Administrivia

- Remember that code from class will be on Web shortly after class (as soon as I can get it there).
- Homework 1 code due today. Updated/corrected JAR file on Web; please download a copy and replace the one you have.
  Feedback on designs coming ASAP (by e-mail).

More Administrivia

- Homework 2 design due next Tuesday. As with Homework 1, “design” just means you have to sketch the outlines — classes, methods — and write comments for the HTML-generating tool. You can fill in variables and code in the “code” phase.
- First quiz next Tuesday. Open book, open notes; also okay to use Java library documentation on Web (just not random Web searches).
  Likely kinds of questions: “What does this code do?” “Write some code to do this.”
Recap — Inheritance

- Many/most object-oriented languages (e.g., Java) allow definition of hierarchies of classes.
- Subclasses inherit methods and variables from their superclass (and its superclass, and so on). This helps with code reuse.
- Subclasses are subtypes of class type. This makes some kinds of polymorphism easy.
  Example — Shape class in a drawing program provides a nice way to abstract out common features (e.g., a `draw` method) while leaving implementation up to subclasses (for circles, rectangles, etc.).
- Classes can be abstract — means at least one method is abstract (no implementation). Subclasses must define abstract methods (unless ...?).

Recap — Interfaces

- Java defines notion of “interface” as essentially a class with no instance variables and only public abstract methods.
- Point is to define a “contract” — any object that implements interface `Foo` has a particular set of methods. Useful in writing library classes that can work on any kind of objects, as long as they provide certain methods (the “contract”).
- Bank example from last time is contrived, but revised version on Web is better than code from Tuesday.
- Better example — Java library interface `Comparable` and its use in `Arrays` class.
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.)
- 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. You could add pad2game if you wanted to.
- 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.

Generics, Revisited

- Java library includes classes for collections of things (Vector, 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 in Java 1.5 (a.k.a 5.0), there are “generics” — Java’s answer to C++ template classes, though not exactly the same. Idea is to allow you to specialize a collection — so, a Vector of Integer objects only, or a Vector of Account objects only, etc., etc.
- Let’s do an example …
- Let’s also look at API for MainFrame in the game framework …
Minute Essay

- Write Java code to create a `Vector` to hold `Strings`, put into it `Strings"hello"` and "goodbye", and print both `Strings`.

Minute Essay Answer

- These lines would work:

```java
Vector<String> v1 = new Vector<String>();
v1.add("hello");
v1.add("goodbye");
System.out.println(v1.elementAt(0));
System.out.println(v2.elementAt(1));
```