Slide 1

Administrivia

- Reminder: Homework 1 due Thursday.

  All homework is considered pledged work. Write "pledged" on hardcopy work, and include it in comments for programming assignments.

- Tentative dates for quizzes on Web. First one next Tuesday.

Slide 2

Multiple Inheritance Versus Interfaces

- What if you want a class to inherit from multiple classes? Some languages (e.g., C++) allow this ("multiple inheritance"). To avoid possible confusion/ambiguity, Java doesn't.
- Instead, define "interfaces" — classes in which *all* methods are abstract.
- In `Account` example, we could define a `HasPersonName` interface with method `getPersonName`. Not obviously useful — unless there's another kind of object that could have a person's name but shouldn't be a subclass of `Account`. (A prospective customer?)
- A class can "implement" as many interfaces as you like.

Slide 3

Interfaces and Types

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

```
HasPersonName o = new Account();
```

- 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 4

Packages and Importing

- Library classes grouped into "packages" — e.g., `java.util`, `java.net`.
- For classes in `java.lang` and "default package", reference using their names only. For other classes, can use full name or `import`. (`import` looks like `#include`, but works differently.)
- Tip: When writing code with Eclipse, if it can't find a particular class because it needs an `import`, select the reference to the class and press shift-control-M, and it will try to generate an appropriate `import`. Shift-control-M "organizes" `import`s (removes any not needed).

Slide 5

Packages, Continued

- You can define your own packages. Convention is to use your e-mail/Web address, in reverse order (e.g., Dr. Lewis's framework is `edu.trinity.cs.gamecore`). For your game, I'm recommending `edu.trinity.cs.yourusername.yourgame` (`yourgame` is something descriptive). Call the main class something with `Main` in its name.
- Packages and filesystem hierarchy are related — after creating a package, look in your Eclipse workspace directory for an example.

Slide 6

"Generics" in Java

- Java library includes classes for collections of things (`ArrayList`, e.g. — like an expandable array). Originally, could put any kind of `Object` in one of these. Nice, except that then there's no way to know anything about types of objects inside except by using reflection (*much* later, if at all) or `instanceof` operator. Must also use explicit casts to do much with objects retrieved from collection.
- So Java 1.5 (a.k.a 5.0) introduced "generics" — Java's answer to C++ template classes, though not exactly the same. Idea is to allow you to specialize a collection — so, a `ArrayList` of `Integer` objects only, or a `ArrayList` of `Account` objects only, etc., etc. Syntax uses angle brackets, e.g., a `ArrayList` that can hold only `Account`s:

```
ArrayList<Account> list = new
ArrayList<Account>();
```

Slide 7

- Also look at API for `MainFrame` in the game framework . . .

Slide 8

Minute Essay

- None — sign in.