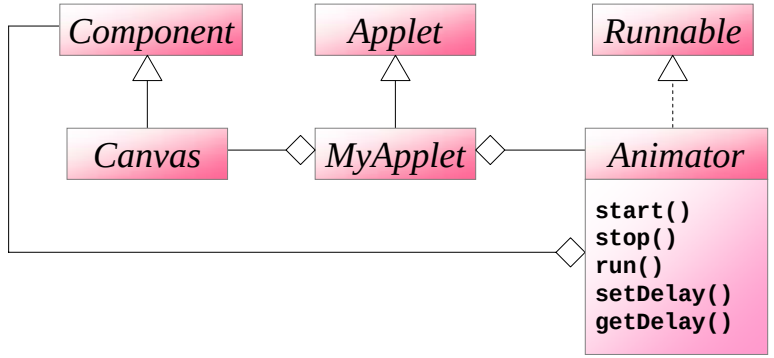
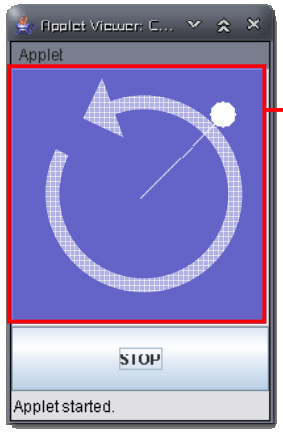
Reusable Animator

Delegating the animation task

Animator Delegation



Example



MyDrawing extends *JPanel*

Animator implements *Runnable*

Animator

```
import java.awt.*;
public class Animator implements Runnable{
	protected Component comp; // component to be animated
	protected int delay;
	protected Thread animationThread;
	public Animator(Component comp,int delay){
		this.comp = comp;
		setDelay(delay);
	}
	public void setDelay(int delay){
		this.delay = delay;
	}
	public int getDelay(){
		return delay;
	}
```

```
        public void startAnimate(){
                animationThread = new Thread(this);
                animationThread.start();
        }
        public void stopAnimate(){
                animationThread = null;
        }
        public void run(){
                while(Thread.currentThread()==animationThread){
                        try{
                                Thread.sleep(delay);
                        }catch(InterruptedException e){}
                        comp.repaint();
                }
        }
 }
```


Applet Subclass

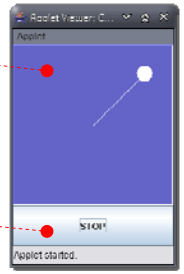
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
public class  CircularMotionApplet extends JApplet{
        … // Define attributes
        public void init(){
          … // Set GUIs including the component to be animated.
        }
        public void start(){
          animator.startAnimate();
        }
        public void stop(){
          animator.stopAnimate();
        }

        … // Class definitions of component to be animated.
        … // Other class definitions
}
```


init()

```
public void init(){
        Dimension dim = getSize();
        setLayout(null);
        canvas = new MyDrawing(10,Color.WHITE);
        canvas.setBounds(0,0,
                        dim.width,dim.height-CONTROL_HEIGHT);
        animator = new Animator(canvas,50);
        add(canvas);

        control = new MyControl();
        control.setBounds(0,
                        dim.height-CONTROL_HEIGHT,
                        dim.width,CONTROL_HEIGHT);
        add(control);
   }
```



Controlling Thread

```
public void start(){
        animator.startAnimate();
        stopped = false;
}
public void stop(){
        animator.stopAnimate();
        stopped = true;
}
```

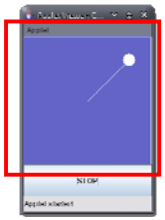
Component to be animated

```
class MyDrawing extends JPanel{
  int degree = 0, r, ballRadius,x,y,xC,yC;
  Dimension dim; Color ballColor;
  final int MARGIN = 10;
  final int DEGREE_INC = 5;

  public MyDrawing(int ballRadius,Color c){
       super(true); //double-buffering ON
       this.ballRadius = ballRadius;
       ballColor = c;
       setBackground(new Color(100,100,200));
       setBorder(
         BorderFactory.createBevelBorder(BevelBorder.RAISED,
         new Color(100,100,200),Color.LIGHT_GRAY));
  }
```



```
  public void paintComponent(Graphics g){
       super.paintComponent(g);
       dim = getSize(null);
       xC = dim.width/2;
       yC = dim.height/2;
       r = (int)Math.floor(Math.min(dim.width/2.0,
                           dim.height/2.0))-MARGIN;
       degree += DEGREE_INC;
       x = (int)Math.round(r*Math.sin(Math.PI*degree/180));
       y = (int)Math.round(r*Math.cos(Math.PI*degree/180));
       g.setColor(ballColor);
       g.fillOval(xC+x-ballRadius,yC+y-ballRadius,
                           ballRadius*2,ballRadius*2);
       g.drawLine(xC,yC,xC+x,yC+y);
  }
}
```

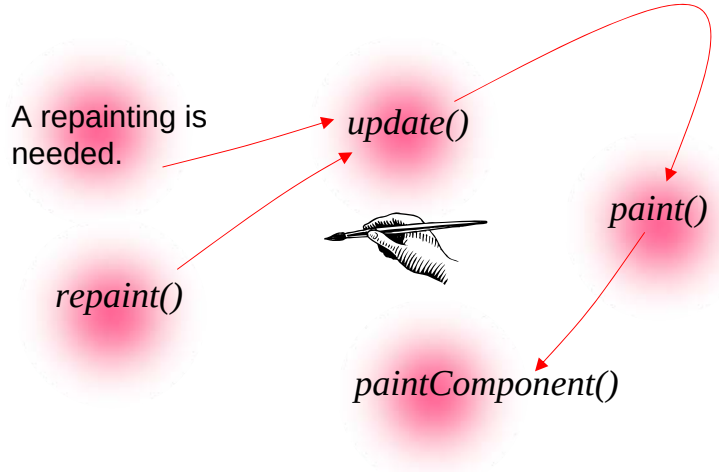


When we extend *JComponent* or *JPanel*, to draw something, override the *paintComponent()* method.

The Button

```
class MyControl extends JButton{
       public MyControl(){
              setText("STOP");
              addActionListener(new MyButtonHandler());
       }
}
class MyButtonHandler implements ActionListener{
       public void actionPerformed(ActionEvent e){
              if(stopped){
                     animator.startAnimate();
                     control.setText("STOP");
                     stopped = false;
              }else{
                     animator.stopAnimate();
                     control.setText("START");
                     stopped = true;
              }
       }
}
```


Painting of *JComponent*



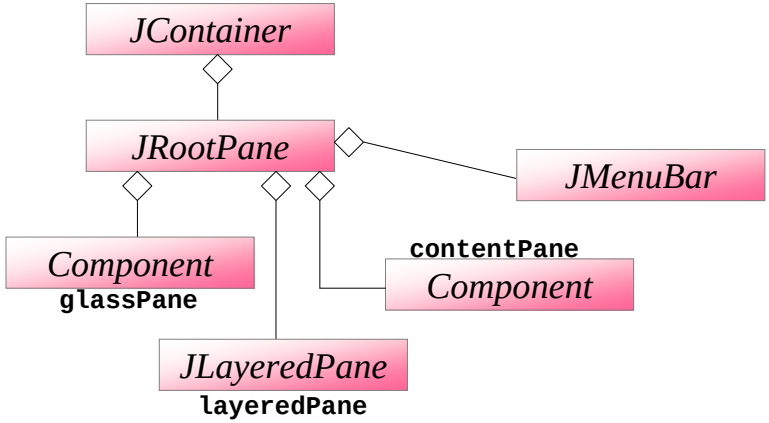
Swing Component

A Closer Look

Swing Container

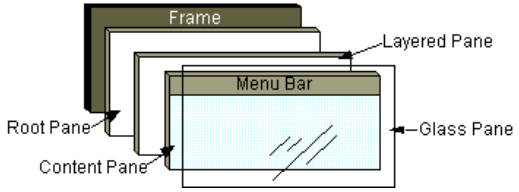
- Swing provides three top-level container classes: *JFrame*, *JDialog*, *JApplet*
- To appear onscreen, every GUI component must be part of a *containment hierarchy* (a tree of components that has a top-level container as its root).
- Each component *can be contained only once*. If a component is already in a container and is added to another container later, the component will be removed from the first container and then added to the second.
- When you create a top-level container, an associate *JRootPane* object comes with the container.

Elements of a Swing Container



JRootPane

- A *JRootPane* is made up of a *Component* object called `glassPane`, an *JLayeredPane* object called `layerPane`, another *Component* object called `contentPane`, and an optional *JMenuBar* object called `menuBar`.
- `contentPane` and `menuBar` are managed by `layerPane`.



The Content Pane

- holds components for the root pane.
- Adding components to the root pane (other than the optional menu bar), the components will be added to `contentPane` automatically.
- This is also true for setting layout managers.

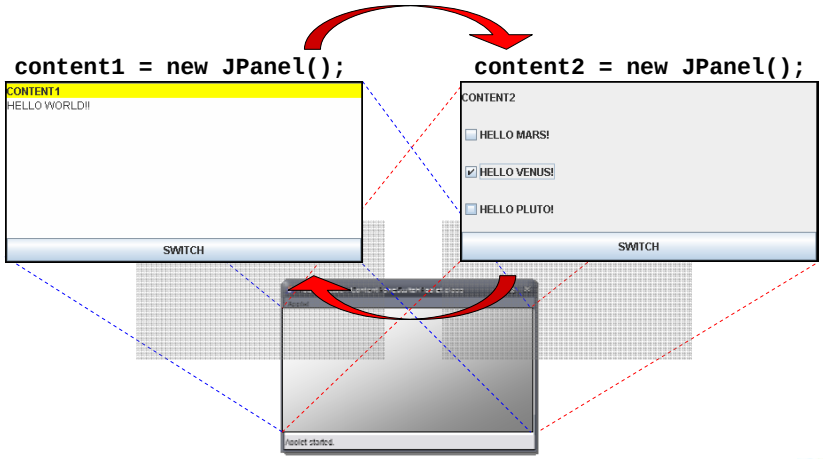
`this.getRootPane().getContentPane().add(`*[Some Component]***`);`**

an object of a top-level container

perform the same thing as

`this.add(`*[Some Component]***`);`**

Example: Switching Content Pane



Use *setContentPane()* to switch the content pane.


```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.applet.Applet;
public class ContentPaneSwitchApplet extends JApplet{
        JPanel content1,content2;
        JButton button1,button2;
        int currentID;
        public void init(){
                setContent1();
                setContent2();
                add(content1);
                currentID = 1;
        }
        public void setContent1(){
                …  // Set components on content1
        }
        public void setContent2(){

                … // Set components on content2
        }
```

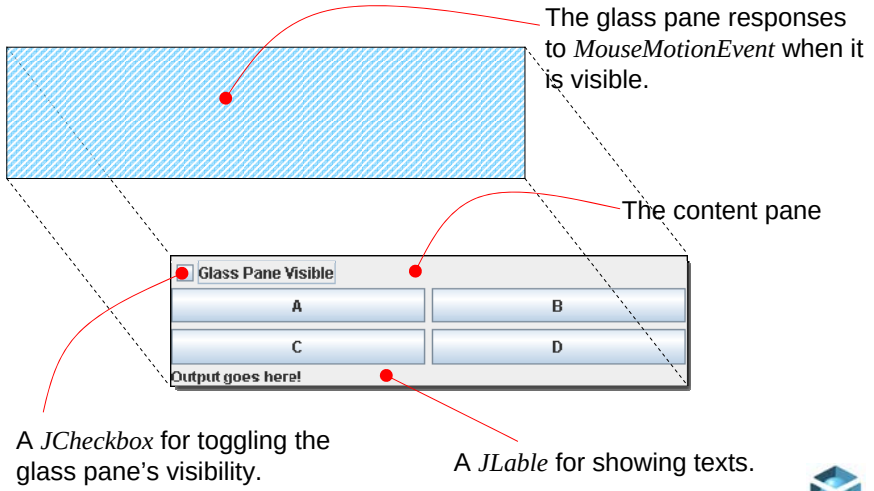

```
        class MyHandler implements ActionListener{
                public void actionPerformed(ActionEvent e){
                        if(currentID==1){
                                setContentPane(content2);
                                currentID = 2;
                        }else{
                                setContentPane(content1);
                                currentID = 1;
                        }
                        validate();
                }
        }
}
```

public void **validate()**
Validates this container and all of its subcomponents. The validate method is used to cause a container to lay out its subcomponents again. It should be invoked when this container's subcomponents are modified (added to or removed from the container, or layout-related information changed) after the container has been displayed.

The Glass Pane

- can be thought of *a sheet of glass that covers all area* of the root pane.
- By default, it is *completely transparent* unless *paintComponent()* is overridden.
- Can be used to intercept mouse events for the root pane.

Example: Turning on the Glass Pane




```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.applet.Applet;
import java.util.*;

public class TestGlassPaneApplet extends JApplet{

       JLabel output;
       JPanel buttonPad;
       JCheckBox glassPaneVisibility;

       public TestGlassPaneApplet(){
              setGUI();
              addListener();
```



```
       public void setGUI(){
              setLayout(new BorderLayout());
              glassPaneVisibility = new JCheckBox(
                                   "Glass Pane Visible");
              add(glassPaneVisibility,BorderLayout.NORTH);
              buttonPad = new JPanel();
              buttonPad.setLayout(new GridLayout(2,2,5,5));
              buttonPad.add(new JButton("A"));
              buttonPad.add(new JButton("B"));
              buttonPad.add(new JButton("C"));
              buttonPad.add(new JButton("D"));
              add(buttonPad,BorderLayout.CENTER);
              output = new JLabel("Output goes here!");
              output.setBackground(Color.YELLOW);
              add(output,BorderLayout.SOUTH);
       }
```


```
public void addListener(){
	EventListener handler = new MyHandler();
	glassPaneVisibility.addItemListener(
			(ItemListener)handler);
	getGlassPane().addMouseMotionListener(
			(MouseMotionListener)handler);
}

class MyHandler implements ItemListener, MouseMotionListener{
	public void itemStateChanged(ItemEvent e){
		if (e.getStateChange() == ItemEvent.SELECTED){
			getGlassPane().setVisible(true);
			output.setText("");
		}else{
			getGlassPane().setVisible(false);
			output.setText("");
		}
	}
```



```
		public void mouseMoved(MouseEvent e){
			output.setForeground(Color.BLACK);
			output.setText(e.getPoint().getX()+
					","+e.getPoint().getY());
		}
		public void mouseDragged(MouseEvent e){
			output.setForeground(Color.RED);
			output.setText(e.getPoint().getX()+
					","+e.getPoint().getY());
		}
	}
}
```

When the glass pane is visible, it covers the entire area of the content pane, and all components on the content pane are not accessible by the user.

Problems??

Useful Swing Utilities

static Point convertPoint(Component source, int x, int y, Component destination)

Convert the point (x,y) in source coordinate system to destination coordinate system.

static Component getDeepestComponentAt(Component parent, int x, int y)

Returns the deepest visible descendent Component of parent that contains the location x, y.

Accessing Components Below the Glass Pane

```
public void mouseReleased(MouseEvent e){
	Point glassPanePoint = e.getPoint();
	Component glassPane = getGlassPane();
	Component contentPane = getContentPane();
	Point contentPanePoint = SwingUtilities.convertPoint(
					glassPane,
					glassPanePoint,
					contentPane);
	Component component = SwingUtilities.getDeepestComponentAt(
					contentPane,
					contentPanePoint.x,
					contentPanePoint.y);
	if ((component != null)&&
			(component.equals(glassPaneVisibility))){
		getGlassPane().setVisible(false);
		output.setText("Hiding the glass pane.");
	}
}
```