Class 19

- Paint(Graphics g) draws JFrames and
- paintComponent(Graphics g) draws JPanels, JButtons, JLabels etc

- Like the garbage collector, paint(...) and paintComponent(...) work behind the scenes. An application does not explicitly invoke paint(...) or paintComponent(...). That's done by the system.

- the system first calls paint(…), which paints a plain, unadorned, boring,
- JFrame and then calls paintComponent() to render each of the five JButtons.

Every component that can be drawn on the screen (JButton, JLabel, JFrame, JPanel) has an associated Graphics object that holds information about the component such as color, size, and font. Remember, objects hold data.

When a component is to be drawn the Graphics object for that component is automatically passed to paint(..) or paintCompoent(..). So paint(..) or paintComponent(..) knows how to draw the object-- the size, color, etc.

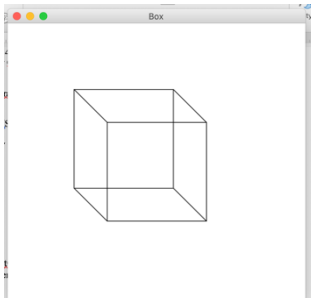
A component's Graphics object is also called the component's *graphics context*. Without the graphics context g, paint(...) cannot do its job; paint(...) needs information.

So, the paint(...) and paintComponent(...) methods use the information contained in the Graphics object g to render a component. And, when the system calls paint(...) or paintComponent(...), it also sends along the graphics context of the particular component via the parameter g.

So why bother looking at these methods?

If an application must draw an image on a frame or panel, such as a 3-dimensional surface or a simple stick figure you have to override the paint(..) or paintComponent(..) methods.

Suppose the we make a class BoxFrame extends JFrame and we want the new frame to look like this:



In order to do that, **BoxFrame must override the paint(Graphics g) method of JFrame** and tell paint(..) how to draw the box.

Here is a quick recap

1. Whenever a JFrame is drawn on the screen the paint(Graphics g) method is automatically called

2. Whenever any other component (JPanel, JButton, JLabet..) is rendered the paintComponent(Graphics g) method is called

3. Every component comes supplied with a Graphics object (its baggage) that has information about how it should be drawn. If a JButton is drawn, its particular Graphics object is passed by the system to paintComponent(Graphics g) so that paintComponent(Graphics g) knows how the button should be drawn; if a JFrame is drawn, its particular Graphics object is passed to paint(Graphics g).

4. If a JFrame or JPanel or any component is to be customized (as the frame above) you need to override paint(..) or paintComponent(..) to do the customization.

So how do we draw on a JFrame or JPanel? How do we make a JFrame of a JPanel with a box or stick figure . We use the methods of the Graphics class.

"Painting" on Panels

Custom "painting" or drawing is usually done on a panel. To paint or draw on a panel,

- extend the JPanel class, and
- override the paintComponent(Graphics g) method so that the redefined paintComponent(...) renders the panel with some customized image or text.
- When overriding the paint() or paintComponent() methods we usually use the methods of the Graphics class, which I will now discuss

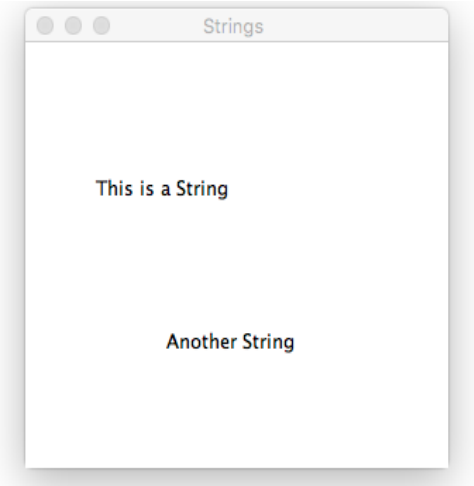
The following methods of the Graphics class are among the most useful of more than three dozen methods that can be invoked by the Graphics object of a component.

Some methods of the Graphics class: drawString(..) setFont(..) and setColor(…)

void drawString(String message, int x, int y)
draws message on the component, starting at position (x, y).
g.drawString("This is a string, 100,300)

The following code creates this frame by overriding paintComponent(Graphics g) and

1. draws two strings on a JPanel and then
2. places the JPanel in a frame



Here is the code. Notice I made two classes

```
import java.awt.*;
import javax.swing.*;
public class StringPanel extends JPanel
{

  public void paintComponent(Graphics g)
  {
    //call paintComponent of the parent
    super.paintComponent(g);
    g.drawString("This is a String", 50,100);// at (50,100)
    g.drawString("Another String", 100, 200);
   }
}
// NOTICE I CALL THE METHODS WITH g
```

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

public class ShowStringFrame extends JFrame
{
 public ShowStringFrame() // constructor
 {
   super("Strings ");
   setBounds(0,0,300,300);
   setBackground(Color.WHITE);
   JPanel panel = new StringPanel();
   add(panel);
   setVisible(true);
 }
 public static void main(String[] args)
 {
  JFrame f = new ShowStringFrame();
 }
}
```

Notice that I first made a class that extends JPanel and overrode paintComponent**.** Then I made a second class that extends JFrame and used the JPanel from the first class. That is:

1. make a class that extends JPanel
2. Make a class that extends JFrame and use the panel of step 1

Also notice that I call drawstring using g –since drawString(..) is a member of the Graphics class and g is a graphics reference

OK..We can now paint a String on a panel and place that panel in the frame. So now let's make the string a little fancier. We can choose a particular fond such as Arial, point size, Bold or plain etc.

To do that we use the Graphics method

void setFont(Font f)

that sets the font used when drawing characters.

Here is a statement showing how it works

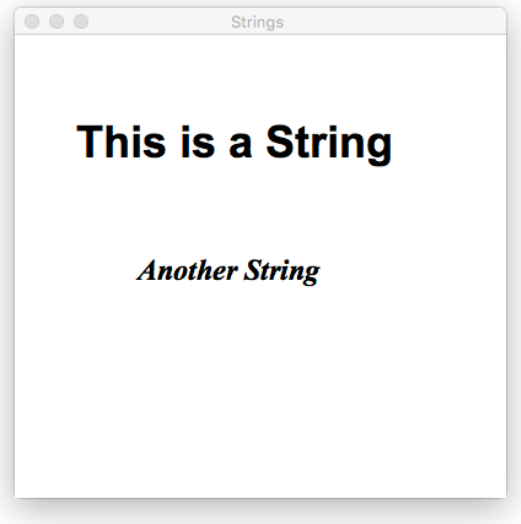
```
Font font = newFont("Arial",Font.BOLD, 24);  // point size is 24
g.setFont(f);
```

Here I created a Font object and passed it to setFont(..).
To create a Font object use the constructor

Font(String name, int style, int size)

where name is the name of a standard font such as "Times New Roman "or "Arial,"
style is a one of these:
Font.PLAIN, Font.BOLD, Font.ITALIC, or Font.BOLD+Font.ITALIC,
And size is the point size.

Now we will change the font of the previous frame and make this frame:



The first string was drawn using Bold Arial size 36. The second string was drawn with a Times New Roman font, size 25, bold and italic. This is done by overriding paintComponent(..)
Here is the code. Again I made a JPanel class and a JFrame class . Two separate classes

```
import java.awt.*;
import javax.swing.*;
public class StringPanel extends JPanel
{

  public void paintComponent(Graphics g)
  {
    //call paintComponent of the parent
    super.paintComponent(g);

    Font font = new Font("Arial", Font.BOLD, 36);
    g.setFont(font);
    g.drawString("This is a String", 50,100

    font = new Font("Times New Roman",
                    Font.BOLD+Font.ITALIC, 24);
    g.setFont(font);
   g.drawString("Another String", 100, 200      }
}
```

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

public class ShowStringFrame extends JFrame
{
  public ShowStringFrame()
  {
    super("Strings ");
    setBounds(0,0,400,400);
    setBackground(Color.WHITE);
    JPanel panel = new StringPanel();
    add(panel);
    setVisible(true);
  }
 public static void main(String[] args)
 {
   JFrame f = new ShowStringFrame();
 }
}
```

OK…We can draw a Srting on a JPanel, choose the font, now we will choose the color that will be used. We use the method

void setColor (Color c)

of the Graphics class. This can be done using any of the standard colors

RED, WHITE, BLUE, GREEN, YELLOW, BLACK, CYAN, MAGENTA, PINK, ORANGE, GRAY, LIGHTGRAY, and DARKGRAY.

For example

g.setColor(Color.RED) or g.setColor(Color.CYAN)

or

you can make your own color by creating a Color object with the constructor

Color (int red, int green, int blue) where red, green and blue are integers from 0 to 255
0 means none of that color and 255 is the most intense

For example

```
Color c = new Color(255,0,0) // all red
g.setColor(c)
```

or

```
Color c = new Color(100, 0, 100); half red halsfblue – purple
G,setColor(c);
```

So now add color to our simple frame

Here is the code:

```
import java.awt.*;
import javax.swing.*;
public class StringPanel extends JPanel
{

    public void paintComponent(Graphics g)
    {
     //call paintComponent of the parent
     super.paintComponent(g);
     Font font = new Font("Arial", Font.BOLD, 36);
     g.setFont(font);

     g.setColor(Color.MAGENTA);

      g.drawString("This is a String", 50,100
     font = new Font("Times New Roman",
                    Font.BOLD+Font.ITALIC, 24);

       Color c = new Color(200, 100, 0);
                                  // mix red and green
        g.setColor(c);

      g.setFont(font);
     g.drawString("Another String", 100, 200);
    }
}
```

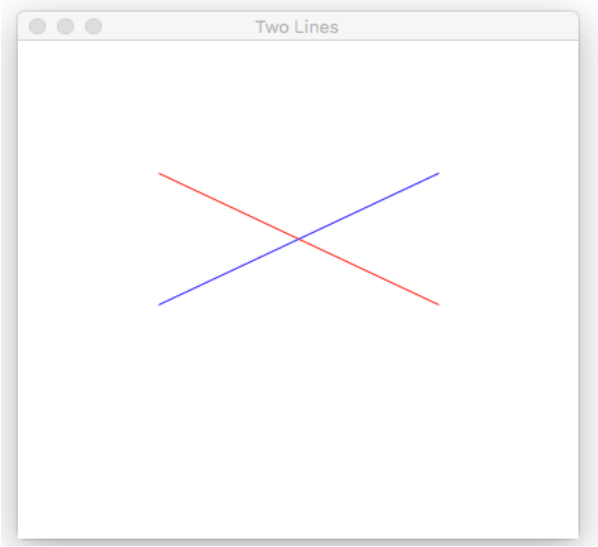
```
import java.awt.*;
import javax.swing.*;

public class ShowStringFrame extends JFrame
{
 public ShowStringFrame()
 {
   super("Strings ");
   setBounds(0,0,400,400);
   setBackground(Color.WHITE);
   JPanel panel = new StringPanel();
   add(panel);
   setVisible(true);
 }
 public static void main(String[] args)
 {
  JFrame f = new ShowStringFrame();
 }
}
```

Drawing Shapes

The Graphics class defines a number of methods that facilitate drawing various shapes on a panel. Among the most commonly used methods are:

void drawLine(int startx, int starty, int endx, int endy)
draws a line segment from point (startx, starty) to point (endx, endy).

Here is a simple program that draws two lines as in the frame below:



Notice I again made 2 classes. One extends JPanel and one extends JFrame. The second uses the JPanel
The drawing is done on a JPanel and then that panel is added to a frame. NOTICE. drawLine(…) is a member of the Graphics class so it is called as **g.drawLine(..). I also used setColor(..)**

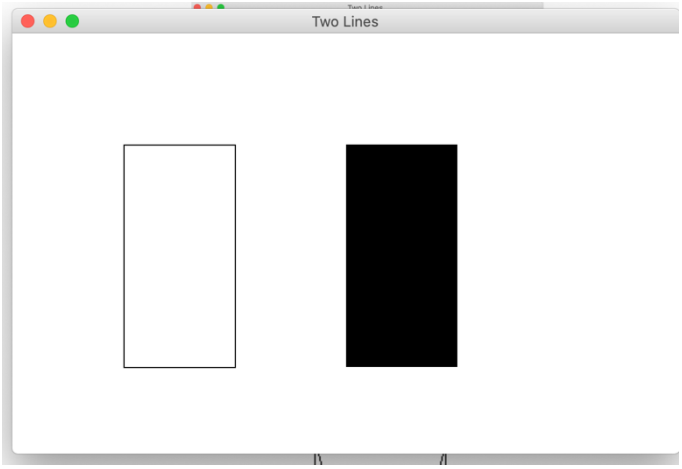
```
import javax.swing.*;
import java.awt.*;

public class TwoLinePanel extends JPanel
{
    public void paintComponent(Graphics g)
    {
      // Call the  paintComponent method of the parent
        super.paintComponent(g);

        g.setColor(Color.RED);
        g.drawLine(100,100, 300,200);

        g.setColor(Color.BLUE);
        g.drawLine(100,200, 300,100);
    }
}
```

```
import javax.swing.*;
import java.awt.*;
// Uses TwoLinePanel
public class LineFrame extends JFrame
 {
    LineFrame()  // default constructor
    {
     super ("Two Lines");  // constructor of parent
     setBounds(0,0,400,400);
     setBackground(Color.WHITE);
     JPanel p = new TwoLinePanel();
     add(p);
     setVisible(true);
   }
   public static void main(String [] args)
    {
        JFrame frame = new LineFrame();
    }
}
```

void drawRect(int x, int y, int width, int height)
draws a rectangle with upper left-hand corner positioned at (x,y). The width and height of the rectangle are width and height respectively.

void fillRect(int x, int y, int width, int height)
draws and fills the specified rectangle.

So here is some simple code that draws this Frame with two rectangles.

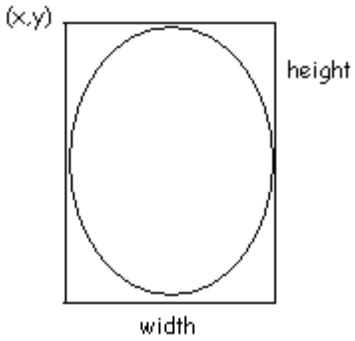


Again I made a class that extended JPanel and overrode PaintComponent(). I made a second class that extended JFrame and added the panel to the frame. So, I made 2 classes.

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

public class RectanglePanel extends JPanel
{
   // override  paintComponent()
   public void paintComponent(Graphics g)
   {
      // Call the paintComponent method of the parent
      super.paintComponent(g);

       g.drawRect(100,100, 100,200); // 100 x 200
       g.fillRect(300,100, 100,200);  // 100 x 200
   }
}
```

```
import javax.swing.*;
import java.awt.*; public class RectFrame
extends JFrame
 {
  RectFrame()
  {
   super ("Two Lines");
   setBounds(0,0,600,400);
   setBackground(Color.WHITE);
   JPanel p = new RectanglePanel();
   add(p);
   setVisible(true);
  }

    public static void main(String [] args)
    {
        JFrame frame = new RectFrame();
    }
}
```

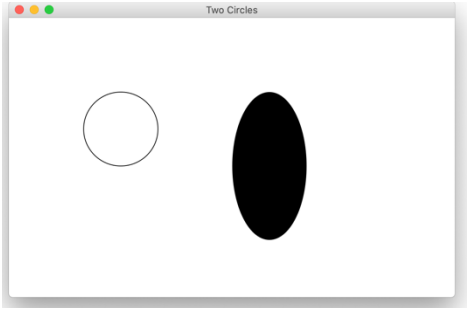
- **void drawOval(int x, int y , int width, int height)**
  draws an oval that fits within the boundary of the rectangle specified by the parameters x, y, width, and height. **If width and height are equal, the figure is a circle**.



An oval with bounding rectangle

- void fillOval(int x, int y, int width, int height)
  draws and fills the specified oval.

Here is a frame with a circle and an oval. Notice for the circle the width and height are the same.

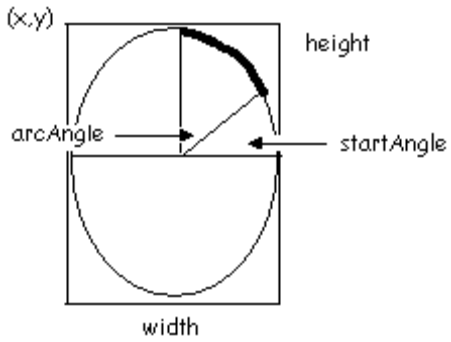


Here is the very simple code that draws these figures

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

public class CirclePanel extends JPanel
{
   // override paintComponent() of JPanel
    public void paintComponent(Graphics g)
    {
       // Call the paintComponent method of the parent
        super.paintComponent(g);

        g.drawOval(100,100, 100,100); // circle
        g.fillOval(300,100, 100,200);    // oval
    }
}
```

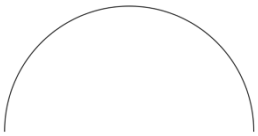
```
import javax.swing.*;
import java.awt.*;
public class CircleFrame extends JFrame
 {
  CircleFrame()
  {
   super ("Two Circles");
   setBounds(0,0,600,400);
   setBackground(Color.WHITE);
   JPanel p = new CirclePanel();
   add(p);
   setVisible(true);
  }

  public static void main(String [] args)
  {
     JFrame frame = new CircleFrame();
  }
}
```

- void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)
  draws an arc using the oval inscribed in the rectangle specified by parameters x, y ,width and height. The arc begins at startAngle and spans arcAngle. Angles are given in degrees.



The arc drawn by the *drawArc*() method.

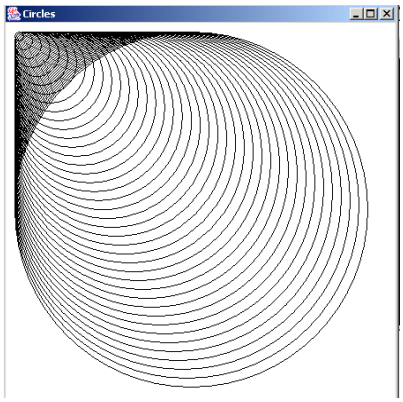
The statement drawArc(100,100,300,300, 0, 180) will draw a semi-circle



The top corner of the bounding rectangle is (100, 100), the width and height of the bounding rectangle are 300 and 300 (a circle) and the parameters 0 and 180 say the start angle is 0 and draw the circle for 180 degees.

Example

Design an application that draws the “megaphone of circles” in a frame."



The following application uses two classes:

- CirclesPanel extends JPanel and overrides paintComponent(Graphics g), and
- CircleFrame extends JFrame, instantiates CirclePanel, and adds a CirclesPanel object to the frame. The CircleFrame class includes a main(...) method.

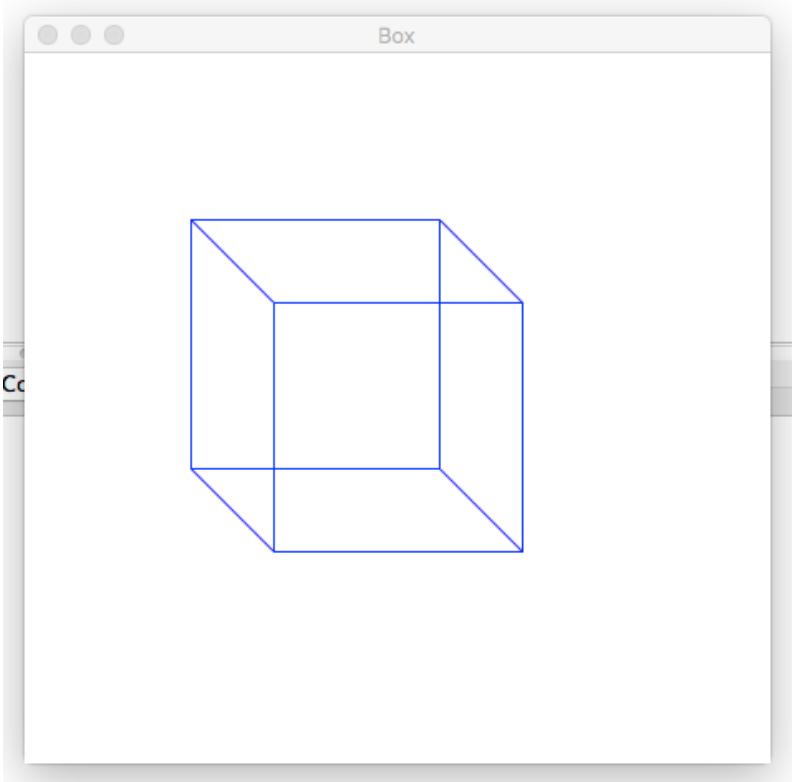
The CirclePanel class overrides paintComponent(Graphics g) of JPanel. Using a loop , it draws 40 circles. The first has diameter 400 pixels , the second 390, the third 380 etc. The upper left hand corner of the rectangle that defines the circle is (10,10)

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

3.  public class CirclePanel extends JPanel

4.          // Displays 39 circles.  The bounding rectangle for each circle is positioned at (10,10).
5.          // The circles range in diameter 10 to 400 pixels.
6.          // A frame size of at least 440 by 440 is used.

7.  {
8.      public void paintComponent(Graphics g)
9.      {
10.         super.paintComponent(g);// VERY IMPORTANT –Call the  paintComponent (..) of the parent
11.         setBackground(Color.white);
12.
13.          // Draw 40 circles  all positioned at (0,0)
            // the first  has diameter  400, the next 390, the next 380 etc
14.         // each time through the loop we decrease the diameter by 10
15.         // the upper corner of the bounding rectangle is always (10,10)
16.         for (int diameter = 400;  diameter >0; diameter-=10)          // draw 39 circles of decreasing diameter
17.             g.drawOval(10,10,diameter,diameter);
18.     }
19. }
```

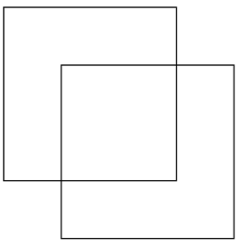
```
1.  public class CircleFrame extends JFrame
2.  {
3.      public CircleFrame(String title)
4.      {
5.          super(title);
6.          setBounds(0,0, 450, 450);
7.          JPanel circles = new CirclePanel();
8.          add(circles);
9.          setVisible(true);
10.         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11.     }

12.     public static void main(String[] args)
13.     {
14.         JFrame frame = new CircleFrame("Circles");
15.
16.     }
17. }
```

Example

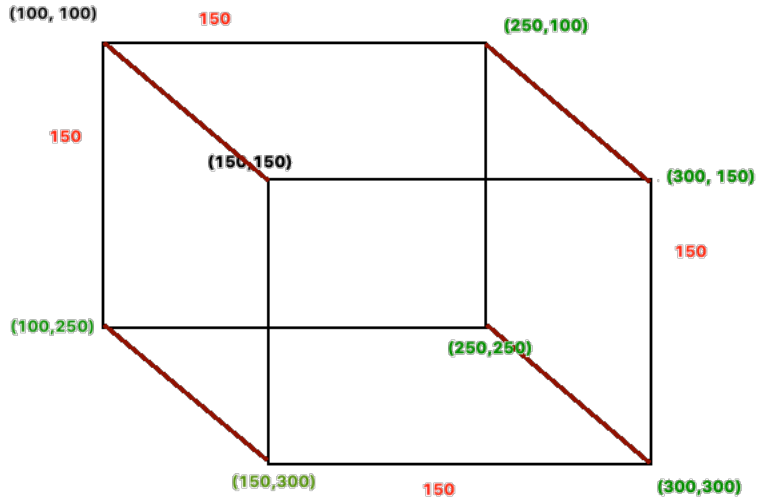
Extend JPanel and override paintComponent(Graphics g) so that paintComponent(…) draws a blue box on the panel. It does not have to be a perfect cube. Then in another class extend JFrame and add the panet to the extended JFrame. The frame should look something like this .



I will draw two squares and hen determine the coordinates of the corners and connect them with lines.



I first figured out the coordinates with paper and pencil.



Here is the code

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

public class CubePanel extends JPanel


{
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);
     // CALL THE paintComponent (..) of the parent

      g.setColor(Color.BLUE);


         g.drawRect(100,100,150,150);
         g.drawRect(150,150,150,150);

         g.drawLine(100,100,150,150);
         g.drawLine(100,250,150,300);
         g.drawLine(250,100,300,150);
         g.drawLine(250,250,300,300);

   }
}
```

```
import javax.swing.*;
import java.awt.*;
public class CubeFrame extends JFrame
{
    public CubeFrame()
    {
        super("Box");
        setBounds(0,0, 450, 450);
        JPanel cube = new CubePanel();
        setBackground(Color.WHITE);
        add(cube);
        setVisible(true);

    }

    public static void main(String[] args)
    {
        JFrame frame = new CubeFrame();

    }
}
```

The *repaint()* method

Question: Suppose we want to "repaint" the screen with something different but the system has no indication that it must repaint? Suppose have not moved, resized, minimized the frame but we want to change the contents???? The system does not call paint..) because there is no reason to do that. If we could call paint(..) that would do it…but we are not allowed to call paint(…)

Calls to paint(...) and paintComponent(...) are system generated.
When a component or its container is first displayed or subsequently resized or moved, the system automatically paints/repaints the component.

On the other hand, the JVM does not *always* know when a component needs to be redrawn. The programmer, in these cases, must take control and explicitly direct the application to repaint the component.

A program does not call paint(...) or paintComponent(...) to redisplay a component, but another method of the Component class:

void repaint().

The repaint() method, in turn, calls paint(...). And the system decides when paint(..) or paintComponent(..) will execute. It is like the garbage collector. The system decides when to run it. So a call to repaint(..) says "Hey System, the component needs to be repainted." And the system replies "OK I will do that, but when I feel like it doing it. On my time."

The following example uses repaint() to change a message displayed on a panel.

Example

Devise an application that paints a message on a panel, prompts (using a Scanner) for a new message, and repaints the panel showing the new message. The new message must be displayed.

The following class Message class extends JPanel and overrides paintComponent(...) so that a new version of paintComponent(…) paints a string on the panel.

The FrameWithAMessage class, which demonstrates Message,

- interactively prompts a user for a String, s
- interactively, reads the s, using the Scanner method, next(),
- sets the variable message to s
- calls repaint() to display the new message

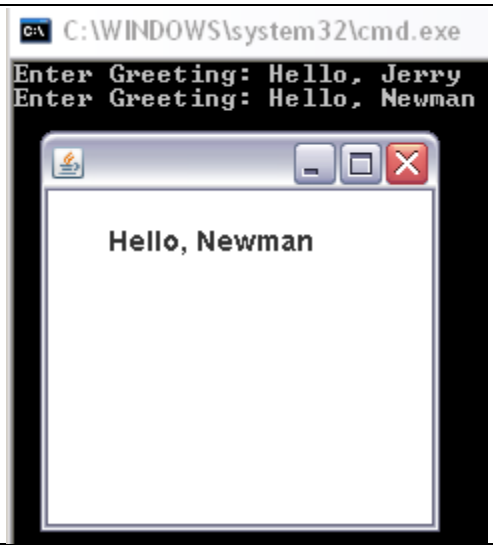
There are two classes
Message is a JPanel that extends paintComponent(..) to draw the message
FrameWithMessage is a JFrame that uses that panel

The bottom line: If you need the component redrawn you can call repaint(). You usually do not need to all repaint(). You call repaint only when, for example, you change a variable and want to redisplay the component with the new value. The system ha no indication that you want to repaint.

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

public class Message extends JPanel
{
    String message;
    public Message()
    {
       super();                                  // call the default constructor of JPanel
        message = "";
        setBackground(Color.WHITE);
    }

    public void paintComponent(Graphics g)       // override paintComponent(…) of JPanel
    {
        super.paintComponent(g);                 // first call paintComponent(...) of JPanel
        Font font = new Font("ARIAL",Font.BOLD,14);
        g.setFont(font);
        g.drawString (message, 30, 30);          // display the message
    }

    public void setMessage(String s)             // set the value of message
    {
            message = s;
    }
}
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
import java.util.*;                 // java.util.* is needed for Scanner
import javax.swing.*;                                   // java.awt is not necessary for this class

public class FrameWithAMessage
{
  public static void main(String[] args)
  {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter Greeting: ");
    String s = input.nextLine();

    JFrame  frame = new JFrame();          // create a frame
    frame.setBounds(0,0,200,200);
    Message panel = new Message();         // create a panel
    panel.setMessage(message);
    frame.add(panel);                                   // add the panel to the frame
    frame.setVisible(true);                             //  triggers system call to paintComponent(...)

    System.out.print("Enter Greeting: ");
    s = input.nextLine();                  // get a new message
    panel.setMessage(s);                // make the new message the panel's message

      panel.repaint();                              // repaint the panel with the new message

  }
}
```


C:\WINDOWS\system32\cmd.exe
Enter Greeting: Hello, Jerry
Enter Greeting:
Hello, Jerry


C:\WINDOWS\system32\cmd.exe
Enter Greeting: Hello, Jerry
Enter Greeting: Hello, Newman
Hello, Newman