Slide 1

Administrivia

- Reminder: Homework 2 design due today, code Thursday.
- Quiz 3 moved to next Tuesday.

Slide 2

Homework 2 Code — How to Approach Defining a Class

- What methods do I need? If implementing an interface, you at least need the methods in the interface. May want additional methods. If making a subclass, remember you automatically inherit all methods from superclass. Can override them and/or provide additional methods.
- What variables do I need to implement the needed methods? e.g., if defining a `Rectangle` class that has a `getArea` method, probably need either area or width and height.
- The class where this advice will be most relevant is the one implementing the `Screen` interface. You will need to represent your 2D grid of blocks and a list of entities. What kinds of variables would be good? (Look at the game framework API for hints about the list.)

Slide 3

Homework 2 Code — Some Tips

- Eclipse will suggest adding a variable called `serialVersionUID` to some of your classes. Do that. (Notice there's one of these in some of the provided code.) Value can be anything. We will talk later about what this means and how to make use of it.
- Notice that x/y coordinates of framework are opposite of row/column. `getSize()` in screen class should return width by height.
- To confirm that your code works:
  - Start the game, and verify that the playing field is what you defined (dimensions, plus appearance of blocks — for now, solid colors are okay).
  - Try running the screen editor (directions in "project description" document). If it comes up, and shows all the kinds of blocks you defined, all is well. (Actually it doesn't have to do that if you don't plan to use it — it just has to not crash.)

Slide 4

Sorting and Searching — Review

- Sorting — putting elements of a list or array in order (according to some notion of ordering). Various methods; for now you should understand the simple/slow ones (bubble sort, insertion sort, selection sort).
- Searching — looking for an element in a list or array. You should understand sequential and binary search.

Slide 5

Polymorphic Sorting and Searching

- Sort/search algorithms are (mostly) independent of the kind of data being sorted — all of the comparison-based sorts just require that a “total ordering” relation on the data (for any two distinct elements a and b, $a < b$ or $b < a$). (“Comparison-based”? yes, as opposed to, e.g., radix sort or counting sort described last time.)
- So we’d like to be able to turn the algorithm into code just once, and let it operate on different kinds of data — “polymorphic sort”. C’s `qsort` is polymorphic, though the mechanics are a bit ugly. Java provides nicer mechanisms — for objects anyway.

Slide 6

Polymorphic Sorting and Searching in Java

- Java library interface `Comparable` is helpful in writing comparison-based sorts. (Look at its API. Example code as time permits.)
- 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?

Slide 7

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. Example: case-insensitive sorting of strings.
- (Examples as time permits.)

Slide 8

Minute Essay

- None — sign in.