Slide 1

## Administrivia

- Reminder: Homework 1 code was due Tuesday, but only a few have turned it in. Problems?
- Reminder: Homework 2 design due today, code Tuesday.
- Office hours schedule on my Web page. T/R afternoon hours will be "open lab", in HAS 340. (Probably also some of Wednesday afternoon, subject to room availability.)

Slide 2

## Arrays in Java — Review

- Declaring and creating arrays in Java is different from in C — examples:

  ```
  int[] x = new int[10];
  String[] s = new String[n];
  ```

- Once created, though, some things are familiar — syntax for finding elements, range of indices.

  (Notice, though, that the second example above creates not `String` objects, but *references* to `String` objects.)

- Under the hood, more differences — in C, arrays are almost indistinguishable from pointers, but in Java, they're objects, with a `length` field you can use (but not change), and built-in bounds checking.

Slide 3

## Multidimensional Arrays

- "Arrays of arrays", e.g.,

  `int[][] x = new int[10][100];`

  declares an array of 10 arrays of 100 `int`s.

- Reference elements with row, column indices, e.g.,

  `x[row][col] = 10;`

- Both dimensions accessible:

  `x.length` = ?

  `x[0].length` = ?

Slide 4

## Homework 2 — General Comments

- Design phase is meant to be about defining classes and interfaces. For every class (or interface) and every method, I want comments (can be be brief). For classes, these should describe (to the best of your understanding) how they fit into your game (e.g., "class for wall blocks").
- In order to generate the HTML documentation ("javadoc"), you probably have to have something minimally compilable. As suggested in assignment — you can create skeleton/stub versions of methods, and fill in real code in code phase. (For classes where you get code, though, might be simpler just to copy it in right away, if there are comments in the code. Or copy comments from game framework API.)
- Be sure to get the updated JAR file (should have name `PAD2F09Assn2.jar`). With every assignment there will be a new JAR file, as you replace various parts of the starter code with your code.

Slide 5

## Homework 2 Design

- Interfaces `YourBlock`, `YourEntity`: In project API, referred to as “general block type” and “general entity type”. You will use these as replacements for `BasicBlock` and `BasicEntity`, and everywhere else you use one of the framework’s generic classes.
- Player and game setup classes. Copy code from `BasicPlayer` and `BasicGameSetup` and edit (change `package` line, block and entity types). May want to change game setup more during code phase. Also edit your main class from the first assignment.

  Don’t worry about player for now — you will start writing your own in the next assignment.

Slide 6

## Homework 2 Design Continued

- Block class(es). These are blocks that make the playing field for your game. Should have one class for each kind of block (floor, walls, ladders, anything that doesn’t move). Try to define as many as you can. Copy code from `BasicBlock`.
- Screen class (class implementing `Screen` interface). This is the most work in this assignment. Eclipse can make stub methods for you. Copy and paste comments from API.

Slide 7

## 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 8

## 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 9

## Sorting and Searching Arrays

- A common thing to do with arrays is sort them. (Remember this from PAD I or equivalent?)
- Various algorithms for sorting and searching. Some fast, some slow; some simple, some complex. Decide which to use based on considerations of simplicity versus speed.
- “Speed”? Yes, but expressed as order of magnitude (“big-oh notation”).

Slide 10

## Order of Magnitude of Algorithms

- Idea is to estimate how work (execution time) for algorithm varies as a function of “problem size” (e.g., for sorting, size of array). (Similar idea can be applied to how much memory is required.)
- Usually do this by counting something that represents most of the “work” in the algorithm and varies with problem size (e.g., for sorting, how many comparisons).

Slide 11

## Order of Magnitude of Algorithms, Continued

- Informally, $O(N)$ means work/time is proportional to $N$ (problem size). $O(N^2)$ means ...?

  (Compare $aN$ and $bN^2$ as $N$ increases, for different values of $a$ and $b$. $bN^2$ larger for larger enough $N$.)
- Formal definition (from CSCI 1323): $g(n)$ is $O(f(n))$ if there are positive constants $n_0$ and $c$ such that for $n \geq n_0$,

$$g(n) \leq cf(n)$$

Slide 12

## Simple (but Slow) Sorts

- Bubble sort. (First pass goes through the whole array, swapping consecutive elements if out of order, so largest element bubbles to the end. Next pass goes through all elements but last. And so forth.)
- Selection sort. (First pass finds largest element and puts it at end. Next pass finds next-to-largest element and puts it at next-to-end. And so forth.)
- Insertion sort. (First pass inserts second element into list of first element. Next pass inserts third element into list of first two elements. And so forth.)
- All of these are $O(N^2)$. And there are others ...

## Other Sorts

- Quicksort (to be discussed later). $O(N \log N)$.
- Mergesort (to be discussed later). $O(N \log N)$.
- Many others ...

## Searches

- Sequential search. $O(N)$.
- Binary search. $O(\log N)$.

## Minute Essay

- None — quiz.