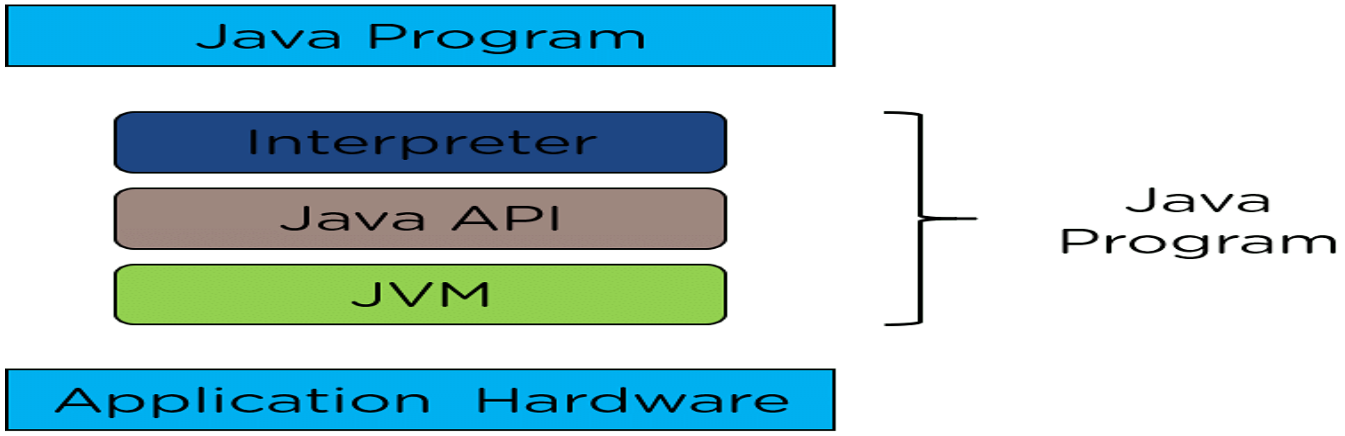
Graphical User Interfaces (GUIs) with Swing, Applet and Graphics

Java GUI API Basics

- ✓ GUI (Graphical User Interface) actually represents that you see on your desktop which interacts with the user using a Graphical interface (it has buttons, menu bar, or anything you can imagine).
- ✓ Using GUI makes it easy for the user to use the feature of an application.
- ✓ In Java, you can create your own desktop applications using two GUI APIs i.e. AWT (abstract window toolkit & Swing (its better than AWT)) .
- ✓ API stands for Application Programming Interface that enable two software components to communicate with each other using a set of definitions and protocols.
- ✓ In java APIs are important software components bundled with the JDK. APIs in Java include classes, interfaces, and user Interfaces.
- ✓ For example, the weather bureau's software system contains daily weather data. The weather app on your phone “talks” to this system via APIs and shows you daily weather updates on your phone.

Java GUI API Basics

- ✓ Three sets of Java APIs for graphics programming: AWT, Swing and JavaFX.
- ✓ The fundamental components of the Java API are follows:



Advantages of GUI:

- ✓ The graphical User Interface is visually very appealing and detailed oriented.
- ✓ It ensures that people with little or even no knowledge of computers can use it and perform basic computer functions.
- ✓ Graphical User Interface is easy to use since it does not require the user to use any command.

Java GUI History

Abstract Windowing Toolkit (**AWT**): Sun's initial effort to create a set of cross-platform GUI classes. *(JDK 1.0 - 1.1)*

- Maps general Java code to each operating system's real GUI system.
- *Problems:* Limited to lowest common Component class and Container ; heavy to use.

Swing: A newer GUI library written from the ground up that allows much more powerful graphics and GUI construction. *(JDK 1.2+)*

- Paints GUI controls itself pixel-by-pixel rather than handing off to OS.
- *Benefits:* Features; compatibility; OO design.

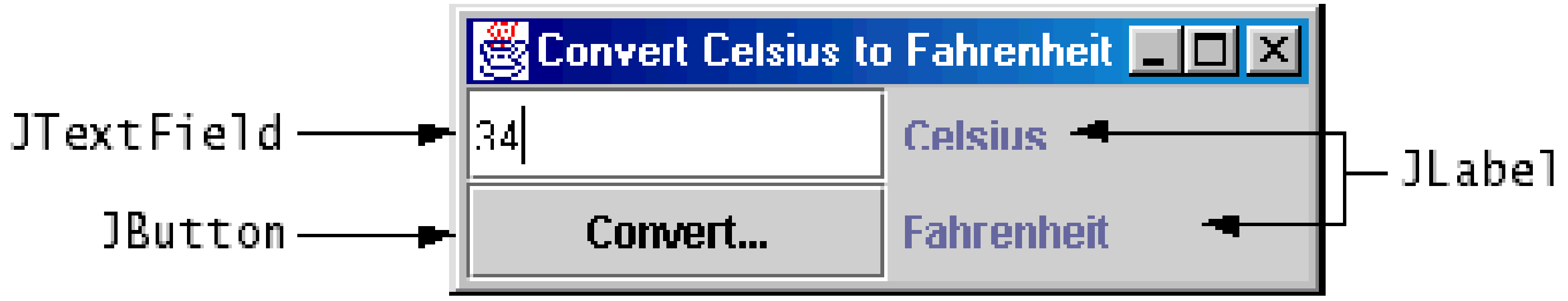
- *Problem:* Both exist in Java now; easy to get them mixed up; still have to use both in various places.

GUI terminology

window: A first-class citizen of the graphical desktop.
Also called a *top-level container*.
examples: frame, dialog box, applet

component: A GUI widget that resides in a window.
Also called *controls* in many other languages.
examples: button, text box, label

container: A logical grouping for storing components.
examples: panel, box



AWT vs. Swing

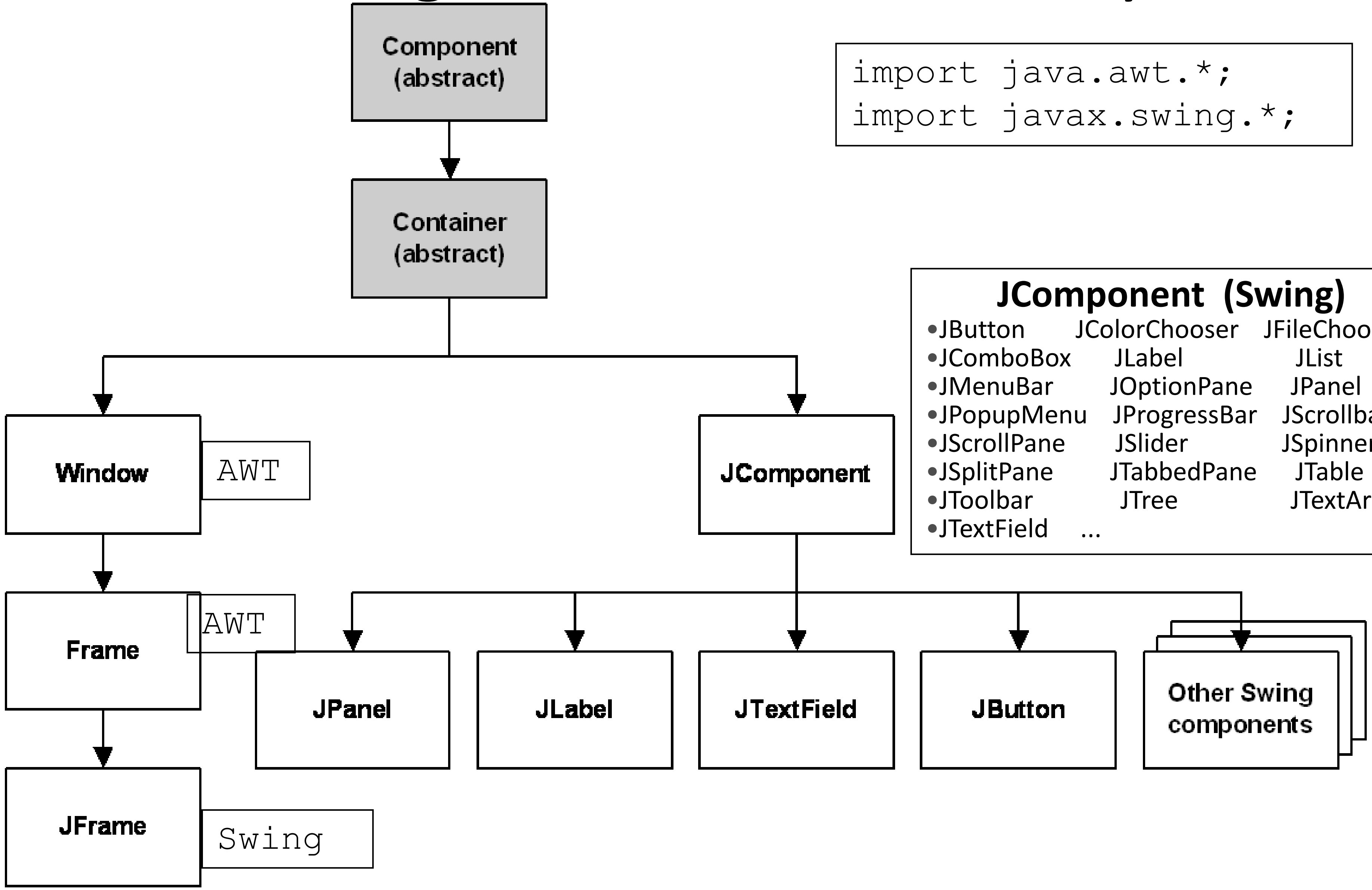
Abstract Windowing Toolkit (AWT)

- Original Java GUI toolkit
- Wrapper API for native GUI components
- Lowest-common denominator for all Java host environments

Swing

- Implemented entirely in Java on top of AWT
- Richer set of GUI components
- More light-weight compared to AWT

Swing inheritance hierarchy



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

JComponent (Swing)

•JButton	JColorChooser	JFileChooser
•JComboBox	JLabel	JList
•JMenuBar	JOptionPane	JPanel
•JPopupMenu	JProgressBar	JScrollbar
•JScrollPane	JSlider	JSpinner
•JSplitPane	JTabbedPane	JTable
•JToolbar	JTree	JTextArea
•JTextField	...	

Swing inheritance hierarchy

A summary of the classes in the Component hierarchy

Class	Description
Component	An abstract base class that defines any object that can be displayed.
Container	An abstract class that defines any component that can contain other components.
Window	The AWT class that defines a window without a title bar or border.
Frame	The AWT class that defines a window with a title bar and border.
JFrame	The Swing class that defines a window with a title bar and border.
JComponent	A base class for Swing components such as JPanel, JButton, JLabel, and JTextField.
JPanel	The Swing class that defines a panel, which is used to hold other components.
JLabel	The Swing class that defines a label.
JTextField	The Swing class that defines a text field.
JButton	The Swing class that defines a button.

Swing Design Principles

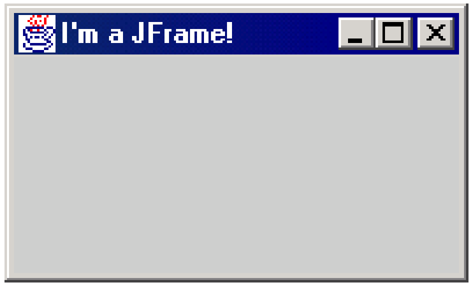
- GUI is built as **containment hierarchy** of widgets (container) (i.e. the parent-child nesting relation between them)
- Event objects and event listeners
  - **Event object**: is created when event occurs (e.g. click), contains additional info (e.g. mouse coordinates)
  - **Event listener**: object implementing an interface with an event handler method that gets an event object as argument
- Separation of Model and View:
  - **Model**: the data that is presented by a widget
  - **View**: the actual presentation on the screen

Component properties

- **Each has a** `get` (or `is`) accessor and a `set` modifier method.
- examples: `getColor, setFont, setEnabled, isVisible`

name	type	description
background	`Color`	background color behind component
border	`Border`	border line around component
enabled	`boolean`	whether it can be interacted with
focusable	`boolean`	whether key text can be typed on it
font	`Font`	font used for text in component
foreground	`Color`	foreground color of component
height, width	`int`	component's current size in pixels
visible	`boolean`	whether component can be seen
tooltip text	`String`	text shown when hovering mouse
size, minimum / maximum / preferred size	`Dimension`	various sizes, size limits, or desired sizes that the component may take

JFrame



❖ **GUI using Swing**

a graphical window to hold other components

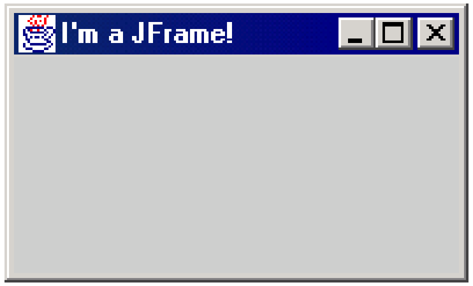
- `public JFrame()`
  `public JFrame(String title)`
  Creates a frame with an optional title.

  – Call `setVisible(true)` to make a frame appear on the screen after creating it.

- `public void add(Component comp)`
  Places the given component or container inside the frame.

More Jframe…

❖ **GUI using Swing**



- `public void setDefaultCloseOperation(int op)`
  Makes the frame perform the given action when it closes.
  - Common value passed: `JFrame.EXIT_ON_CLOSE`
  - If not set, the program will never exit even if the frame is closed.
- `public void setSize(int width, int height)`
  Gives the frame a fixed size in pixels.
- `public void pack()`
  Resizes the frame to fit the components inside it snugly/nicely.

JButton

❖ **GUI using Swing**

a clickable region for causing actions to occur

- `public JButton(String text)`
  Creates a new button with the given string as its text.

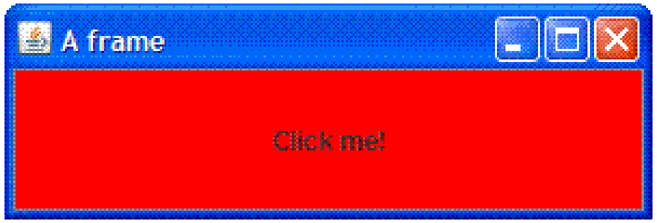
- `public String getText()`
  Returns the text showing on the button.

- `public void setText(String text)`
  Sets button's text to be the given string.

GUI Example

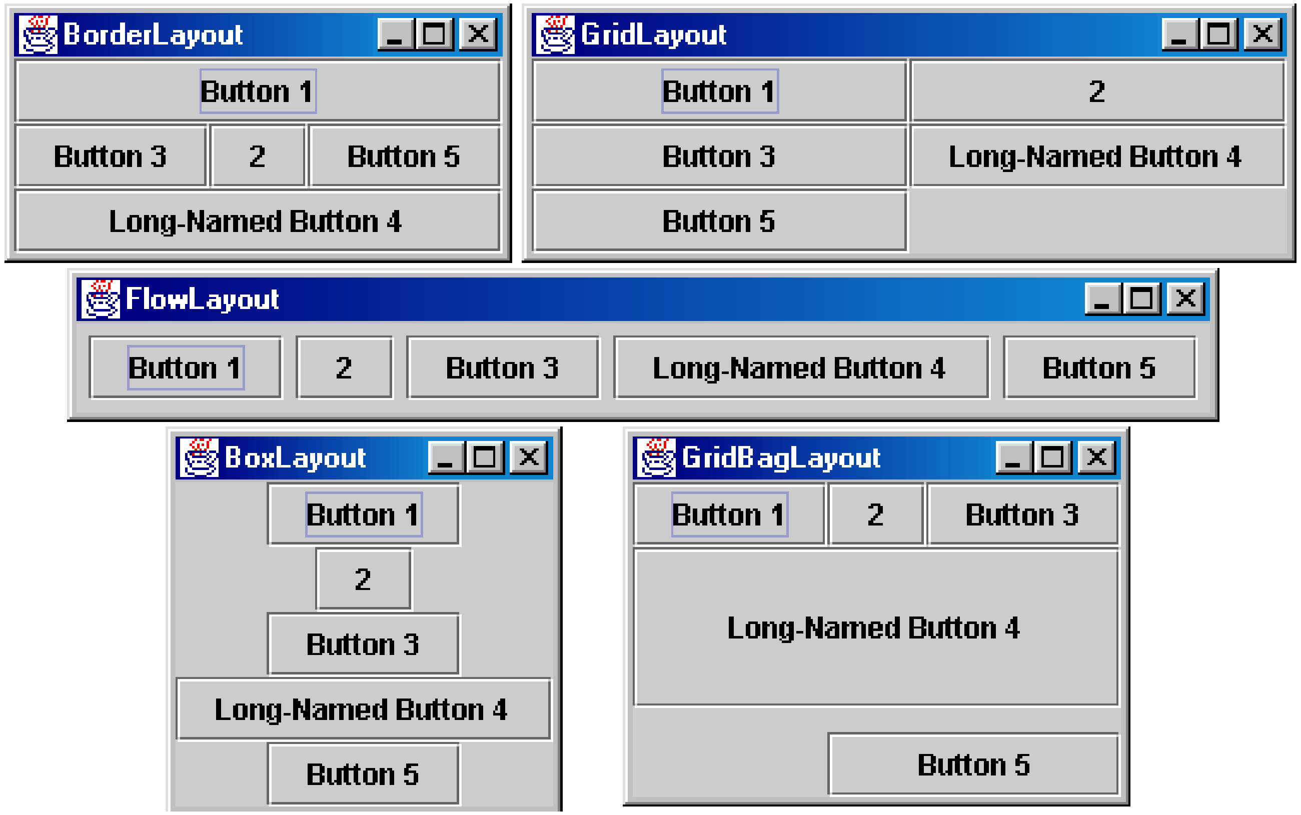
❖ **GUI using Swing**

```
import java.awt.*;        // other button if needed
import javax.swing.*;

public class GuiExample1 {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(new Dimension(300, 100));
        frame.setTitle("A frame");

        JButton button1 = new JButton();
        button1.setText("I'm a button.");
        button1.setBackground(Color.BLUE);
        frame.add(button1);

        JButton button2 = new JButton();
        button2.setText("Click me!");
        button2.setBackground(Color.RED);
        frame.add(button2);

        frame.setVisible(true);
    }
}
```



Containers and layout

- Place components in a *container*; add the container to a frame.
  - **container**: An object that stores components and governs their positions, sizes, and resizing behavior.



JFrame as container

A `JFrame` is a container. Containers have these methods:

- `public void` **`add`**`(Component comp)`
  `public void` **`add`**`(Component comp, Object info)`
  Adds a component to the container, possibly giving extra information about where to place it.

- `public void` **`remove`**`(Component comp)`

- `public void` **`setLayout`**`(LayoutManager mgr)`
  Uses the given layout manager to position components.

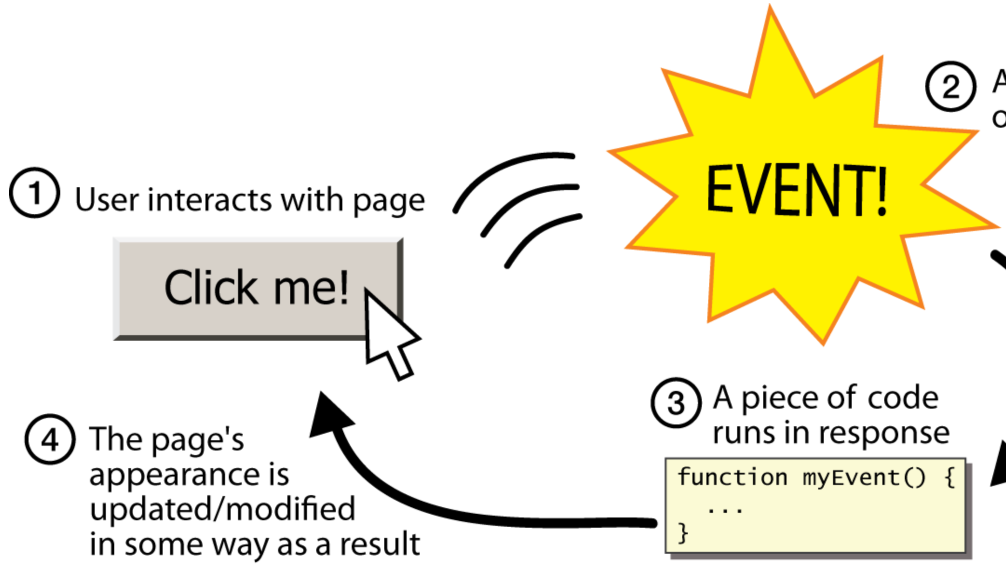
- `public void` **`validate`**`()`
  Refreshes the layout (if it changes after the container is onscreen).

Event Listeners

Graphical events

- **event**: An object that represents a user's interaction with a GUI component; can be "handled" to create interactive components.

- **listener**: An object that waits for events and responds to them.
  - To handle an event, attach a *listener* to a component.
  - The listener will be notified when the event occurs (e.g. button click).



1 User interacts with page
Click me!
EVENT!
2 An "event" occurs
3 A piece of code runs in response
4 The page's appearance is updated/modified in some way as a result

```
function myEvent() {
  ...
}
```

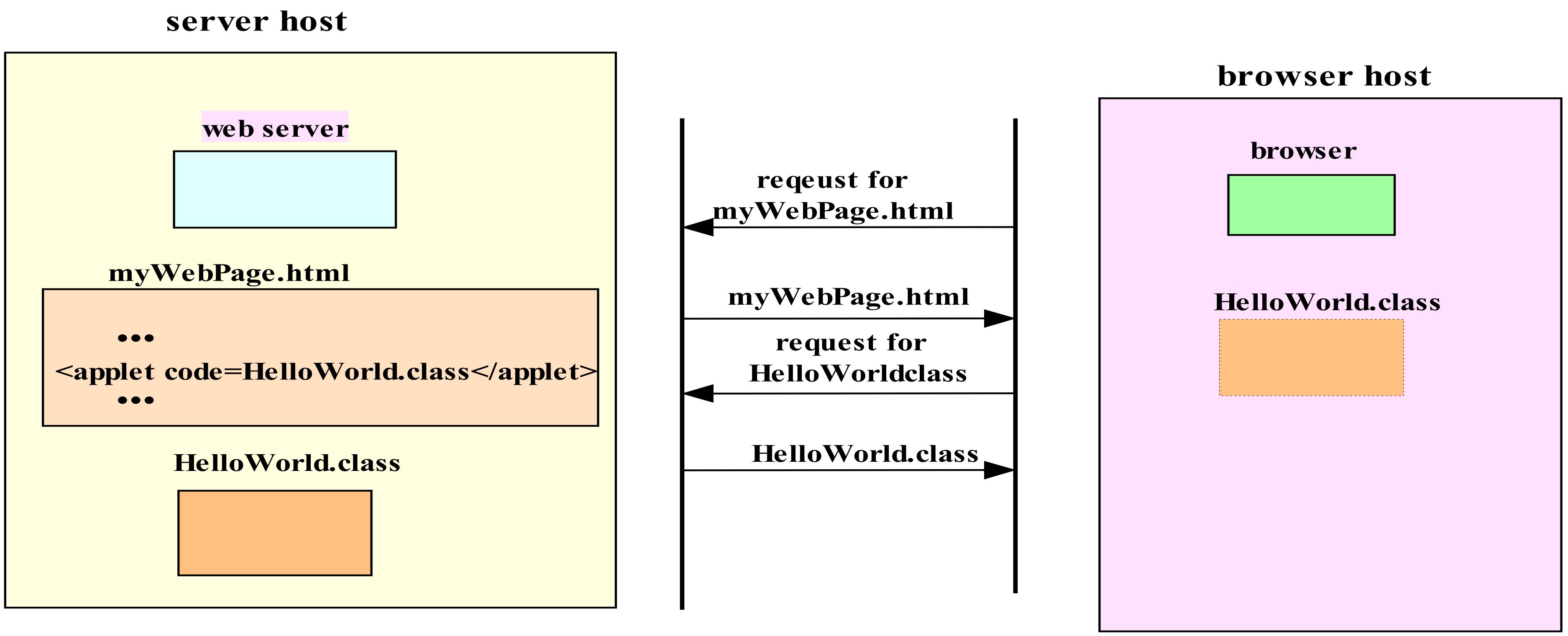
Java Applet

Java applets are one of three kinds of Java programs:

- An ***application*** is a standalone program that can be invoked from the command line.
- An ***applet*** is a program that runs in the context of a browser session.
- A ***servlet*** is a program that is invoked on a server program, and it runs in the context of a web server process.

More Applet….

- Applets are programs stored on a web server, similar to web pages.
- When an applet is referred to in a web page that has been fetched and processed by a browser, the browser generates a request to fetch (or download) the applet program, then executes the applet program in the browser's execution context on the client host.



Applets, web page, client, server

Applet Execution and Security

- **An applet program is a written as a subclass of the java.Applet class or the javax.swing.Japplet class.**
- **There is no main() method in an Applet.**
- **An applet uses AWT for graphics, or JApplet, a subclass of javax.swing.**

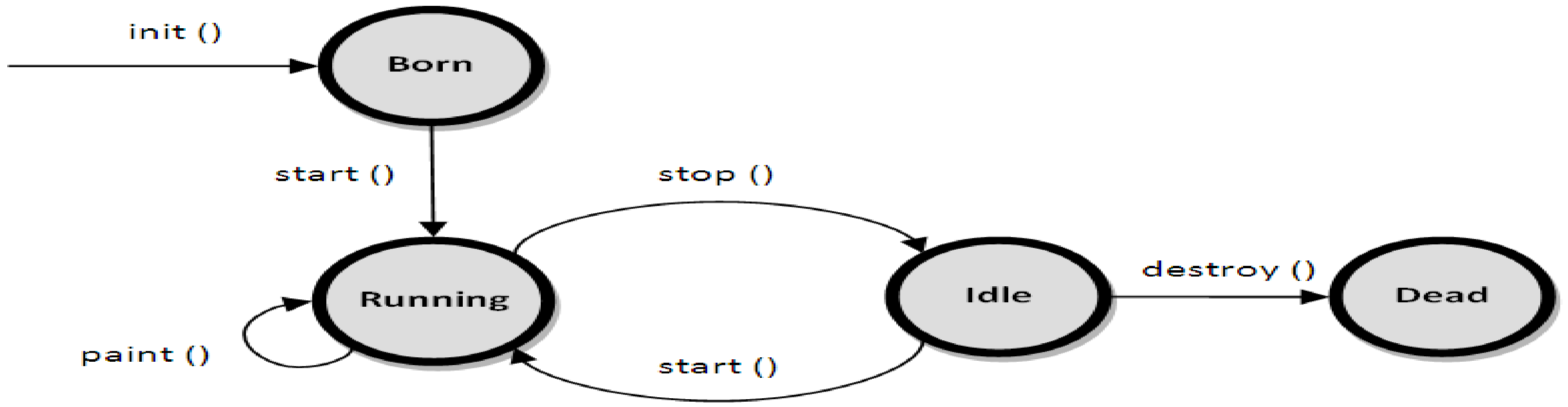
- **Applet Security**

For security reasons, applets that are loaded over the network have several restrictions.

- ✓ an applet **cannot** ordinarily **read** or **write** files on the computer that it's executing on.
- ✓ an applet **cannot** **make** network connections except to the **host** that it came from.

Applet Life Cycle

- **init()**: This method is intended for whatever initialization is needed for an applet.
- **start()**: This method is automatically called after init method. It is also called whenever user returns to the page containing the applet after visiting other pages.
- **stop()**: This method is automatically called whenever the user moves away from the page containing applets. This method can be used to stop an animation.
- **destroy()**: This method is only called when the browser shuts down normally.
- **paint()**: This method is used to draw shapes like circle, square, trapezium, etc., in the applet. It is executed after the start() method and when the browser or applet windows are resized.



More Applet Life Cycle….

- **The applet is running and rendered on the web page.**
- **Every Applet needs to implement one of more of the init(), the start( ) and the paint( ) methods.**
- **At the end of the execution, the stop( ) method is invoked, followed by the destroy( ) method to deallocate the applet's resources.**

HTML tags for applets

<APPLET
  // the beginning of the HTML applet code
    CODE="demoxx.class"
    // the actual name of the applet (usually a 'class' file)
    CODEBASE="demos/"
    // the location of the applet (relative as here, or a full URL)
      NAME=“SWE622"
      // the name of the instance of the applet on this page
      WIDTH="100"
      // the physical width of the applet on the page
      HEIGHT="50"
      // the physical height of the applet on the page
      ALIGN="Top" // align the applet within its page space (top, bottom, center)

```
<APPLET CODE=“SWE622.class" CODEBASE="example/"
                    WIDTH=460 HEIGHT=160
                                  NAME="buddy" >
<PARAM NAME="imageSource" VALUE="images/Beans">
<PARAM NAME="backgroundColor" VALUE="0xc0c0c0"> <PARAM NAME="endImage"
VALUE=10>
</APPLET>
```

The HelloWorld Applet Example

```
<HTML>
<BODY>
<APPLET code=hello.class width=900 height=300>
</APPLET>
</BODY>
</HTML>
```

```
// applet to display a message in a window
import java.awt.*;
import java.applet.*;

public class hello extends Applet {
    public void init( ) {
        setBackground(Color.yellow);
    } // end of init()

    public void paint(Graphics g) {
        final int FONT_SIZE = 42;
        Font font = new Font("Serif",
            Font.BOLD, FONT_SIZE);
// set font, and color and display message
// on the screen at position 250,150
            g.setFont(font);
            g.setColor(Color.blue);
// The message in the next line is the one
// you will see
            g.drawString("Hello,
                world!",250,150);
    } // end of paint()

} // end of hello
```

Graphics

❖ Drawing on Your Own

- ✓ Sometimes you want to have more control than just using the GUI components provided by the Java libraries. In these situations you might want to have custom control over what gets drawn to the space inhabited by a given component.
- ✓ The power and quality of this was enhanced with the Graphics 2D library in Swing.

❖ Overriding the paint Method

- ✓ The way that you can control what is drawn on a component is to override the paint method of that component (for AWT use **paint** and Swing use **paintComponent**). This means that you need to create a subclass of a component. I typically do custom components from JPanels.
- ✓ The paint method takes a **java.awt.Graphics** object. What you draw with that object shows up on the Component.

AWT Graphics

- ✓ Graphics is an abstract class provided by Java AWT which is used to draw or paint on the components.
- ✓ It consists of various fields which hold information like components to be painted, font, color, XOR mode, etc., and methods that allow drawing various shapes on the GUI components.
- ✓ Graphics is an abstract class and thus cannot be initialized directly.
- ✓ Objects of its child classes can be obtained in the following two ways:
  1. **Inside paint() or update() method** (paint() and update() methods are present in the Component class and thus can be overridden for the component to be painted. Example: void paint(Graphics g), void update(Graphics g).
  2. **getGraphics() method** (This method is present in the Component class and thus can be called on any Component in order to get the Graphics object for the component.)

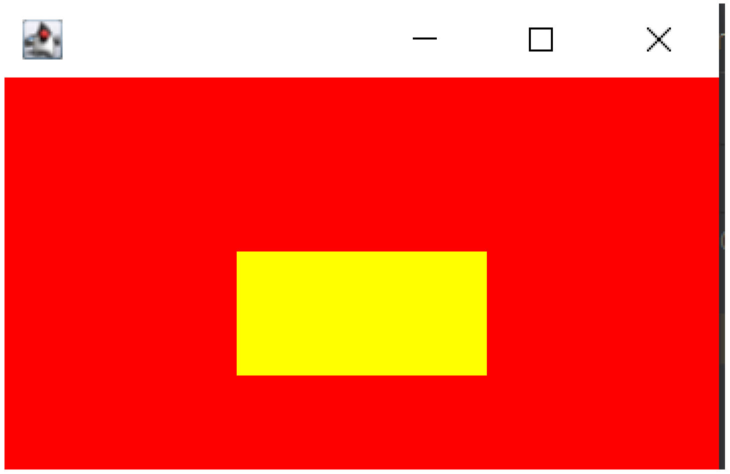
AWT Graphics Example

Use paint() Method:

```
import java.awt.*;
public class Myframe extends Frame {
   public Myframe()
   {
      setVisible(true);
      setSize(300, 200);
      setBackground(Color.red);
//       addWindowListener(new WindowAdapter() {
//           public void windowClosing(WindowEvent e)
//           { System.exit(0);   }  });
   }
   public void paint(Graphics g)
   {
      g.setColor(Color.green);
      g.setXORMode(Color.black);
      g.fillRect(100, 100, 100, 50);
 }
   public static void main(String[] args)
   {
      new Myframe(); }   }
```

Output:

Rectangle Shape



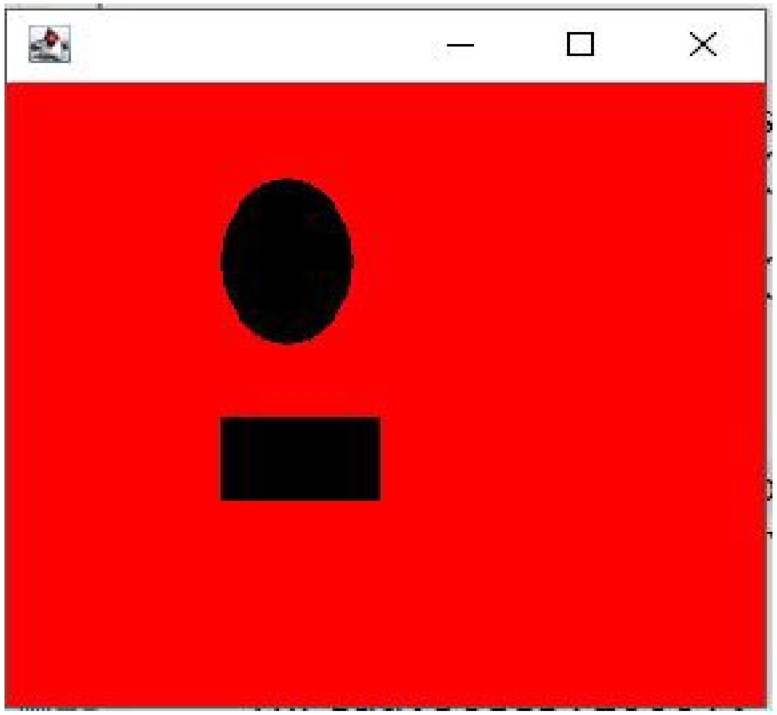
AWT Graphics Example

Use getGraphics() Method:

```
import java.awt.*;
import javax.swing.JFrame;
public class GetGraphics {
public static void main(String[] args) {
JFrame panel = new JFrame();
panel.setSize(300,300);
panel.setVisible(true);
Graphics g = panel.getGraphics();
 try   //without try cath output will not shows
{
Thread.sleep(1000);
} catch(Exception e)
{System.out.println(e);
}
g.fillRect(10, 30, 60, 35);
g.fillOval(80, 40, 50, 70);
}}
```

Output:

Rectangle and Oval Shape

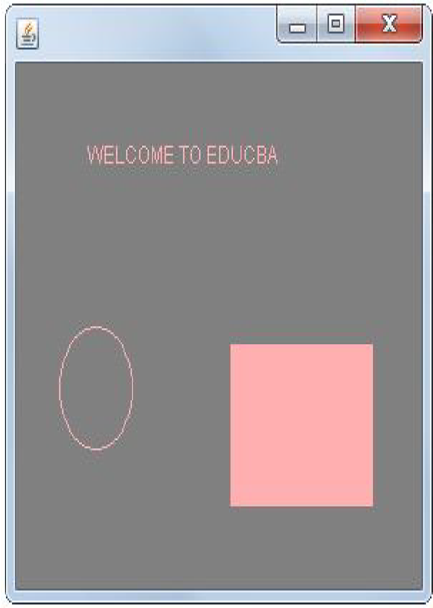


Swing Graphics Example

```
import java.awt.*;
import javax.swing.JFrame;
public class GraphicsDemo extends Canvas { //or use JPanel
public void paint(Graphics graphics) {
        //adding the string to graphics
        graphics.drawString("WELCOME TO ICE", 50, 50);
        setBackground(Color.GRAY); //background color
        graphics.fillRect(150, 140, 100, 81); //rectangle shape
        graphics.drawOval(30, 131, 51, 61); //oval shape
        setForeground(Color.pink); //setting object color
            }
public static void main(String[] args) {
     GraphicsDemo graphicsDemo = new GraphicsDemo();
     JFrame jFrame = new JFrame(); //creating frame object
     jFrame.add(graphicsDemo); //adding graphics to the frame
     jFrame.setSize(300, 300);   // f.setLayout(null);
     jFrame.setVisible(true);
     } }
```

Output:

Drawing Rectangle and Oval shape


WELCOME TO EDUCBA