

# For Wednesday

- Read Becker, chapter 13 (the handout)
- Ethics Exercise due

# Program 9

- Any questions?

Questions before the quiz?

# Quiz

# Building the Windows

- This is the easy part.
- You're going to use some existing classes, maybe do some inheritance from those classes.
- There is good documentation and there are a lot of tutorials on these components.

# The Basics

- Create a window (JFrame)
- Provide places to display any data needed
- Provide ways for the user to trigger events (get stuff done)
  - Can include buttons, menus, control keys

# SWING and AWT

- Java classes/interfaces

# Text Components

- Label
- Text field
- Text area

# Button Types

- Buttons
- Check boxes
- Radio buttons
  - The ButtonGroup

# Listeners

- What are listeners for?

# Listeners

- We add listeners to components to get the underlying Java mechanisms to catch events and call the appropriate listener to get the work done.
- Note that listeners are interfaces—we're always providing the instructions on how to respond when the user does something.

# ActionListener

- Used for things like buttons, menu selections
- Method is `actionPerformed` which takes an `ActionEvent` parameter
- How do we attach an `ActionEventListener` to its button?

# Other Listeners

- `MouseListener`
- `MouseMotionListener`
- `ItemListener`
- `KeyListener`
- Many others: see descendants of `EventListener`

# Listener Design

- Can be a separate class
- Can be the component
- May be individual classes for each component
- May use one listener that handles several different event (differentiated by source)
- Any CAN be the right choice – think about the design. Think about how closely related different pieces of the program are.

Some Code

# Program Design with GUIs

- Example GUIs trivial
- Mix data with interface
- Is this a good idea?

# Design Principles

- Change happens: design for it.
- It should be easy to locate a change.
- One change = one place.
- Change should require little or no change to other parts of program.
- If there's error in changing X, Y should not be affected.
- Everything in the real world is much more complex.

# Designing GUIs 2

- How do these principles apply to GUI design?

# MVC Pattern

- In OO programming, there's a concept of "design patterns"
- Basically, ways of organizing classes to do very common sorts of things
- One pattern is the Model-View-Controller pattern

# Basic Idea

- Model == the data and behavior (the basic class)
- View == the display mechanism for the data
- Controller == the input mechanism for the data
- Allows for a single model to have many views/controllers
- Creates better code reuse – Why?

# The Model

- Model is the data manager.
- Has tasks to carry out.
- Does not have any interface to user.
- Does have to provide appropriate methods to allow access to view and allow controller to send appropriate messages.
- Also has to let the view(s) know when it changes. Why?

# Java Support for the Model

- `java.util.Observable`
- Can “register” views
- Can notify views

# The View

- Provides the output part of the interface
- Usually GUI
- Must be aware of the model (so it can ask for information to display)
- Must be able to let the model know about it (so it can get notified)
- Must be notifiable
- May be several views—Examples?

# Java Support for the View

- Observer interface
- A single method to implement: update

# The Controller

- Provides the input part of the interface
- Must know about the model (to tell it to do things)
- Controllers are typically listeners in Java
- Again, may be several—Examples?

# Sample MVC Program