# CSC 143 Java

## Introduction to Graphical Interfaces in Java: Swing and AWT



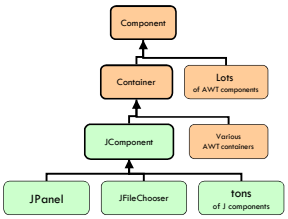
## Opposing Styles of Interaction

- "Algorithm-Driven"
  - When *program* needs information from user, it asks for it
  - Program is in control
  - Typical in non-GUI environments (examples: payroll, batch simulations)
- "Event Driven"
  - When *user* wants to do something, he/she signals to the program
    Moves or clicks mouse, types, etc.
  - These signals come to the program as "events"
  - Program is interrupted to deal with the events
  - User has more control
  - Typical in GUI environments



## Components & Containers

- Every GUI-related component descends from Component, which contains dozens of basic methods and fields common to all AWT/Swing component
- "Atomic" components: labels, text fields, buttons, check boxes, icons, menu items, …
- Some components are Containers – components like panels that can contain other nested subcomponents




Naming: Most Swing components start with J (no such standard for AWT components).



## Types of Containers

- Top-level containers: JFrame, JDialog, JApplet
  - Often correspond to OS Windows
  - An Applet is a special kind of container – we won't cover this
- Mid-level containers: panels, scroll panes, tool bars, …
  - can contain certain other components
  - JPanel is best for general use
- Specialized containers: menus, list boxes, combo boxes...
- Technically, all J components are containers

http://java.sun.com/docs/books/tutorial/ui/features/components.html



## JFrame – A Top-Level Window

- Top-level application window

```
JFrame win = new JFrame( "Optional Window Title" );
```

- Some common methods

```
setDefaultCloseOperation( … );   // specify action to be taken when closed
                                 //   setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE)
getContentPane( … ) …            // add components – see ahead…
pack( );                         // lay out components and resize window to fit
setVisible( boolean v );         // show or hide
```

---

```
repaint( );                      // request repaint after content change
setSize( int width, int height );// set size for window; it is more common to set
                                 // a preferred size for components, then pack
setBackground( Color c );        // set background color
```



## JPanel – A General Purpose Container

- Commonly added to a window to provide a space for graphics, or collections of buttons, labels, etc.
- JPanels can be nested to any depth
- Many methods in common with JFrame (since both are ultimately instances of Component)

```
setBackground( Color c );
setPreferredSize( Dimension d );
repaint( );
```

  - Advice: Can't find the method you're looking for?
    Check the superclass.
- Only visible when added to a top-level container

## Adding Components to Containers

- Top-level swing containers have a "content pane" that manages the components in that container
  [Differs from original AWT containers, which managed their components directly]
- To add a component to a top-level container, get the content pane, and use its add method

```
JFrame jf = new JFrame( );
JPanel panel = new JPanel( );
…
jf.getContentPane( ).add(panel);
   or
Container cp = jf.getContentPane( );
cp.add( panel );
```

- Can add directly to a JPanel
  `panel.add(new JButton("on"));`



## Non-Component Classes

- Not all AWT/Swing classes are components
- AWT
  - Color, Dimension, Font, layout managers
  - Shape and subclasses like Rectangle, Point, etc.
  - Graphics
- Swing
  - Borders
  - Further geometric classes
  - Graphics2D
- Neither AWT nor Swing
  - Images, Icons



## Layout Managers

- What happens if we add several components to a container?
  - What are their relative positions?
- Answer: each container has a *layout manager*. Some kinds:
  - FlowLayout (left to right, top to bottom)
  - BorderLayout("center", "north", "south", "east", "west")
  - GridLayout (2-D grid); others
    http://java.sun.com/docs/books/tutorial/uiswing/layout/visual.html
- Default layout managers:
  - JFrame's content pane: BorderLayout
  - JPanel: FlowLayout



## *pack* and *validate*

- Container state is "valid" or "invalid" depending on whether layout manager has arranged components since last change.
- When a container is altered, either by adding components or changes to components (resized, contents change, etc.), the layout needs to be updated (i.e., the container state needs to be set to valid).
  - Swing does this automatically more often than AWT, but not always
- Common methods after changing layout
  - validate( ) – redo the layout to take into account new or changed components
  - pack( ) – redo the layout using the preferred size of each component



## Layout Example

- Create a JFrame with a label at the bottom and a panel in the center (don't forget imports…)

```
JFrame frame = new JFrame( "Trivial Window" ); //default layout: Border
frame.setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );
JPanel panel = new JPanel( );
panel.setPreferredSize( new Dimension( 250, 250 ) );
JLabel label = new JLabel( "Smile!" );
label.setHorizontalAlignment( SwingConstants.CENTER );
Container cp = frame.getContentPane( );
cp.add( panel, BorderLayout.CENTER );
cp.add( label, BorderLayout.SOUTH );
frame.pack( );
frame.setVisible( true );
```



## Graphics and Drawing

- Simple things like labels have suitable default code to paint themselves
- For more complex graphics, extend a suitable class and override the inherited method *paintComponent* to draws its contents
  (Different from AWT, where you override *paint* – don't do that in swing!)

```
public class Drawing extends JPanel {
  ...
    /** Repaint this Drawing whenever requested by the system */
    public void paintComponent( Graphics g ) {
      super.paintComponent( g );
      g.setColor( Color.green );
      g.drawOval( 40, 30, 100, 100 );
      g.setColor( Color.red );
      g.fillRect( 60, 50, 60, 60 );
    }
```

## *paintComponent*

- Method paintComponent is called by the underlying system *whenever it needs the window to be repainted*
  - Triggered by window being move, resized, uncovered, expanded from icon, etc.
  - Can happen anytime – you don't control when
- If your code does something that requires repainting, call method repaint( )
  - Requests that paintComponent be called sometime in the future, when convenient for underlying system window manager



## Painter's Rules

- *Always* override paintComponent( ) of any component you will be drawing on
  - But not necessary if you make simple changes, like changing background color, title, etc. that don't require a graphics object
- *Always* call super.paintComponent(g) to paint the background
- *Never* call paint( ) or paintComponent( ). Never means never!
  - This is a hard rule to understand. Follow it anyway.
- *Always* paint the entire picture, from scratch
- Don't create a Graphics object to draw with
  - only use the one given to you as a parameter of paintComponent( )
  - and, don't save that object to reuse later!
  - This rule is bent in advanced graphics applications



## What Happens If You Don't Follow The Rules...



## Classes Graphics and Graphics2D

- The parameter to *paintComponent* or *paint* is a graphics context where the drawing should be done
  - In Swing components, the parameter has static type Graphics, but dynamic type Graphics2D, a subclass of Graphics
    Cast it to Graphics2D if you want to use the newer, more sophisticated graphics operations like image rotation
- *Graphics* has a more procedural-like interface than many graphics packages used with introductory textbooks
  - Call Graphics methods to draw on the Graphics object
    [instead of creating new shape objects and adding them to the window]