Slide 1

Administrivia

- Reminder: Homework 2 design due Tuesday. Remember that it's not a bad idea, after you generate HTML documentation, to point a browser at it and check that your comments are showing up where they should.
- Student chapter of the ACM says they plan to start doing tutoring, 6pm–7:30pm Sundays through Thursdays in HAS 340. Might be a reasonable source of help?
- (Comments on minute essay from last time.)

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"), 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 `PAD2F10Assn2.jar`). With every assignment there will be a new JAR file, as you replace various parts of the starter code with your code.

Slide 3

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 4

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 5

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 6

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 7

Strings and Arrays — Review/Recap

- Java has a library class `String` for working with text strings. Many useful methods. Be aware that instances of this class are immutable.
- Java arrays are objects (in the same way that instances of classes are objects), with a `length` variable and runtime checks to prevent out-of-bounds access.
- (Examples as time permits. Next topic — sorting and searching — will use arrays too.)

Slide 8

Bank Example Revisited/Improved

- Current classes represent balance as `int`. Use a `USMoney` class instead.
- Current `withdraw` and `deposit` methods don't do any error checking. Add some (using exceptions).
- Current hierarchy has `Account` class with subclasses `PersonalAccount` and `CorporateAccount`, and interface `PaysInterest`. Review/expand — subclasses of `PersonalAccount`, one implementing `PaysInterest`, etc.

Slide 9

Minute Essay

- None — quiz.