WEEK7A

JAVA II — BARRY

TOPICS

- Q & A
- RoadMap
- Upcoming project
- Event-driven Programming
- Java Swing
  - Creating simple applications
  - Using common widgets and supplied dialogs
  - Drawing graphics

ROADMAP

WHERE WE ARE & WHAT'S COMING UP

ROADMAP

#	Week	Topics/Reading	Read	Proj Due	Exam
7	5/15	Events, Swing Observer Design Pattern, MVC	Online	#5	
8	5/22	Complexity, Sorting, Searching Midterm Exam #2 (covers Weeks 4 to 6)	Ch. 13		#2
9	5/29	Monday: Memorial Day Holiday Stacks and Queues	Ch. 14		
10	6/5	Maps, Hashing	Ch. 18	#6	
11	6/12	Final Prep Final Exam (comprehensive): Wednesday		#7	#3

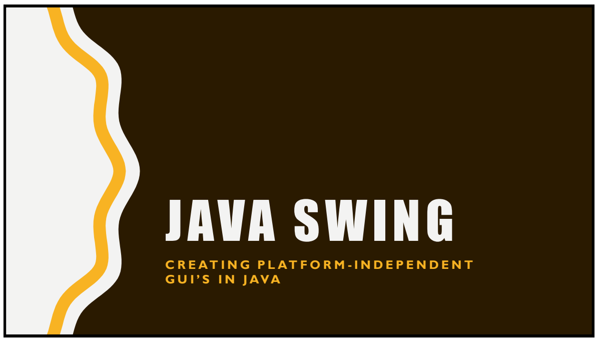
EVENT-DRIVEN PROGRAMMING

A DIFFERENT WAY TO INTERACT WITH THE WORLD

EVENT-DRIVEN PROGRAMMING

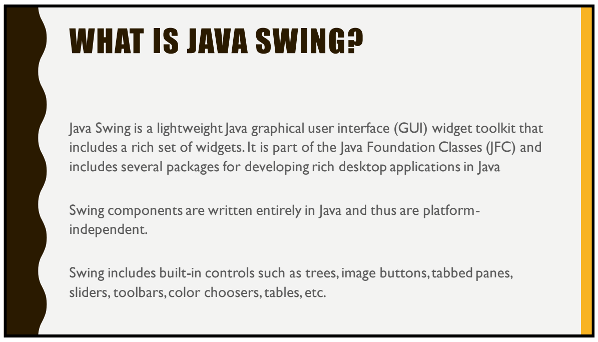
Event-driven programming is a paradigm in which the flow of the program is **determined by events** such as user actions (mouse clicks, key presses), sensor outputs, timers, or messages from other programs/threads

It is the dominant paradigm used in graphical user interfaces



JAVA SWING

CREATING PLATFORM-INDEPENDENT GUI'S IN JAVA

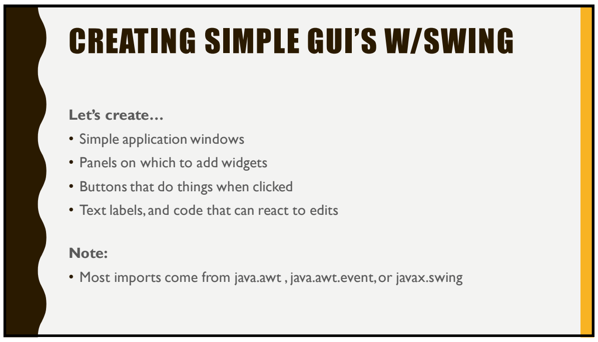


WHAT IS JAVA SWING?

Java Swing is a lightweight Java graphical user interface (GUI) widget toolkit that includes a rich set of widgets. It is part of the Java Foundation Classes (JFC) and includes several packages for developing rich desktop applications in Java

Swing components are written entirely in Java and thus are platform-independent.

Swing includes built-in controls such as trees, image buttons, tabbed panes, sliders, toolbars, color choosers, tables, etc.



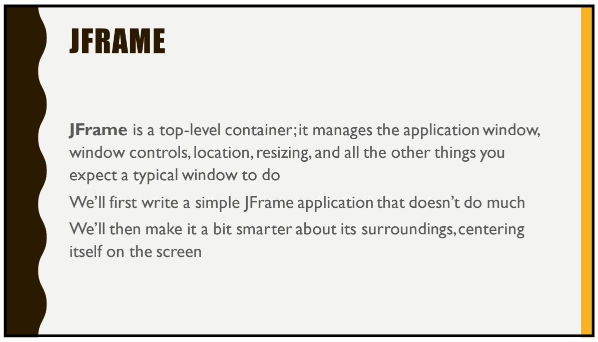
CREATING SIMPLE GUI'S W/SWING

Let's create…

- Simple application windows
- Panels on which to add widgets
- Buttons that do things when clicked
- Text labels, and code that can react to edits

Note:

- Most imports come from java.awt , java.awt.event, or javax.swing

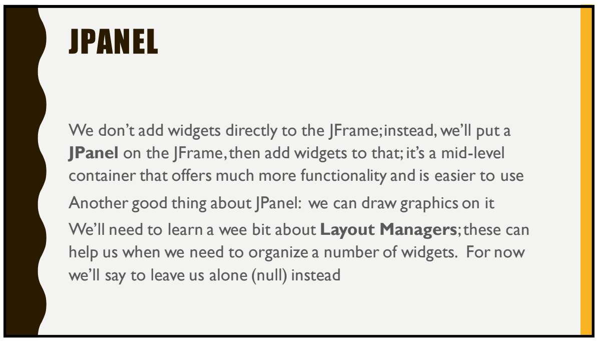


JFRAME

JFrame is a top-level container; it manages the application window, window controls, location, resizing, and all the other things you expect a typical window to do

We'll first write a simple JFrame application that doesn't do much

We'll then make it a bit smarter about its surroundings, centering itself on the screen

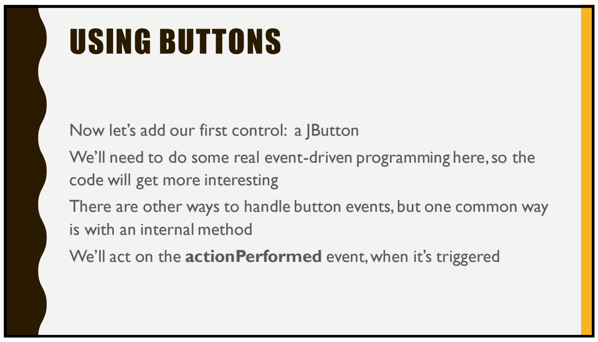


JPANEL

We don't add widgets directly to the JFrame; instead, we'll put a **JPanel** on the JFrame, then add widgets to that; it's a mid-level container that offers much more functionality and is easier to use

Another good thing about JPanel: we can draw graphics on it

We'll need to learn a wee bit about **Layout Managers**; these can help us when we need to organize a number of widgets. For now we'll say to leave us alone (null) instead



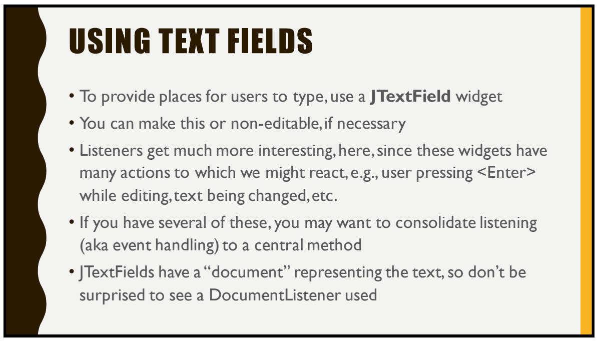
USING BUTTONS

Now let's add our first control: a JButton

We'll need to do some real event-driven programming here, so the code will get more interesting

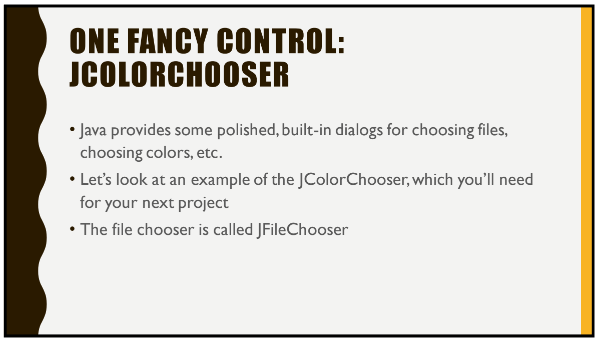
There are other ways to handle button events, but one common way is with an internal method

We'll act on the **actionPerformed** event, when it's triggered



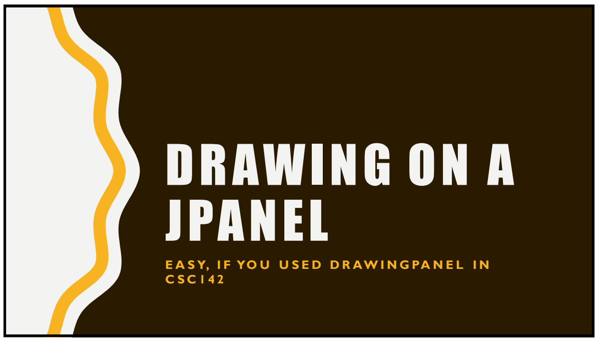
USING TEXT FIELDS

- To provide places for users to type, use a **JTextField** widget
- You can make this or non-editable, if necessary
- Listeners get much more interesting, here, since these widgets have many actions to which we might react, e.g., user pressing <Enter> while editing, text being changed, etc.
- If you have several of these, you may want to consolidate listening (aka event handling) to a central method
- JTextFields have a "document" representing the text, so don't be surprised to see a DocumentListener used



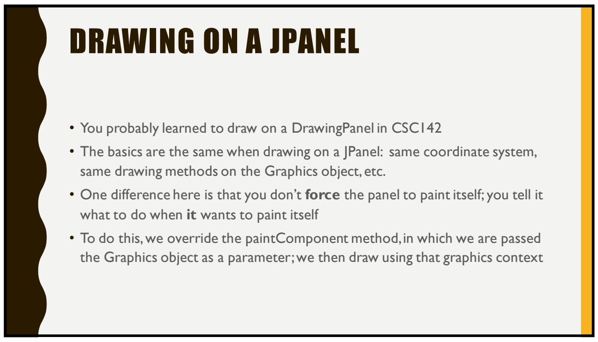
ONE FANCY CONTROL: JCOLORCHOOSER

- Java provides some polished, built-in dialogs for choosing files, choosing colors, etc.
- Let's look at an example of the JColorChooser, which you'll need for your next project
- The file chooser is called JFileChooser



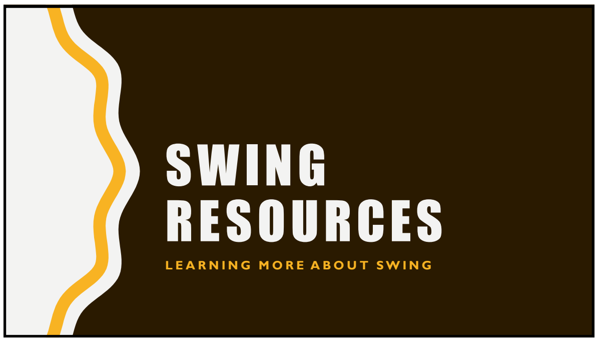
DRAWING ON A JPANEL

EASY, IF YOU USED DRAWINGPANEL IN CSC142



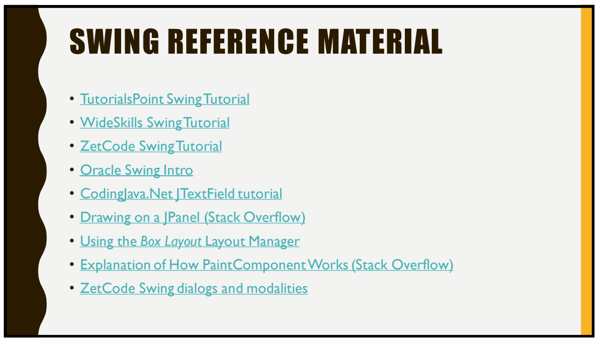
DRAWING ON A JPANEL

- You probably learned to draw on a DrawingPanel in CSC142
- The basics are the same when drawing on a JPanel: same coordinate system, same drawing methods on the Graphics object, etc.
- One difference here is that you don't **force** the panel to paint itself; you tell it what to do when **it** wants to paint itself
- To do this, we override the paintComponent method, in which we are passed the Graphics object as a parameter; we then draw using that graphics context



SWING RESOURCES

LEARNING MORE ABOUT SWING



SWING REFERENCE MATERIAL

- TutorialsPoint Swing Tutorial
- WideSkills Swing Tutorial
- ZetCode Swing Tutorial
- Oracle Swing Intro
- CodingJava.Net JTextField tutorial
- Drawing on a JPanel (Stack Overflow)
- Using the *Box Layout* Layout Manager
- Explanation of How PaintComponent Works (Stack Overflow)
- ZetCode Swing dialogs and modalities

END