

Slide 1

Administrivia

- Reminder: Homework 2 design due Thursday.

Slide 2

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”), probably have to have something minimally compilable. As suggested in assignment — create skeleton/stub versions of methods, and fill in real code in code phase.
- Be sure to get the updated JAR file (should have name `PAD2F08Assn2.jar`). With every assignment there will be a new JAR file, as you replace various parts of the starter code with your code.
- Note that order of array indices (row then column) is the opposite of the “graphics convention” used in the game.

Homework 2 — Design

Slide 3

- 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.

Homework 2 — Design Continued

Slide 4

- 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.

Objects Versus References — A Caution

- What does `new MyClass[10]` actually create?
- If `MyClass` contains a method `foo`, will the following code work properly?

```
MyClass[] a = new MyClass[10];  
a[0].foo();
```

Slide 5

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 6

Slide 7

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 8

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 9

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 ...

Slide 10

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. $O(\log N)$.

Slide 11

Sorting and Searching — Example Code

- See “Sample programs” page ([here](#)) Code performing an instrumented sort (count number of comparisons), and other examples.

Slide 12

Slide 13

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.
- One thing that's nice about Java is "polymorphic sorting"; can sort objects of any class that implements `Comparable`. Can also provide, when you call `Arrays.sort`, a `Comparator` that defines the ordering you want. Example: case-insensitive sorting of strings.
- (Example code next time.)

Slide 14

Minute Essay

- None — quiz.
- (Quiz solutions posted on the Web shortly after class, and I will usually bring a hardcopy to class, if you want to take a quick look after turning in your paper.)