Slide 1

Administrivia

- Reminder: Homework 3 design due today; code Tuesday. Also, remember that:
  - You should send me mail telling me your design is ready (most people forgot for Homework 2).
  - You should generate a “final” version of the javadocs (most people forgot for Homework 2).
- Reminder: Quiz 3 Tuesday. Likely topic is stacks and queues (implementing them, using them).

Slide 2

Quote of the Day/Week/?

- “As soon as we started programming, we found to our surprise that it wasn’t as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent finding mistakes in my own programs.” (Maurice Wilkes: 1948)

  (Wilkes was a key figure in the early days of computing.)

Slide 3

Homework 3 Hints — General

- Remember that most game framework interfaces and classes are generic, so to use them you should supply two "type parameters" (your block and entity interfaces). Why is it written that way? should start to become clearer with this assignment.
- Two groups of methods to define:
  - Methods of two framework interfaces, `Player` and `GameEntity`. Called once per game tick (normally — but you can have it called less often).
  - Methods of appropriate listener interface(s). Called when human player provides input.
- Think about what variables you need — in general, if there's something that's part of the object's "state" and needs to be used by more than one method, it should be an instance variable.

Slide 4

Homework 3 Hints — Drawing Things

- Individual blocks and entities: What's drawn is controlled by `getImage`. Blocks all scaled to same size. Entities scaled based on `partialSizeX/Y`. Positions of entities based on locations.
- "Partial"? The framework allows you to define, for the purposes of moving and scaling, a "partials in whole" number (to allow moving in fractions-of-a-block units).

Slide 5

Homework 3 Hints — Drawing Things, Continued

- Laying out screens: You can do this in code (probably in your screen class) or using the “screen editor” (brief description and links to more info in writeups for Homeworks 2 and 3).

  Potential “gotcha”: If you set “partials in whole” to a non-default value (in your game setup class), and you want to use the screen editor, you also need to set “partials in whole” in your screen class.

- Notice / recall that not everything has to be part of the playing field: Your game can also include “panels” on any or all four sides. (We won’t add those until Homework 6; for now you could consider just printing information to the console.)

Slide 6

Homework 3 Hints — Responding to Input, Moving Around

- Game ticks and keyboard/mouse events aren’t particular in synch.
- So listener methods should probably just record information, to be processed by `update` method.
- Look at documentation of (Java library) listener interfaces to know what methods to write. Follow links to find out about other useful classes (e.g., `KeyEvent`).
- “Move” by changing location. Useful methods in (framework) `Location` class.

Slide 7

Homework 3 Hints — Interacting With Blocks

- At least some code for interacting with blocks goes in player classes. Similarly for other entities. However, good use of block hierarchy can help.
- Example — how do you not go through walls?

  (Contrast the "old way" using `instanceof` versus the "new way" using polymorphism and your interfaces.)
- Interacting with other entities starts in Homework 4, and is done in a similar (but not exactly the same) way.

Slide 8

Lists — Recap

- List ADT (review):
  - "Values" are lists of elements.
  - Many operations possible — add element, remove element, search for element, etc., etc.
- More than one possible implementation, but a typical one is a "linked list". We sketched some code last time — let's finish that.

Iterators

Slide 9

- Something we often want to do with this and other “container classes” is do something to all elements — i.e., we want a way to visit all elements, in some (or any) order.
- An object-oriented way to address this is to have “iterator” objects with methods to support “visit every element, one at a time”. In Java — `java.util.Iterator` interface. (Look at its methods.)

Iterators, Continued

Slide 10

- To see how this plays out in code, we could define a simple interface for lists, including an iterator, and implement it using `java.util.Vector` (something that more or less behaves like an array but can grow and shrink).
- Let’s do that . . .

  (No, it’s not a very sensible implementation of the list ADT, but it’s one I’m willing to put on the Web as sample code. You’ll write a linked-list class as part of Homework 4.)

Slide 11

Minute Essay

- In Homework 2 you wrote, as part of your screen class, a method called `createEntityIterator`. Using this method and what you now know about iterators, sketch code you could use elsewhere in your game to "walk through" the list of entities on a screen and do something with them (count them, for example). Assume that variable `firstScreen` points to an instance of your screen class.
- Can you think of something you want to do in your game where you would need similar code?

Slide 12

Minute Essay Answer

- You could write something like the following:

```
Iterator<MyGameEntity> iter =
            firstScreen.createEntityIterator();
int count = 0;
while (iter.hasNext()) {
    iter.next();
    ++count;
}
```

- It might be useful for detecting whether your player has collided with another entity.