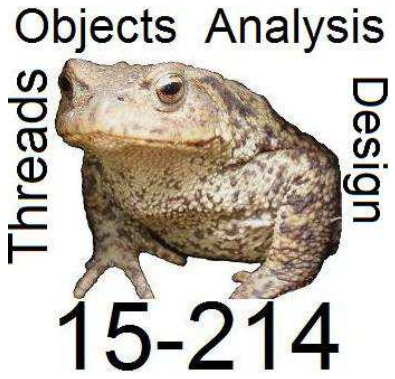
GUIs with Swing



Principles of Software Construction: Objects, Design, and Concurrency

Jonathan Aldrich and Charlie Garrod

Fall 2012

What makes GUIs different?

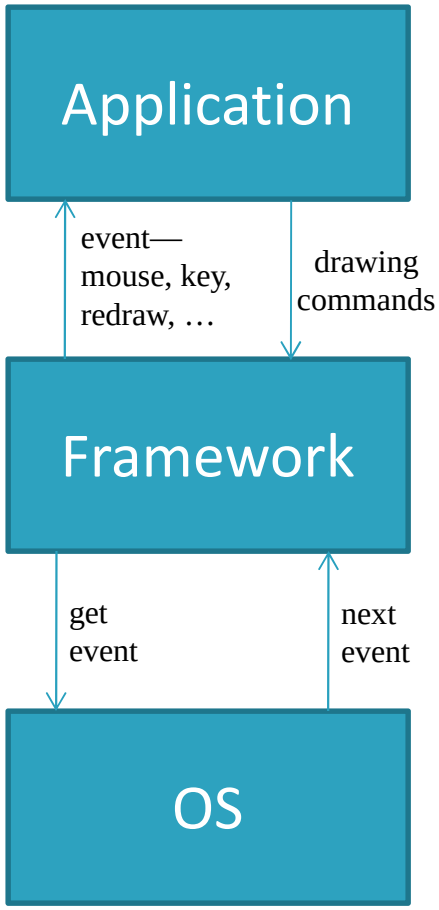
- How do they compare to command-line I/O?

What makes GUIs different?

- How do they compare to command-line I/O?
- One major difference: the user is in control
  - GUI has to react to the user's actions
    - Not just a response to a prompt
    - Could involve entirely different functionality
  - Requires structuring the GUI around reacting to events

Reacting to events - from framework

- Setup phase
  - Describe how the GUI window should look
  - Use libraries for windows, widgets, and layout
  - Embed specialized code for later use
- Customization (provided during setup)
  - New widgets that display themselves in custom ways
  - How to react to events
- Execution
  - Framework gets events from OS
    - Mouse clicks, key presses, window becomes visible, etc.
  - Framework triggers application code in response
    - The customization described above



Pseudocode for GUIs

Application code

- Creates and sets up a window
- Asks framework to show the window

- Takes action in response to event
- May contact GUI
  - E.g. consider if event was a redraw
  - Call GUI to paint lines, text

GUI framework code

- Starts a GUI thread
- This thread loops:
  - Asks OS for event
  - Finds application window that event relates to
  - Asks application window to handle event

  - Draws lines/text on behalf of application

Example: RabbitWorld GUI

- …hw2.lib.ui.WorldUI.main()
  - Creates a JFrame (i.e. a top-level window)
  - Creates a WorldUI to go in it
  - Sets some parameters
  - Makes the JFrame (and its contents) visible

- …hw2.lib.ui.WorldPanel.paintComponent()
  - Called when the OS needs to show the WorldPanel (part of WorldUI)
    - Right after the window becomes visible
  - super.paintComponent() draws a background
  - ImageIcon.paintIcon(…) draws each item in the world

Cookbook Programming

- Typical mode of using a framework
  - Let's you follow a recipe for writing your programs
  - All cakes are different, but there are a few basic recipes and everything else is a slight variation
    - Add some cinnamon
    - Substitute chocolate chips instead of nuts
- Tends to be most effective way to learn a framework
  - Typically infeasible to read the documentation of all operations
  - Instead, find a "recipe" similar to what you need to do
  - Understand the recipe by reading about the ingredients
    - Selective reading of the documentation
  - Then you can combine the ingredients in new ways with confidence

Cookbook Programming

- You have a template for your program
- You change things around, but you don't mess with the overall structure
- Examples:

```
public static void main(String[] args) { … }
for (int i=0; i<args.length; i++) { … }
```

- Many people consider Swing development to be cookbook programming

A Little History

In the beginning…

- There was Java. It was like C++, but simpler and cleaner.

- Then came HotJava, a Java-based browser
  - You could run chunks of Java code called Applets
  - It was cool → Netscape & then IE added Java support

- But Applets were a pain
  - Browsers had out of date JVMs
  - Used the AWT (lots of platform-based non-Java code)
  - Didn't have the look and feel of the rest of the platform
  - Couldn't run as a standalone program with a GUI

Swing

- A new user interface environment
  - Implemented in Java
    - More consistent across implementations
  - Offers different “look and feel” options
    - Windows, Unix, and other (Metal)
  - Can be a main method or a Japplet

- Still uses AWT for event handling, fonts, etc.
  - BTW – still issues with Swing non-native look and feel, predictable performance
  - SWT – An alternate Standard Widget Toolkit (from Eclipse) addresses this by staying closer to OS windowing support
    - but, not standard for Java

Simplest Structure

- You make a Window (a JFrame)
- Make a container (a JPanel)
  - Put it in the window
- Add your Buttons, Boxes, etc to the container
  - Use layouts to control positioning
  - Set up listeners to receive events
  - Optionally, write custom widgets with application-specific display logic
- Set up the window to display the container

- Then wait for events to arrive…

Components

Swing has lots of components:

- JLabel
- JButton
- JCheckBox
- JChoice
- JRadioButton
- JTextField
- JTextArea
- JList
- JScrollBar
- … and more

JFrame & JPanel

- JFrame is the Swing Window
- JPanel (aka a pane) is the container to which you add your components (or other containers)

Layout Managers

- The default Layout Manager is FlowLayout
  - Place items in the container from left to right
  - When a line is full, FlowLayout goes to the next

More Layout Options

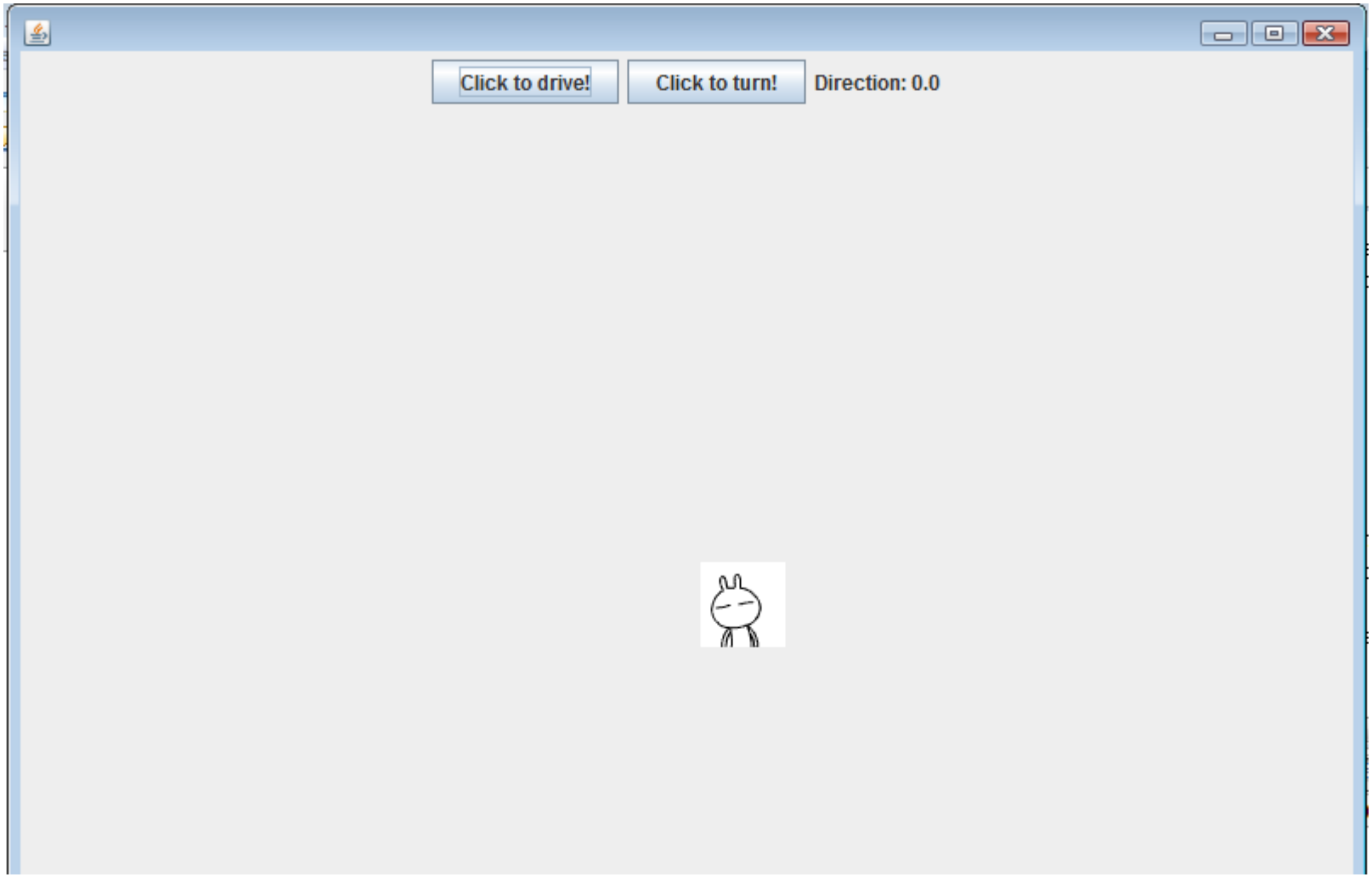
- GridLayout
- GridBagLayout
- Explicit Placement

Example: RabbitWorld GUI

- …hw2.lib.ui.WorldUI.WorldUI()
  - Sets the layout to a BorderLayout
  - Adds a WorldPanel in the CENTER of the UI
  - Creates a JPanel for the buttons at the bottom
  - Adds 2 buttons to the JPanel (WEST and CENTER)
  - Puts the button JPanel at the SOUTH side of the WorldPanel

Question

- How do you make a button work?



Events in Swing

- An event is when something changes
  - Button clicked, scrolling, mouse movement
- Swing (actually AWT) generates an event
- To do something you need to implement a Listener Interface and register interest

Event Listeners

Swing has lots of event listener interfaces:

- ActionListener
- AdjustmentListener
- FocusListener
- ItemListener
- KeyListener
- MouseListener
- TreeExpansionListener
- TextListener
- WindowListener
- …and on and on…

ActionListener

- Events for JButtons, JTextFields, etc
  - The things we are using
- Implement ActionListener
  - Provide actionPerformed method
- In actionPerformed method
  - Use event.getSource() to determine which button was clicked, etc.

Example: RabbitWorld GUI

- …hw2.lib.ui.WorldUI.WorldUI()
  - Sets ActionListeners for the **run** and **step** buttons
    - Anonymous inner classes used
    - A single method actionPerformed(…) is overridden
    - **step** button: just calls step() on the WorldPanel
      - Steps the world
      - Requests that the window be refreshed (so the user can see the changes)
    - **run** button
      - Starts the world continuously stepping
      - Disables the **step** button (no point!)
      - Sets a toggle flag so that pressing the button again will stop the simulation

Organizational Tips

- Declare references to components you'll be manipulating as instance variables
- Put the code that performs the actions in private "helper" methods. (Keeps things neat)

GUI design issues

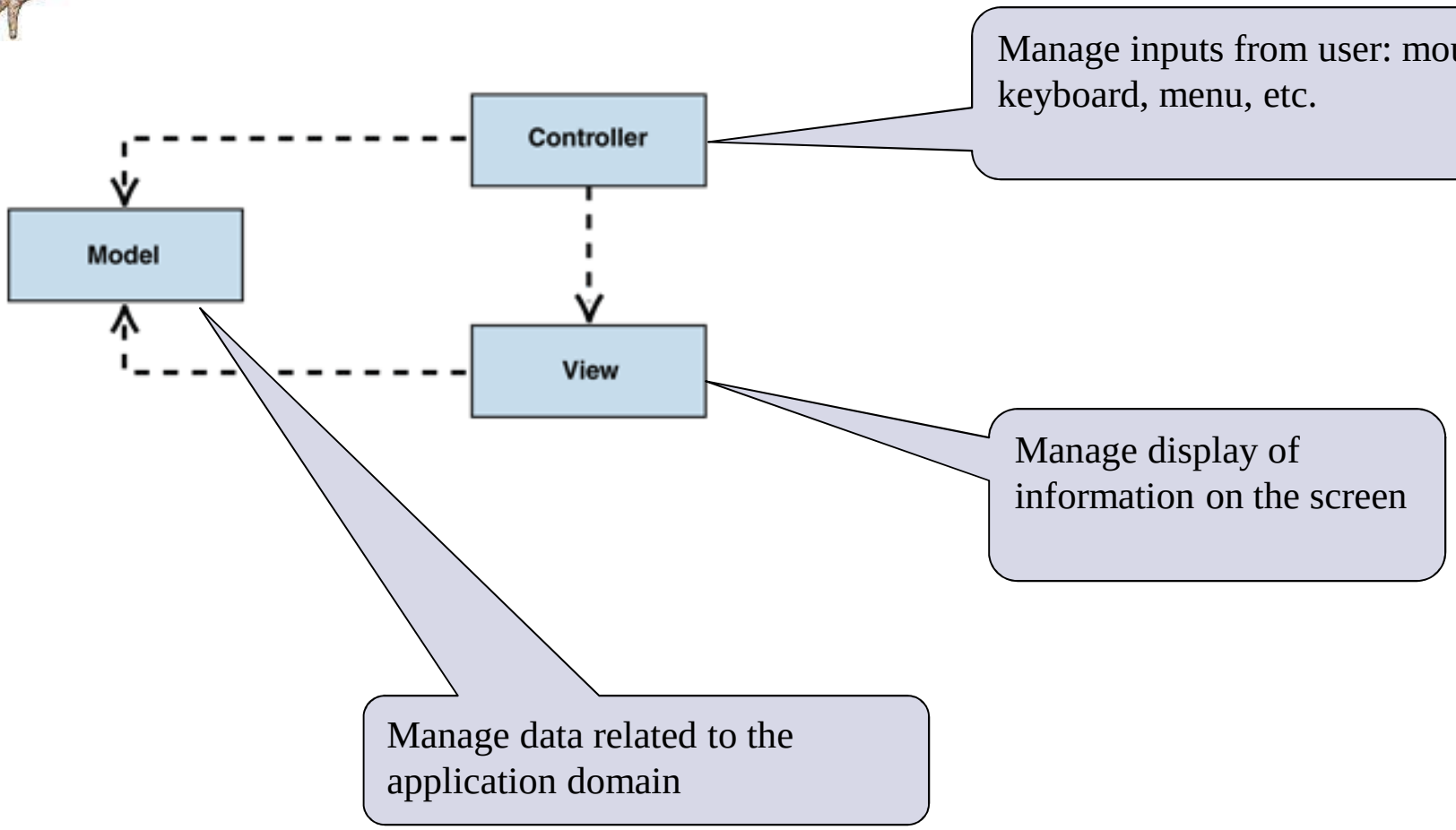
- Interfaces vs. inheritance
  - Inherit from JPanel with custom drawing functionality
  - Implement the ActionListener interface, register with button
  - Why this difference?

- Models and views

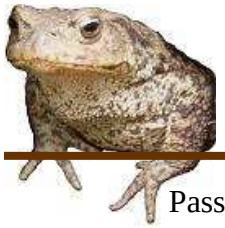
GUI design issues

- Interfaces vs. inheritance
  - Inherit from JPanel with custom drawing functionality
    - Subclass “is a” special kind of Panel
    - The subclass interacts closely with the JPanel – e.g. the subclass calls back with super()
    - The way you draw the subclass doesn’t change as the program executes
  - Implement the ActionListener interface, register with button
    - The action to perform isn’t really a special kind of button; it’s just a way of reacting to the button. So it makes sense to be a separate object.
    - The ActionListener is decoupled from the button. Once the listener is invoked, it doesn’t call anything on the Button anymore.
    - We may want to change the action performed on a button press—so once again it makes sense for it to be a separate object
- Models and views

Model-View-Controller (MVC)

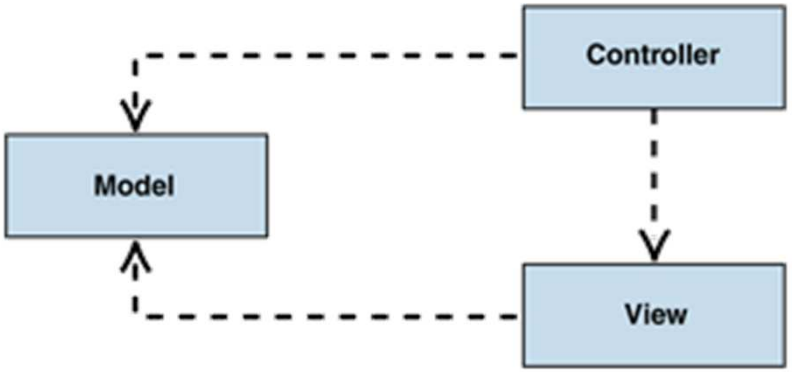


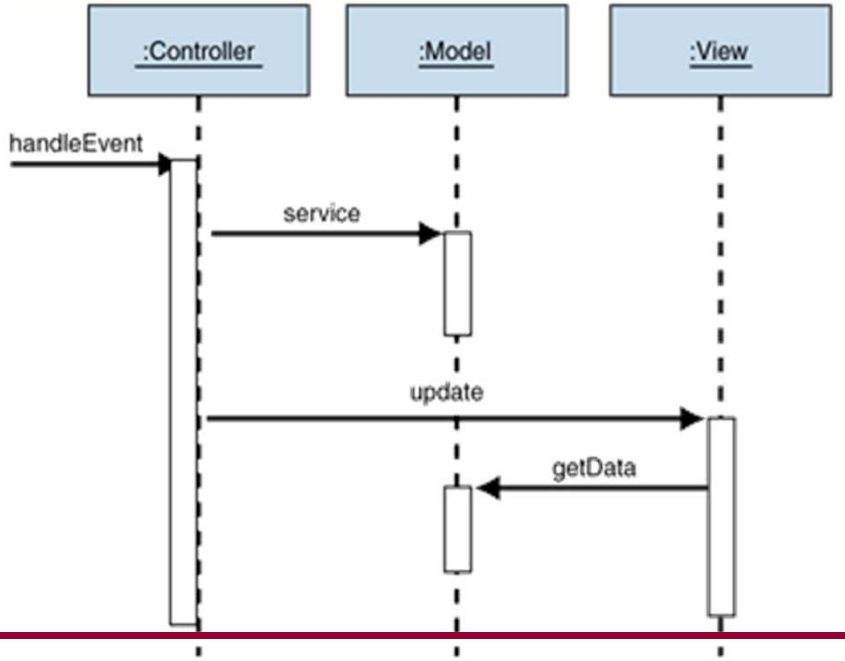
http://msdn.microsoft.com/en-us/library/ff649643.aspx



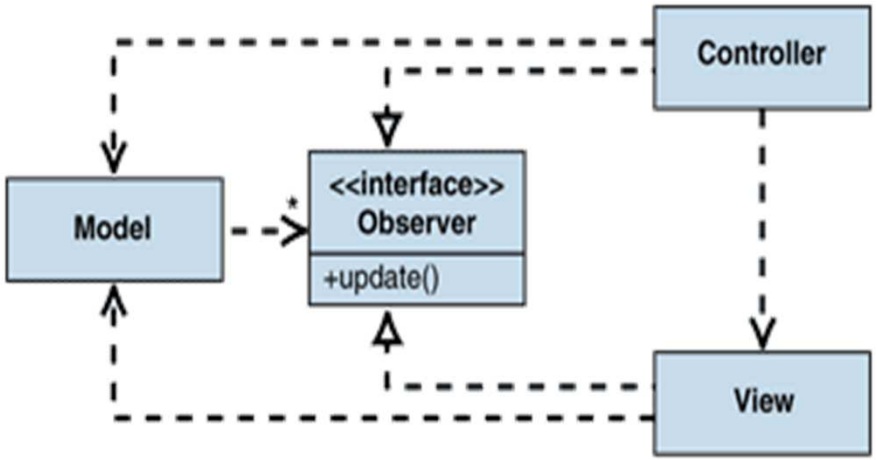
Model-View-Controller (MVC)

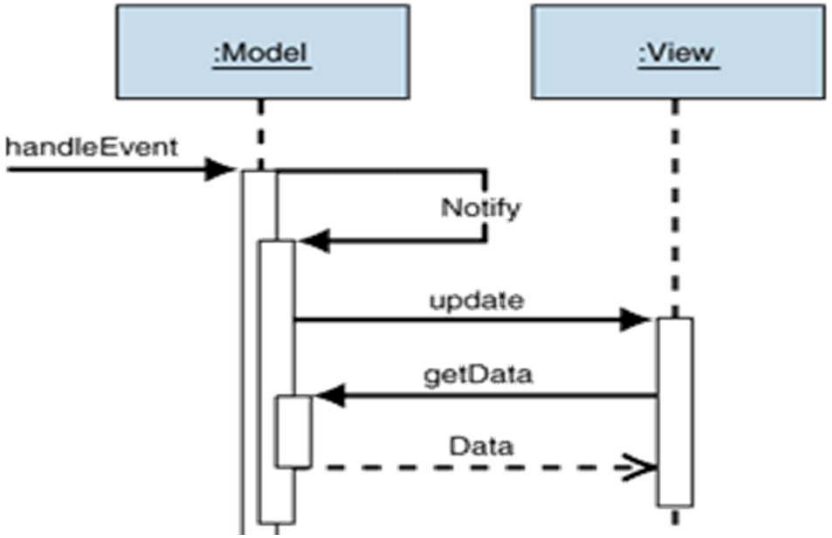
Passive model





Active model





http://msdn.microsoft.com/en-us/library/ff649643.aspx

Example: RabbitWorld GUI

- …hw2.lib.ui.WorldImpl
  - The Model class
  - Model is passive: does not have a reference to the view
- …hw2.lib.ui.WorldUI
  - The Controller class
  - Listener callbacks in constructor react to events
    - Delegating to the view (is this design ideal?)
- …hw2.lib.ui.WorldPanel
  - The View class
  - Gets data from Model to find out where to draw rabbits, foxes, etc.
  - Implements stepping (in step())
    - Invokes model to update world
    - Invokes repaint() on self to update UI

Find That Pattern!

- What pattern is BorderLayout a part of?
- What pattern is JPanel a part of?
- What pattern are the ActionListeners part of?
- There are classes representing the AI's decision to Eat, Breed, or Move. What pattern are these representing?
- Look at the documentation for JComponent.paint(). What pattern is used?

For More Information

- Oracle's Swing tutorials
  - http://download.oracle.com/javase/tutorial/uiswing/
- Introduction to Programming Using Java, Ch. 6
  - http://math.hws.edu/javanotes/c6/index.html

Questions?