
CSE 331

Software Design & Implementation

Kevin Zatloukal
Summer 2017
Java GUIs

(Based on slides by Mike Ernst, Dan Grossman, David Notkin, Hal Perkins, Zach Tatlock)

Reminders

- HW8 due today
- Section tomorrow on HW9 and Android / Swing
 - install Android Studio beforehand if you plan to use Android
- Quiz 6 due Friday

Review

- Event-driven program is one whose main loop waits for an event and then processes it (over and over until quit time)
 - this sort of loop is called an event loop
- Examples of event-driven programs:
 - (web) servers
 - GUIs
- Technicalities:
 - OSes only let you wait for certain types of events at once
 - work around it by having another thread list for other types
 - (but be careful about what work is done on which thread)

Java AWT / Swing

References on Java AWT / Swing

Very useful start: Sun/Oracle Java tutorials

- <http://docs.oracle.com/javase/tutorial/uiswing/index.html>

Mike Hoton's slides/sample code from CSE 331 Sp12 (lectures 23, 24 with more extensive widget examples)

- <http://courses.cs.washington.edu/courses/cse331/12sp/lectures/lect23-GUI.pdf>
- <http://courses.cs.washington.edu/courses/cse331/12sp/lectures/lect24-Graphics.pdf>
- <http://courses.cs.washington.edu/courses/cse331/12sp/lectures/lect23-GUI-code.zip>
- <http://courses.cs.washington.edu/courses/cse331/12sp/lectures/lect24-Graphics-code.zip>

Good book that covers this (and much more):

Core Java vol. I by Horstmann & Cornell

- there are other decent Java books out there too

What not to do...

- Don't try to learn the whole library: there's way **too much**
- Don't memorize – look things up as you need them
- Don't miss the main ideas & fundamental concepts
- Don't get bogged down implementing eye candy for HW9
 - (unless you finish everything else)

A very short history (1)

Java's standard libraries have supported GUIs from the beginning

Original Java GUI: [AWT](#) (Abstract Window Toolkit)

- mapped Java UI to host system UI widgets
- limited set of user interface elements (widgets)
 - lowest common denominator

Advantage: looks native

Disadvantage: “write once, debug everywhere”

A very short history (2)

Swing: newer GUI library, introduced with Java 2 (1998)

Basic idea: underlying system provides only a blank window

- Swing draws all UI components directly
- doesn't use underlying system widgets
- (built on top of parts of AWT)

Advantage: **should** work the same on all platforms

- less testing work in principle (but be skeptical of that claim)

Disadvantage: doesn't look like a native GUI for that OS

A very short history (3)

SWT: improved version of AWT approach (2004?)

- tries to expose all the functionality of native GUIs
- Eclipse is built using SWT
- not part of the standard Java library

Two choices:

1. Use Swing to make a GUI that looks / works consistently
2. Use SWT to make a native-looking GUI on each platform

Option 1 is less work.

Option 2 usually makes users happier.

We'll cover Swing since it's standard Java...

A very short history (4)

Android: platform for writing phone/tablet apps with Java

- not part of the standard Java library
- open source project from Google

Conceptually similar to AWT/Swing

- but Android devices should look and behave similarly

Unfortunately cannot reuse AWT/Swing code

Main topics to learn

Using AWT/Swing components (a.k.a. widgets):

- different types of components
- how to lay them out in a window
- how to handle widget events

Writing your own components (Thursday section):

- how to draw your own UI
- how to handle lower level events

GUI terminology

window: A first-class citizen of the graphical desktop

- also called a *top-level container*
- Examples: *frame* (window), dialog box

component: A GUI *widget* that resides in a window







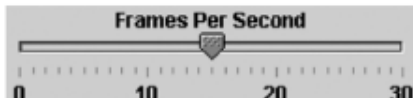

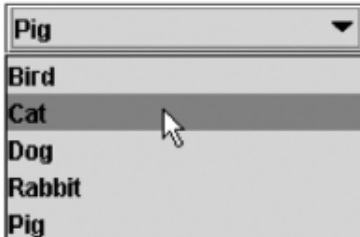

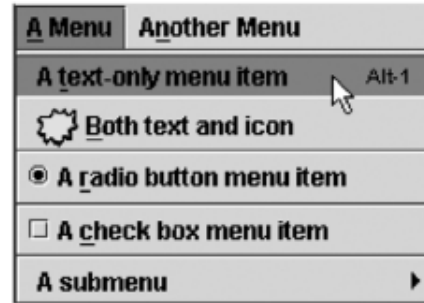
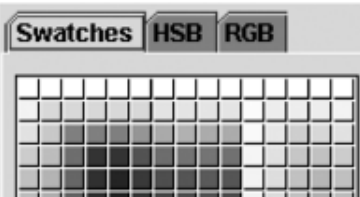
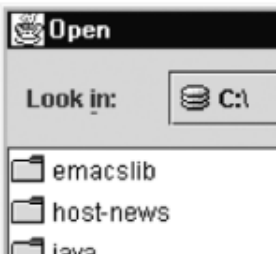




- called *controls* in many other languages
- Examples: button, text box, label

container: A component that hosts (holds) components

- Examples: frame, *panel*, box

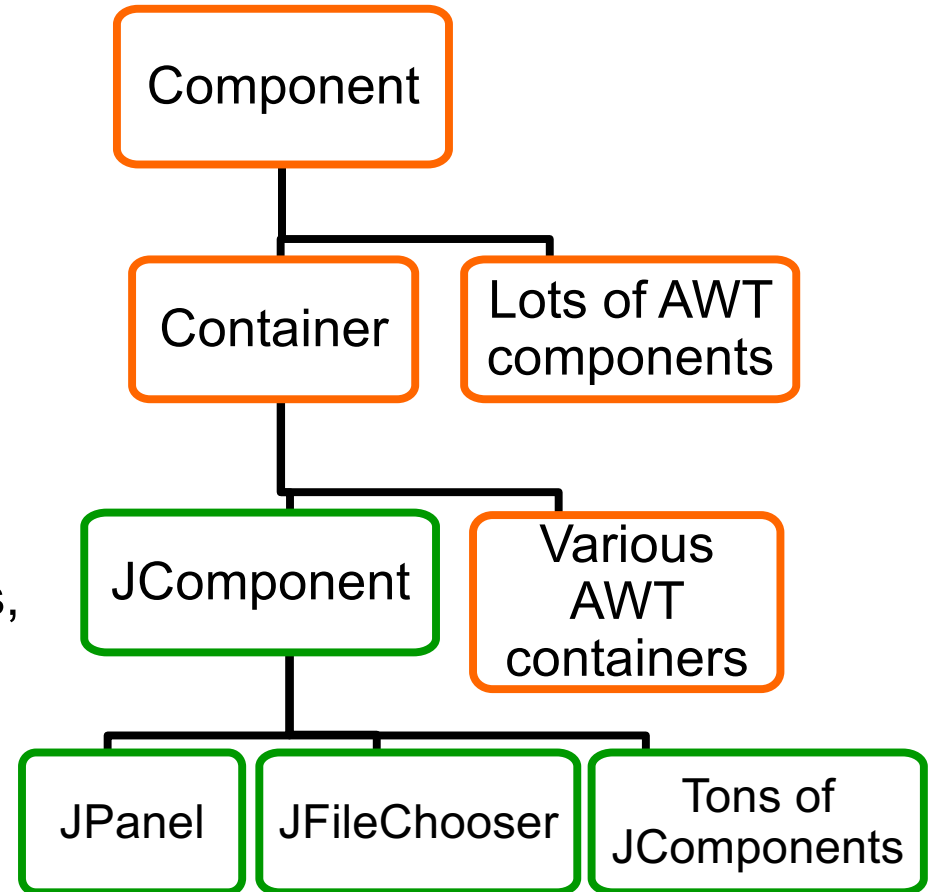


Some components...

JButton 	JCheckBox 	JRadioButton 	 Image and Text  Text-Only Label																		
JTextField 	JSlider 	JToolBar 																			
JComboBox 	JList 	JMenuBar, JMenu, JMenuItem 																			
JColorChooser 	JFileChooser 	JTable <table><thead><tr><th>First Name</th><th>Last Name</th><th>Favorite F</th></tr></thead><tbody><tr><td>Jeff</td><td>Dinkins</td><td></td></tr><tr><td>Ewan</td><td>Dinkins</td><td></td></tr><tr><td>Amy</td><td>Fowler</td><td></td></tr><tr><td>Hania</td><td>Gajewska</td><td></td></tr><tr><td>David</td><td>Gearv</td><td></td></tr></tbody></table>	First Name	Last Name	Favorite F	Jeff	Dinkins		Ewan	Dinkins		Amy	Fowler		Hania	Gajewska		David	Gearv		JTree 
First Name	Last Name	Favorite F																			
Jeff	Dinkins																				
Ewan	Dinkins																				
Amy	Fowler																				
Hania	Gajewska																				
David	Gearv																				

Component and container classes

- Every GUI-related class descends from **Component**, which contains dozens of basic methods and fields
 - Examples: **getBounds**, **isVisible**, **setForeground**, ...
- “Atomic” components: labels, text fields, buttons, check boxes, icons, menu items...
- Many components are **containers** – things like panels (**JPanel**) that can hold nested subcomponents



Swing/AWT inheritance hierarchy

Component (AWT)

Window

Frame

JFrame (Swing)

JDialog

Container

JComponent (Swing)

JButton

JComboBox

JMenuBar

JPopupMenu

JScrollPane

JSplitPane

JToolBar

TextField

JColorChooser

JLabel

JOptionPane

JProgressBar

JSlider

JTabbedPane

JTree

...

JFileChooser

JList

JPanel

JScrollbar

JSpinner

JTable

JTextArea

Component properties

Zillions. Each has a **get** (or **is**) accessor and a **set** modifier.

Examples: **getColor**, **setFont**, **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

Types of containers

- Top-level containers: **JFrame**, **JDialog**, ...
 - usually correspond to OS windows
 - a “host” for other components
 - live at top of UI hierarchy, not nested in anything else
- Mid-level containers: panels, scroll panes, tool bars
 - sometimes contain other containers, sometimes not
 - **JPanel** is a general-purpose component for drawing or hosting other UI elements (buttons, etc.)
- Specialized containers: menus, list boxes, ...

JFrame – top-level window

- Graphical window on the screen
- Holds other components
- Common methods:
 - `JFrame(String title)`: constructor, title optional
 - `setDefaultCloseOperation(int what)`
 - What to do on window close
 - `JFrame.EXIT_ON_CLOSE` terminates application
 - `setSize(int width, int height)`: set size
 - `setVisible(boolean b)`: make window visible or not

Example

`SimpleFrameMain.java`

JFrame – top-level window

- Graphical window on the screen
- Holds other components
- Common methods:
 - `JFrame(String title)`: constructor, title optional
 - `setDefaultCloseOperation(int what)`
 - What to do on window close
 - `JFrame.EXIT_ON_CLOSE` terminates application
 - `setSize(int width, int height)`: set size
 - `setVisible(boolean b)`: make window visible or not
 - `add(Component c)`: add component to window

Example

`SimpleButtonDemo.java`

Where is the event loop?

GUIs are event-driven programs, so where is the event loop?

- It is created automatically by Swing
 - presumably when we call `frame.setVisible(true)`
- The main method actually returns...
- Swing creates another thread to run the GUI event loop
 - this is called the UI thread
 - the Java VM does not quit the program until *all threads* exit

Example

`SimpleButtonDemo2.java`

JPanel – a general-purpose container

- Commonly used to hold a collection of button, labels, etc.
 - (also has another use you will learn about in section)
- Needs to be added to a window or other container:
`frame.add(new JPanel (...))`
- **JPanels** can be nested to any depth
- Many methods/fields in common with **JFrame** (since both inherit from **Component**)
 - Can't find a method/field? Check the superclasses.

A particularly useful method:

- `setPreferredSize(Dimension d)`

Example

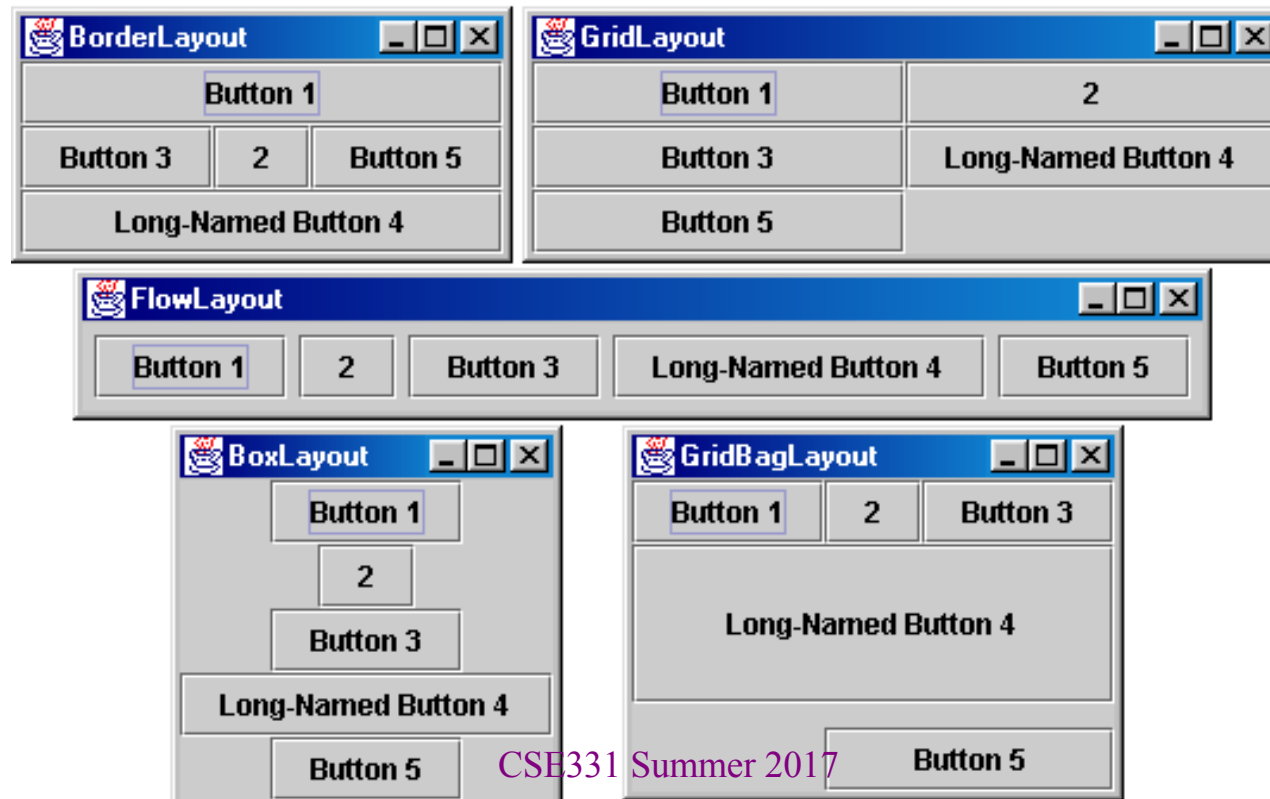
`SimpleButtonDemo3.java`

Example

`SimpleFieldDemo.java`

Containers and layout

- What if we add several components to a container?
 - How are they positioned relative to each other?
- Answer: each container has a *layout manger*



Layout managers

Kinds:

- **FlowLayout** (left to right [changeable], top to bottom)
 - Default for **JPanel**
 - Each row centered horizontally [changeable]
- **BorderLayout** (“center”, “north”, “south”, “east”, “west”)
 - Default for **JFrame**
 - No more than one component in each of 5 regions
 - (Of course, component can itself be a container)
- **GridLayout** (regular 2-D grid)
- Others... (Some are incredibly complex. None are perfect.)

Layout managers

You can change the layout manager on any `JComponent c`

- `c.setLayout(new GridLayout())`

`FlowLayout` and `BorderLayout` are likely good enough for now...

(There are similar issues creating UI in HTML...)

Example

`SimpleFieldDemo2.java`

Example

SimpleFieldDemo3.java

pack ()

Instead of having the components lay out within the window size, you can instead size the window to fit the components:

```
frame.pack () ;
```

pack () figures out the sizes of all components and calls the container's layout manager to set locations in the container
– (recursively as needed)

Example

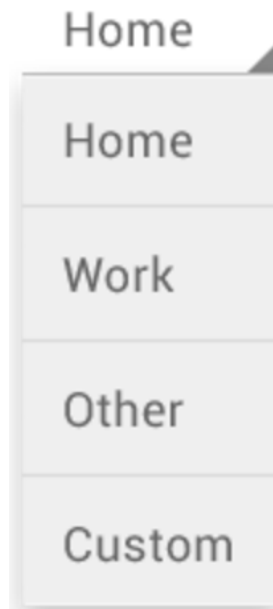
`SimpleFieldDemo4.java`

Android

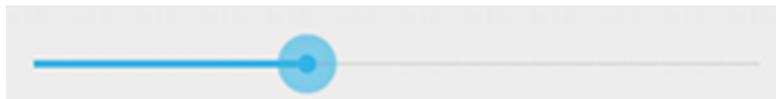
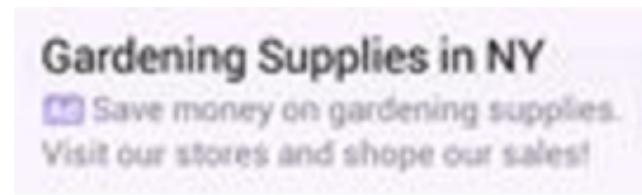
Components

Many of the same ones
But some new ones

- spinner
- seek bar
- rating bar
- calendar view
- ad view
- ...



February 2014						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	



Containers

Components are subclasses of View

Containers are subclasses of ViewGroup

Commonly used containers:

- linear layout (horizontal or vertical)
- positions children relative to others (e.g., above, to right, centered)
- grid
- list view

(Ideally, you would skip this and layout at fixed positions.)

Latter two can be easily used to display data

– (see HW9)

Activities

Android uses a model similar to a web browser:

- each page is called an “activity”
- back button takes you back to the previous activity

Each app creates one or more activities

- main activity is (normally) started when the app starts
- `startActivity(this, OtherActivity.class)` starts another activity

Activity is notified when it is in use

- `onCreate` called to create the UI
- `onStop` called when it is no longer visible
- `onDestroy` called when it is destroyed