Slide 1

Administrivia

- Reminder: Homework 1 design due Tuesday.

Slide 2

Inheritance, Review/Continued

- Inheritance allows you to define specialized versions of classes, called subclasses.
- Subclasses "inherit" variables and methods from parent classes. This helps avoid duplication of code. (But subclasses can override parent class's variables/methods.)
- Subclasses also define subtypes. This can make it easier to add functionality.
- Look again at bank-account example — write the `Bank` class to work mostly with `Accounts` and it will be easy (easier?) to add new account types.

Slide 3

Inheritance Versus Interfaces

- What if you don't need/want the superclass to provide any code? you just want it to define a "contract" that all subclasses must meet (i.e., a list of methods they must provide?) then you want a Java *interface*.
- In the `Account` example, we could define a `PaysInterest` interface with method `addMonthlyInterest`. This would let us decide for each type of account whether it should pay interest — e.g., `CorporateAccount` and some subclasses of `PersonalAccount`.
- A class can "implement" as many interfaces as you like.

  (This helps if you want a class to inherit from multiple classes — Java, unlike some languages (e.g., C++), doesn't allow that because of possible confusion/ambiguity, but you can fake it by implementing multiple interfaces.)

Slide 4

Interfaces and Types

- Interfaces also define types. So if `CorporateAccount` implements interface `PaysInterest`, we can use a `Account` anywhere a `PaysInterest` is required.

  ```
  PaysInterest p = new CorporateAccount();
  ```

- This is "inclusion polymorphism" — and is what will allow your project code to plug neatly into Dr. Lewis's framework. (The framework is written in terms of interfaces such as `Block` and `Screen`; your classes will implement those interfaces.)

Slide 5

Homework 1 Clarification(s)

- Design phase should be relatively straightforward. (Short demo of generating HTML documentation.)
- (We'll talk about this next time.) Code (due next Thursday) will be short, but may take some time to write. Far from unusual for students to feel a little lost at this point — so try on your own, then ask! One comment now for when you get to that point:

  Method `instance` in `BasicGameSetup` mentions "singleton". What's that about? Reference to "singleton design pattern" — idea that for some classes there should only ever be one instance.

Slide 6

Compiling and Running Java Programs, Revisited

- (We'll talk about this next time.)
- Recall discussion from a previous class about differences between Java and C with regard to compiling, linking, and executing — Java source code compiled to byte code, no linking step (such as is done for C), instead byte code for all classes loaded at runtime by the JVM.
- Where does the JVM find needed byte code? via "class path" — which can include
  - Directories/folders, such as the one(s) containing byte code for your classes — look for `.class` files.
  - "JAR (Java Archive)" files, which can contain byte code for many classes.

  Adding a JAR file to the default class path is a little like adding a flag such as `-lm` when you compile a C program.

Slide 7

Minute Essay

- None — quiz.