Slide 1

## Administrivia

- Homework 4 on Web. Design due next Tuesday, code Thursday.
- Homework 1 grades and comments mailed, finally, and comments on Homework 2 design. More coming soon, I hope.

Slide 2

## Homework 4 Overview

- Start writing code for your game entities. Similar to what you did for player last time.
- Review/revise how you're creating layout for your game. Several options.
- Write replacement for framework `GameEntityList`. This will be a linked list, based on discussion from last week. You may find it helpful to draw pictures, as we did last week.

Slide 3

## Error Handling — The Problem

- When you have a function in which something goes wrong, how do you tell the rest of the program?
- Examples:
  - Calling a square-root function with a negative number.
  - Trying to open (for reading) a file that doesn't exist.
  - Trying to convert a string to an integer, when the string doesn't contain something appropriate.

Slide 4

## Error Handling — "Ostrich Approach"

- Idea — hope it doesn't happen.
- Might sort of work if you tell users in your documentation, and maybe use assertions.
- But users make mistakes, and what then? e.g., out-of-bounds array access.
- And it may not always be easy to tell what inputs will produce errors (e.g., file access).

Slide 5

## Error Handling — Return Codes

- Idea — have method return an error code if something goes wrong.
- Works well in situations where it might be hard to avoid sometimes causing the error.
- But requires that users of the method check for the “error” return value — tedious and error-prone.
- And what about methods that want to return a value? is it always possible to designate some value as “this means an error”?

Slide 6

## Error Handling — Setting Flags

- Idea — have method set a flag somewhere if something goes wrong.
- Also useful in situations where it might be hard to avoid sometimes causing the error.
- Again, though, users have to check.
- Requires either an extra parameter (and changing it may be tricky in Java) or a “global” variable somewhere.

Slide 7

## Error Handling — Exceptions

- Idea — when something goes wrong, "throw an exception". What then?
- Aside — as program runs, we can think of it keeping a stack of nested method calls ("push" when we call a method, "pop" when one returns).
- When an exception is thrown, runtime system works its way up this stack until it finds something to "catch" the exception. If it never finds anything, it terminates the program (actually the thread).
- *Mostly* this is what Java library classes use to indicate errors — but some use return codes, so read documentation carefully.

Slide 8

## Dealing With Exceptions

- Catching an exception — "try block":

```
try { ....  }
catch (TypeOfException e) { ....  }
catch (OtherTypeOfException e) { ....  }
finally { .... } // optional
```

- Letting an exception "bubble up":

```
void foo() throws WeirdException { .... }
```

- `Exception` class has some useful methods, e.g., `printStackTrace`.

Slide 9

## Checked Versus Unchecked Exceptions

- “Checked exceptions” — ones that sensible programs are supposed to do something about (e.g., file not found).

  Must either catch these, or declare that your method lets them bubble up (and then callers must do likewise).

- “Unchecked exceptions” — ones for which maybe the reasonable thing to do is to just let the program crash.

  Can catch these, or let them bubble up (with or without declaration), possibly eventually crashing the program.

Slide 10

## Throwing Exceptions

- Throwing an exception:

  `throw new TypeOfException(....)`

- Usually best to try to find an existing `Exception` class that fits, but can declare your own.

- Example — we could add to stack and queue classes from previous lecture . . .

## Exceptions Versus Other Approaches

- What's the attraction?
  - Nice mechanism for dealing with errors and unexpected events.
  - Unlike return codes, can't just be ignored.
- But checked exceptions can be annoying to deal with . . .

Slide 11

## Minute Essay

- Here's a line of code that can throw two kinds of exceptions:

  ```
  FileInputStream fis = new FileInputStream("someFile");
  ```

  Write a few lines of code to catch these exceptions and print out a meaningful error message.

Slide 12

## Minute Essay Answer

- First it would be useful to know what the two kinds of exceptions are. You can find that out from the Java library API; if you look up class `FileInputStream`, specifically the constructor that takes a `String` argument, you find that it can throw two kinds of exceptions. Here is code to catch them and print something:

```
try {
    FileInputStream fis = new FileInputStream("someFile");
}
catch (FileNotFoundException e) {
    System.out.println("File " + someFile + " not found");
}
catch (SecurityException e) {
    System.out.println("Security exception accessing " +
        someFile);
}
```