

CSE 143 Java

Introduction to Graphical Interfaces in Java: AWT and Swing

Reading: Sec. 17.1-17.3, 17.7

8/8/2004

(c) 2001-4, University of Washington

06-1

Overview

- Roadmap
 - Today: introduction to Java windows and graphical output
 - Future: event-driven programs and user interaction
- Topics
 - A bit of history: AWT and Swing
 - Some basic Swing components: JFrame and JPanel
 - Java graphics
- Reading:
 - Textbook: Ch. 17
 - Online: Sun Java Swing tutorial (particularly good for picking up details of particular parts of Swing/AWT as needed); Swing API javadoc web pages <http://java.sun.com/docs/books/tutorial/uiswing/index.html>

8/8/2004

(c) 2001-4, University of Washington

06-2

Graphical User Interfaces

- GUIs are a hallmark of modern software
- Hardly existed outside research labs until Mac's came along
 - Picked up by PC's later
- User sees and interacts with "controls" or "components" (sometimes called "widgets")
 - menus
 - scrollbars
 - text boxes
 - check boxes
 - buttons
 - radio button groups
 - graphics panels
 - etc. etc.

8/8/2004

(c) 2001-4, University of Washington

06-3

Opposing Styles of Interaction

- | | |
|--|--|
| <ul style="list-style-type: none">• "Algorithm-Driven"<ul style="list-style-type: none">• When <i>program</i> needs information from user, it asks for it• Program is in control• Typical in non-GUI environments (examples: payroll, batch simulations) | <ul style="list-style-type: none">• "Event Driven"<ul style="list-style-type: none">• When <i>user</i> wants to do something, he/she signals to the program<ul style="list-style-type: none">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 |
|--|--|

8/8/2004

(c) 2001-4, University of Washington

06-4

A Bit of Java History

- Java 1.0: **AWT** (Abstract Windowing Toolkit)
- Java 1.1: AWT with new event handling model
- Java 1.2 (aka Java 2): **Swing**
 - Greatly enhanced user interface toolkit built on top of AWT
 - Same basic event handling model as in Java 1.1 AWT
- Java 1.3, 1.4
 - Bug fixes and significant performance improvements; no major revolution
- **Naming**
 - Most Swing components start with J.
 - No such standard for AWT components

8/8/2004

(c) 2001-4, University of Washington

06-5

Bit o' Advice

- Use Swing whenever you can
- Use AWT whenever you have to
 1. For classes not redefined in Swing, such as Color and Dimension
 2. For superclasses of Swing components
 3. When writing code that must be compatible for older version of Java -- obsolete browsers, etc.



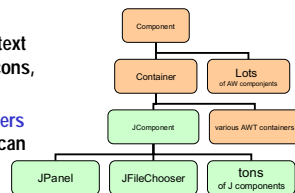
8/8/2004

(c) 2001-4, University of Washington

06-6

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



8/8/2004

(c) 2001-4, University of Washington

06-7

Types of Containers

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

8/8/2004

(c) 2001-4, University of Washington

06-8

JFrame – A Top-Level Window

- **Top-level application window**

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

- **Some common methods**

```
setBackground(Color c);    // background color
pack()                     // lay out components
setVisible(true);          // make visible -- or use show();
repaint();                 // request repaint after content change
setSize(int w, int h);     // default size for window; also can set min
                           // and max sizes
dispose();                 // get rid of the window when done
```

8/8/2004

(c) 2001-4, University of Washington

06-9

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); //DON'T use setSize()!
repaint()
```

- **Advice: Can't find the method you're looking for?**

Check the superclass.



8/8/2004

(c) 2001-4, University of Washington

06-10

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);
```

8/8/2004

(c) 2001-4, University of Washington

06-11

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

8/8/2004

(c) 2001-4, University of Washington

06-12

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")
 - BoxLayout(one vertical stack or one horizontal row)
 - GridLayout (2-D grid)
 - GridBagLayout (makes HTML tables look simple!); others
- Default LayoutManager for JFrame is BorderLayout
- Default for JPanel is FlowLayout

8/8/2004

(c) 2001-4, University of Washington

06-13

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

8/8/2004

(c) 2001-4, University of Washington

06-14

Layout Example

- Create a JFrame with a button at the bottom and a panel in the center

```
JFrame frame = new JFrame("Trivial Window"); //default layout: Border
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);
```

8/8/2004

(c) 2001-4, University of Washington

06-15

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); //don't forget this step!
        g.setColor(Color.green);
        g.drawOval(40,30,100,100);
        g.setColor(Color.red);
        g.fillRect(60, 50, 60, 60);
    }
}
```

8/8/2004

(c) 2001-4, University of Washington

06-16

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

8/8/2004

(c) 2001-4, University of Washington

06-17



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

8/8/2004

(c) 2001-4, University of Washington

06-18

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



8/8/2004

(c) 2001-4, University of Washington

06-19

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 `uwcse.GWindow` (if you've used that)
 - Call `Graphics` methods to draw on the `Graphics` object
 - [instead of creating new shape objects and adding them to the window]

8/8/2004

(c) 2001-4, University of Washington

06-20

Graphics 2D

- In the Graphics 2D package, many graphical objects implement the Shape interface
 - When possible, chose a Shape rather than a non-Shape
- Shapes contain their own location and size
- Shapes do not contain their own color
- Lots of methods available to draw various kinds of outline and solid shapes and control colors and fonts

8/8/2004

(c) 2001-4, University of Washington

06-21

Graphics2D Survival Kit

- In reading and experimenting, focus on these classes:
 - JPanel (and ancestors)
 - (interface) Shape
 - Line2D/Ellipse2D/Rectangle2D
 - Polygon
 - Graphics2D, especially these methods:
 - draw(Shape)
 - fill(Shape)
 - drawString(String, int, int)
 - setColor(Color)
- Avoid methods like drawLine, drawPolygon, etc.

8/8/2004

(c) 2001-4, University of Washington

06-22

Roadmap

- Future: Events
 - User interaction
 - GUI components
- What to do
 - Start reading textbook chs. 17 and 18
 - Browse the Swing tutorial and Java Swing/AWT documentation from Sun to start to feel your way around

8/8/2004

(c) 2001-4, University of Washington

06-23