Slide 1

Administrivia

- Reminder: Homework 3 design due today, code Tuesday.

Slide 2

A Little About Homework 3

- In this homework you start writing code for your player, to replace the stick figure in the starter game.
- Key parts of this assignment are making the player
  - interact with different kinds of blocks.
  - move in response to keyboard or mouse input from human player.

  (If these don't apply to your game, talk to me about whether there are reasonable substitutes.)

  For design phase, you just need to describe this interaction.

Slide 3

Homework 3, Continued

- `Player` defines some constants you should use.
- You will implement `KeyListener` or one/both of the mouse-listener interfaces. When you do this, the framework will deliver key and/or mouse "events" to you.
- Most logic will go in `update`, `getUpdateTime`, and the listener methods.
- A general comment: If you find yourself looking up something like the ASCII value of a character, or the value of one of the game framework's constants — *stop*. There is probably an easier and more Javaesque way to do what you want.

Slide 4

Polymorphic Sorting and Searching in Java, Recap

- Java library interface `Comparable` is helpful in writing comparison-based sorts. (Example code from last time.)
- But what if you sometimes want to sort data one way and sometimes another? With C's `qsort` you can pass in a function pointer. In Java? You can't do that. What you can do (very typical) is create an object whose purpose is to contain the desired code. Here, we want something to hold our `compareTo` method. Simplest to illustrate using a library class (next slide).

Slide 5

Sorting and Searching Arrays in Java

- Writing your own sorting routines is pedagogically useful, but in practice you would probably use something from Java library.
- `Arrays` class has some useful methods. The ones for objects require either a class that provides a `compareTo` method, or a `Comparator` object that defines the ordering you want.

Slide 6

Abstract Data Types

- "Abstract data type" (ADT) is defined as
  - A set of values.
  - A set of operations on those values.
- In other words — something that stores data (in an unknown form) and provides a standard interface for dealing with it.

Slide 7

Stack ADT

- Value — list of elements.
- Operations — push, pop, “empty?”
- Implementing this? might be a good example of
  - Defining a (generic) interface.
  - Writing a class to implement it (using arrays — for, um, fun? practice?).
  - Working with exceptions.

Slide 8

Queue ADT

- Value — list of elements.
- Operations — enqueue, dequeue, “empty?”
- We could implement similarly to what we did for stacks . . .

Slide 9

Minute Essay

- None — quiz.