.
        content.add(...);
        content.add(...);

    }
```

All JFrames already have a content pane, so there's no need to create a new one, just get the existing pane. And if you're wondering about the Container type, it's a superclass of JPanel. In fact, if you look at the *actual* type of the object that's currently returned by getContentPane(), it actually is a JPanel.

Idiom 2: Create your own content pane

It's common to create a new panel for the content pane and tell the window to use this new panel for its content pane. For example,

```
class MyFrame extends JFrame {

    MyFrame()  {     // constructor
        JPanel content = new JPanel();    // Create a new content pane.
        setContentPane(content);
        content.add(...);
        content.add(...);
    }
```

JPanel class (Outsource: 12-26 – 12-28)

The `JPanel` class is a light weight container class that serves two purposes. It can be used to hold other light weight GUI objects and/or it can be used as a drawing surface.

Method Summary (JPanel clsss)	
`Container`	**`getContentPane`**`()` Returns the `contentPane` object for this frame.
`Graphics`	**`getGraphics`**`()` Creates a graphics context for this component.
`void`	**`paint`**`(Graphics g)` Paints this component.
`void`	**`paintComponent`**`(Graphics g)` Paints this component. Called by the **paint** method.
`void`	**`repaint`**`()` Repaints this component.
`void`	**`setBackground`**`(Color c)` Sets the background color of this component.
`void`	**`setContentPane`**`(Container contentPane)` Sets the `contentPane` property. This method is called by the constructor.
`void`	**`setDoubleBuffered`**`(Boolean aFlag)` Sets whether this component should use a buffer to paint.
`void`	**`setForeground`**`(Color c)` Sets the foreground color of this component.
void	**`setLayout`**`(LayoutManager manager)` Sets the `LayoutManager`.
void	**`setPreferredSize`**`(Dimension  preferredSize)` Sets the `LayoutManager`.
void	**`setSize`**`(int width, int height)` Resizes this component so that it has width `width` and height `height`.
void	**`setVisible`**`(boolean b)` Shows or hides this `Window` depending on the value of parameter `b`.

For example, we can add a `JPanel` to a `JFrame` and draw on the `JPanel`. This works much better than drawing directly onto the `JFrame`. However, we will need to create our own JPanel object by extending the JPanel class. Generally this works well as a nested class (a class defined within another class). In Java, nested classes direct access to all fields and methods of the parent class.

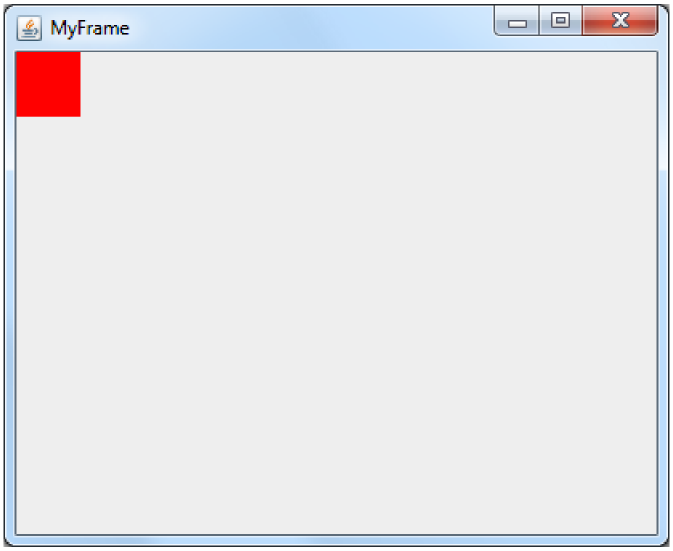
```
public class MyFrame extends JFrame
{
   public class MyPanel extends JPanel
   {
      public MyPanel() {
         setPreferredSize(new Dimension(400, 300));
      }

      public void paintComponent(Graphics g) {
         super.paintComponent(g);
         g.setColor(Color.red);
         g.fillRect(0, 0, 40, 40);
      }
   }

   public static void main (String[] args) {
      JFrame frame = new MyFrame();
      frame.setVisible(true);
   }

   public MyFrame() {
      super("MyFrame");
      setContentPane(new MyPanel());
      pack();
   }
}
```

When we execute the program shown above we get a window that displays a red rectangle that is 40 pixels wide by 40 pixels high.



You will notice that we did the drawing in `paintComponent` rather than in the paint method. Any drawing done in a light-weight component should be accomplished in the `paintComponent` method NOT in `paint`. Override `paint` only in heavy-weight components[1].

Graphics class

The `Graphics` class is the abstract base class for all graphics contexts that allow an application to draw onto components that are realized on various devices, as well as onto off-screen images.

Method Summary	
abstract void	**clearRect**(int x, int y, int width, int height) Clears the specified rectangle by filling it with the background color of the current drawing surface.
abstract void	**dispose**() Disposes of this graphics context and releases any system resources that it is using.
abstract boolean	**drawImage**(Image img, int x, int y, ImageObserver observer) Draws as much of the specified image as is currently available.
abstract void	**drawLine**(int x1, int y1, int x2, int y2) Draws a line, using the current color, between the points (x1, y1) and (x2, y2) in this graphics context's coordinate system.
abstract void	**drawOval**(int x, int y, int width, int height) Draws the outline of an oval.
abstract void	**drawPolygon**(int[] xPoints, int[] yPoints, int nPoints) Draws a closed polygon defined by arrays of *x* and *y* coordinates.
abstract void	**drawPolyline**(int[] xPoints, int[] yPoints, int nPoints) Draws a sequence of connected lines defined by arrays of *x* and *y* coordinates.
void	**drawRect**(int x, int y, int width, int height) Draws the outline of the specified rectangle.
abstract void	**drawRoundRect**(int x, int y, int width, int height, int arcWidth, int arcHeight) Draws an outlined round-cornered rectangle using this graphics context's current color.
abstract void	**drawString**(String str, int x, int y) Draws the text given by the specified string, using this graphics context's current font and color.
abstract void	**fillOval**(int x, int y, int width, int height) Fills an oval bounded by the specified rectangle with the current color.

[1] JFrame, JDialog, and Window classes are heavy-weight components. All other components are light-weight.

abstract void	`fillPolygon(int[] xPoints, int[] yPoints, int nPoints)` Fills a closed polygon defined by arrays of *x* and *y* coordinates.
abstract void	`fillRect(int x, int y, int width, int height)` Fills the specified rectangle.
abstract void	`fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)` Fills the specified rounded corner rectangle with the current color.
abstract void	`setColor(Color c)` Sets this graphics context's current color to the specified color.
abstract void	`setFont(Font font)` Sets this graphics context's font to the specified font.

Color class

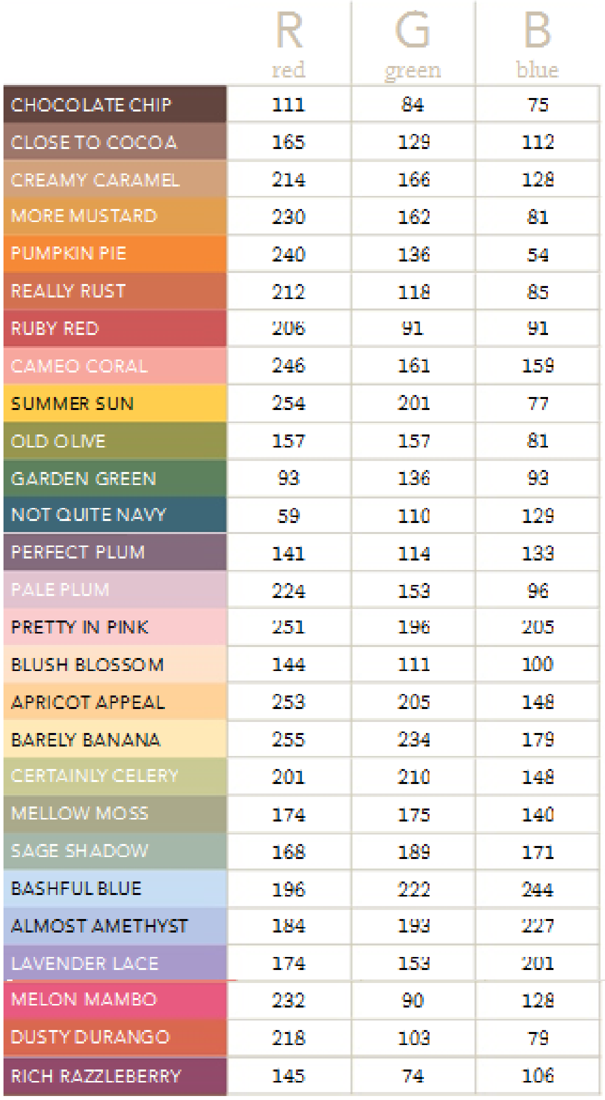
The Color class is used to encapsulate colors in the default sRGB color format. Every color has an implicit alpha value of 255 or an explicit one provided in the constructor. The alpha value defines the transparency of a color and can be represented by an integer value in the range 0 - 255. An alpha value of 255 means that the color is completely opaque and an alpha value of 0 means that the color is completely transparent.

The Color class contains the following constant values that represent Color objects:

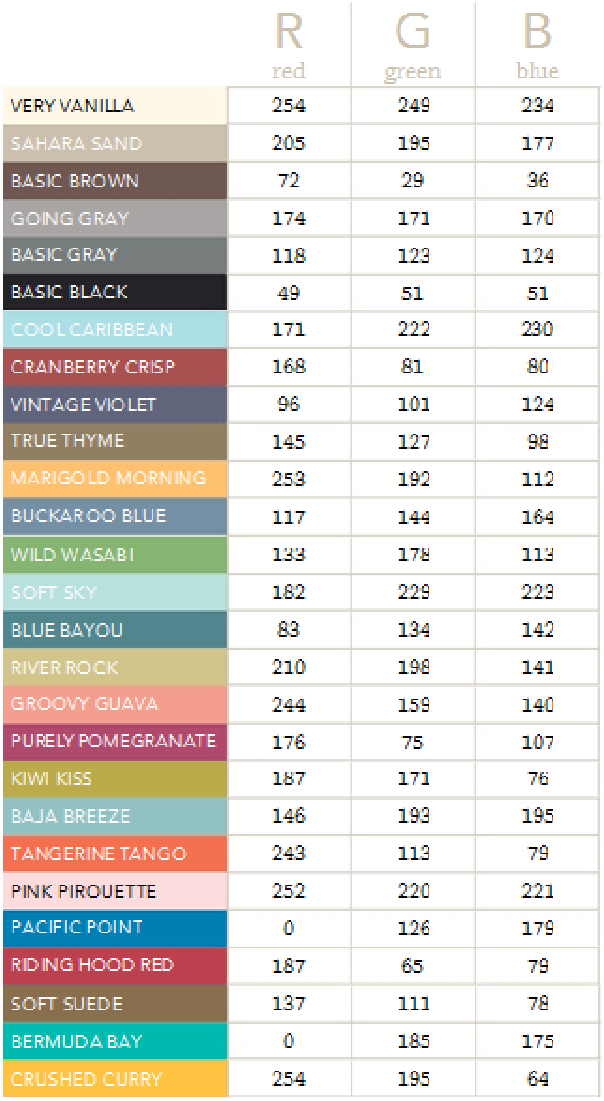
Color.BLACK	Color.black
Color.BLUE	Color.blue
Color.CYAN	Color.cyan
Color.DARK_GRAY	Color.darkGray
Color.GRAY	Color.gray
Color.GREEN	Color.green
Color.LIGHT_GRAY	Color.lightGray
Color.MAGENTA	Color.magenta
Color.ORANGE	Color.orange
Color.PINK	Color.pink
Color.RED	Color.red
Color.WHITE	Color.white
Color.YELLOW	Color.yellow

Constructor Summary
`Color(int r, int g, int b)` Creates an opaque sRGB color with the specified red, green, and blue values in the range (0 - 255).
`Color(int r, int g, int b, int a)` Creates an sRGB color with the specified red, green, blue, and alpha values in the range (0 - 255).

Color Values



	R red	G green	B blue
CHOCOLATE CHIP	111	84	75
CLOSE TO COCOA	165	129	112
CREAMY CARAMEL	214	166	128
MORE MUSTARD	230	162	81
PUMPKIN PIE	240	136	54
REALLY RUST	212	118	85
RUBY RED	206	91	91
CAMEO CORAL	246	161	159
SUMMER SUN	254	201	77
OLD OLIVE	157	157	81
GARDEN GREEN	93	136	93
NOT QUITE NAVY	59	110	129
PERFECT PLUM	141	114	133
PALE PLUM	224	153	96
PRETTY IN PINK	251	196	205
BLUSH BLOSSOM	144	111	100
APRICOT APPEAL	253	205	148
BARELY BANANA	255	234	179
CERTAINLY CELERY	201	210	148
MELLOW MOSS	174	175	140
SAGE SHADOW	168	189	171
BASHFUL BLUE	196	222	244
ALMOST AMETHYST	184	193	227
LAVENDER LACE	174	153	201
MELON MAMBO	232	90	128
DUSTY DURANGO	218	103	79
RICH RAZZLEBERRY	145	74	106



	R red	G green	B blue
VERY VANILLA	254	249	234
SAHARA SAND	205	195	177
BASIC BROWN	72	29	36
GOING GRAY	174	171	170
BASIC GRAY	118	123	124
BASIC BLACK	49	51	51
COOL CARIBBEAN	171	222	230
CRANBERRY CRISP	168	81	80
VINTAGE VIOLET	96	101	124
TRUE THYME	145	127	98
MARIGOLD MORNING	253	192	112
BUCKAROO BLUE	117	144	164
WILD WASABI	133	178	113
SOFT SKY	182	229	223
BLUE BAYOU	83	134	142
RIVER ROCK	210	198	141
GROOVY GUAVA	244	159	140
PURELY POMEGRANATE	176	75	107
KIWI KISS	187	171	76
BAJA BREEZE	146	193	195
TANGERINE TANGO	243	113	79
PINK PIROUETTE	252	220	221
PACIFIC POINT	0	126	179
RIDING HOOD RED	187	65	79
SOFT SUEDE	137	111	78
BERMUDA BAY	0	185	175
CRUSHED CURRY	254	195	64

Import statements

A minimum of two import statements are required for GUI applications:

```
import java.awt.*;        // Imports the Abstract Windowing Toolkit package.
import javax.swing.*;     // Imports the Swing package.
```

Audio Clips

Java supports several different types of audio formats among them being the **.wav** format. A **.wav** file can be loaded into an instance of an **`AudioClip`**. The sound can then be played by calling the **`play()`** method of the **`AudioClip`** class.

The **AudioClip** class is located in the applet folder so will need the following import:

```
import java.applet.AudioClip;
```

I have provided you with a **Utility** class that includes two methods: one to load audio files and another to load image files.

Once the **.wav** file is stored in an instance of an **AudioClip**, the sound can be played by calling the **play()** method.

LAB 20 - ASSIGNMENT

Lab 20A - 50 points

OBJECTIVE

WAP that displays the French Flag in a GUI window and plays the French National Anthem. The Flag of France consists of three broad vertical stripes: blue, white, and red.

Your program should extend the **`JFrame`** class. Your program will need only two methods - a `main` method and a `constructor`.

Your program will also need a nested class that extends the JPanel class. All drawing will be done in the paintComponent method of the nested class.

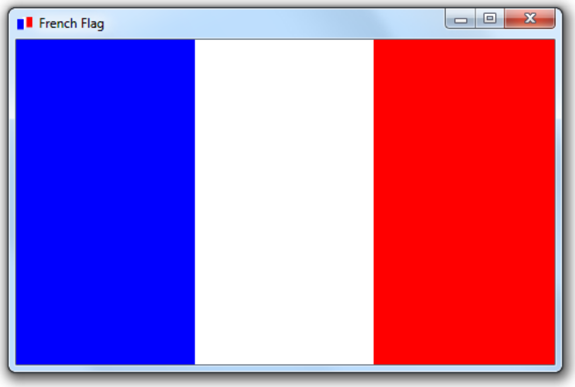
METHOD SUMMARY – JFrame class

- **main** – Instantiate an instance of your class. Make the frame visible.
- **constructor** – Set the title to “French Flag”. Set the default close operation to JFrame.EXIT_ON_CLOSE. Set the Icon image to “flag.gif”. Set the content pane to be an instance of your nested JPanel class. Load the audio file “la_marseillaise.wav”. Play the audio clip, and pack the frame.

METHOD SUMMARY – nested JPanel class

- **constructor** – Set the preferred size of the JPanel to 500 x 350. Set the background color to white.
- **paintComponent** – Draw a blue rectangle that occupies the first 1/3 of the JPanel. Draw a red rectangle that occupies the last **1/3** of the JPanel. Use the JPanel’s getWidth() and getHeight() methods to determine the JPanel’s current width and height. These values will change if you resize the JFrame. You can determine the width of each stripe by dividing the width of the panel by 3, i.e. **int width = getWidth() / 3;**

SAMPLE OUTPUT



Lab 20B - 60 points

OBJECTIVE

WAP that draws a 3D box in a GUI window. You can draw a 3D box by drawing two rectangles and then connecting the corners with lines.

Your program should extend the **`JFrame`** class. Your program will need only two methods - a `main` method and a `constructor`.

Your program will also need a nested class that extends the JPanel class. All drawing will be done in the paintComponent method of the nested class.

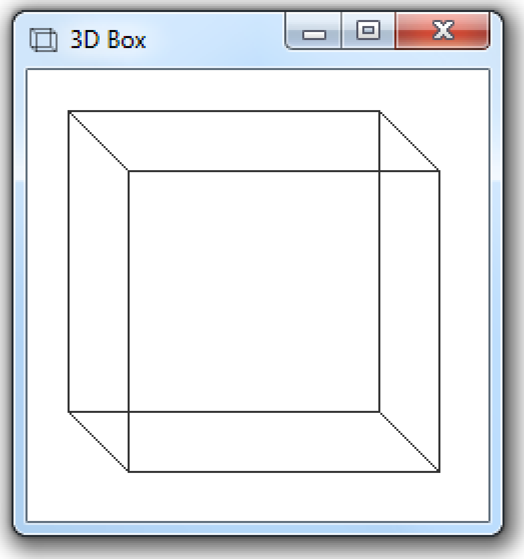
METHOD SUMMARY – JFrame class

- **main** – Instantiate an instance of your class. Make the frame visible.
- **constructor** – Set the title to "3D Box". Set the minimum size to 200 x 200. Set the default close operation to `JFrame.EXIT_ON_CLOSE`. Set the Icon image to "3dbox.gif". Set the content pane to be an instance of your nested JPanel class. Pack the frame.

METHOD SUMMARY – nested JPanel class

- **constructor** – Set the preferred size of the `Component` to 300 x 300. Set the background color to any color your want.
- **paintComponent** – draw a 3D box using line drawing commands available in the `Graphics` class. Use any color you want. Hint: Draw two rectangles and connect the corners with lines.

SAMPLE OUTPUT



Lab 20C - 70 points

OBJECTIVE

WAP that draws a pyramid in the desert on a starry night. You can draw a pyramid using the fillPolygon method of the Graphics class.

Your program should extend the **`JFrame`** class. Your program will need only two methods - a `main` method and a `constructor`.

Your program will also need a nested class that extends the JPanel class. All drawing will be done in the **`paintComponent`** method of the nested class.

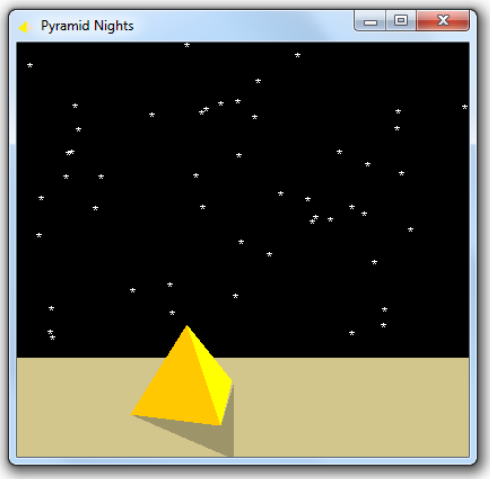
METHOD SUMMARY – JFrame

- **main** – Instantiate an instance of your class. Make the frame visible.
- **constructor** – Set the title to "Pyramid Nights". Set the default close operation to JFrame.EXIT_ON_CLOSE. Set the Icon image to "pyramid.gif". Set the content pane to be an instance of your nested JPanel class. Pack the frame.

METHOD SUMMARY – nested JPanel class

- **constructor** – Set the preferred size of the `Component` to 600 x 500. Set the background color to black.
- **paintComponent** – Draw a filled rectangle that occupies the lower 1/3 if the JPanel. Draw two triangles that intersect and form a 3D pyramid using two **fillPolygon** commands. Sprinkle some stars into the night using **drawString**.

SAMPLE OUTPUT



Lab 20D - 80 points

OBJECTIVE

WAP that draws a square, a circle and a triangle. Each shape should be a different color and should have its alpha value set to 100. The colors should be randomly generated so that they are different each time the panel redraws itself.

Your program should extend the **`JFrame`** class. Your program will need only two methods - a `main` method and a `constructor`.

Your program will also need a nested class that extends the JPanel class. All drawing will be done in the paintComponent method of the nested class.

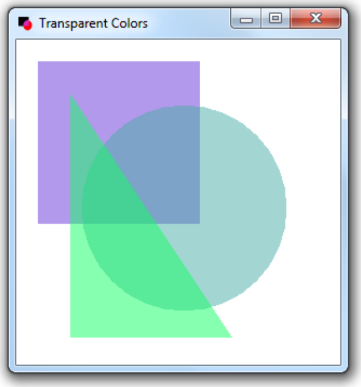
METHOD SUMMARY – JFrame class

- **main** – Instantiate an instance of your class. Make the frame visible.
- **constructor** – Set the title to "Transparent Colors". Set the default close operation to JFrame.EXIT_ON_CLOSE. Set the Icon image to "transparent.gif". Set the content pane to be an instance of your nested JPanel class. Pack the frame.

METHOD SUMMARY – nested JPanel class

- **constructor** – Set the preferred size of the `JPanel` to 400 x 300. Set the background color to WHITE.
- **paintComponent** – Draw a square, a circle, and a triangle. Use the `fillPolygon` method to draw the triangle. Each shape should be in a different random color. Make calls to `randomColor` to set the drawing color to a new random `Color` before drawing each shape.
- **randomColor** – return a random `Color` with the alpha component set to 128.

SAMPLE OUTPUT



Lab 20E - 90 points

OBJECTIVE

WAP that displays Pecos Bill in a desert scene as shown in the sample output below. The image icon name is “pecos.png”.

All the required images are included in the **resources** folder. In addition a sound file “music.wav” is also stored in the resources folder. Your program should play the sound file when the frame is displayed. Your picture doesn’t have to look exactly like mine. Be creative.

Drawing Bitmaps

The drawImage method of the Graphics class requires four arguments. The first argument is an instance of an Image. The next two arguments are the x and y coordinates of the upper left corner of the image (for placement purposes), the last argument is an instance of an **ImageObserver**. This argument can be **null**. We do not need an **ImageObserver**.

```
g.drawImage(myImage, 10, 20, null);
```

Scaling Bitmaps

The **Graphics** class will allow you to stretch (resize) a bitmap. The **drawImage** method is an overloaded method. It includes a version that takes as arguments the **image** to be drawn, **x** and **y** coordinates where the image should be placed, the **width** and **height** of the drawn image, and an **ImageObserver**, which can still be **null**. For example, given an Image called **myImage** whose dimensions are 30 by **60** (30 pixels wide and 60 pixels high), we can double the size of the image by making the following call:

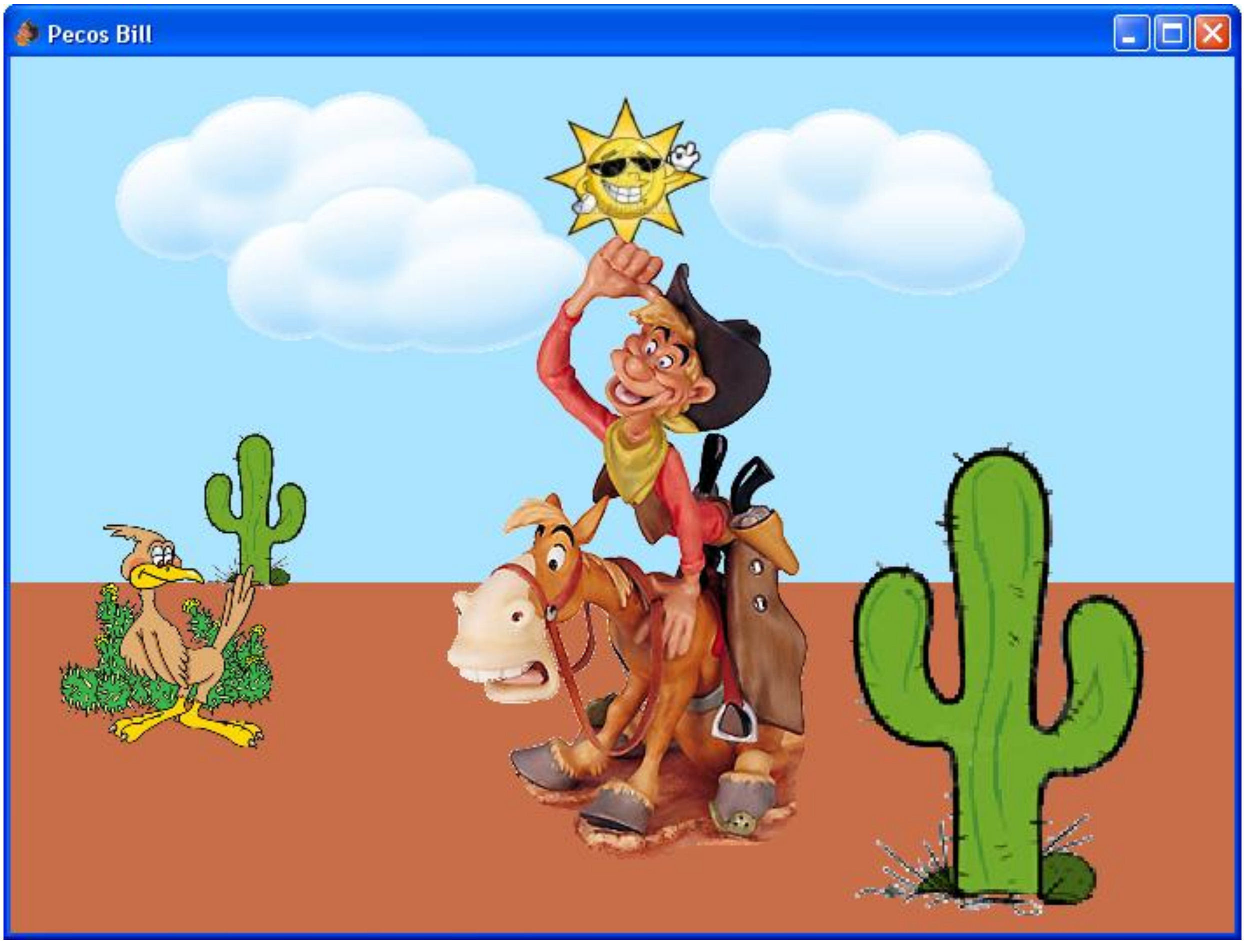
image width is **30 - 60** doubles the *width*

```
g.drawImage(myImage, 10, 20, 60, 120, null);
```

image height is **60 - 120** doubles the *height*

increase both **width** and **height** proportionally to maintain proper aspect ration

where 10 is the **x** coordinate, 20 is the **y** coordinate, 60 is the **width** of the drawn image, and 120 is the **height** of the drawn image (the image will be twice as big).

SAMPLE OUTPUT


Lab 20F - 100 points

OBJECTIVE

WAP that draws a checkerboard.

Your program should extend the **JFrame** class. Your program will need only two methods - a `main` method and a `constructor`.

Your program will also need a nested class that extends the JPanel class. All drawing will be done in the paintComponent method of the nested class.

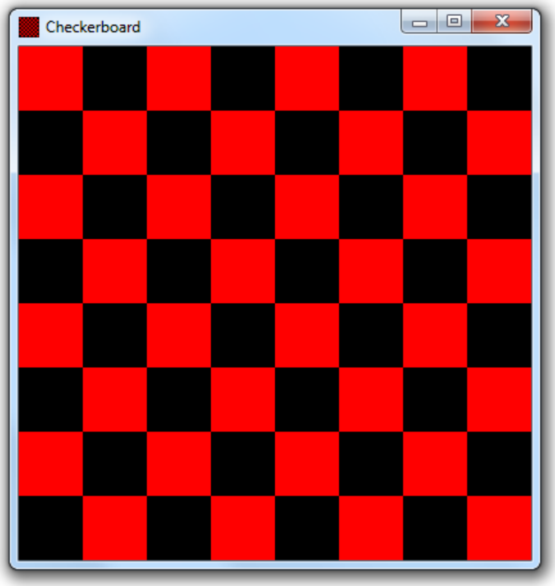
METHOD SUMMARY – JFrame class

- **main** – Instantiate an instance of your class. Make the frame visible.
- **constructor** – Set the title to "Checkerboard". Set the default close operation to JFrame.EXIT_ON_CLOSE. Set the Icon image to "checkerboard.gif". Set the content pane to be an instance of your nested JPanel class. Pack the frame.

METHOD SUMMARY – nested JPanel class

- **constructor** – Set the preferred size of the `JPanel` to 400 x 400. Set the background color to BLACK.
- **paintComponent** – Using a nested loop to control the y and x values. Draw a red checkerboard pattern on the black background. The width of each square should be 1/8th the width of the panel.

SAMPLE OUTPUT


Checkerboard