Slide 1

Administrivia

- "Design" phase of Homework 1 due today (at 11:59pm). If you have last-minute questions, try e-mail. I'm likely to be lenient about late penalties for this assignment.
- "Code" phase due Tuesday. There should be time to sort out last-minute questions Tuesday (if not earlier via e-mail or office hours).

Slide 2

A Few More Java Basics

- `final` modifier:
  - For variables — can't change value after initialization (so constants are usually `final` and `static`).
  - For methods — can't be overridden by subclasses (more about that later).
  - For classes — can't be subclassed (more later).
- Some classes are "immutable" — once created, objects can't be changed. (Why is this good/bad?) Example — `String` — if you look at the API, you notice that methods that "change" the string actually return a new one.

CSCI 1321 September 11, 2003

Slide 3

Inheritance, Revisited

- Recall two roles from "short version" earlier — code reuse, subtypes.
- Recall that classes form a hierarchy/tree (with `Object` at root).

Slide 4

Inheritance and Code Reuse

- If class `Shape` defines

  ```
  private String colorName;
  public String getColorName();
  ```

  then if `Circle` is a subclass of `Shape`, `Circle` also has variable `colorName` and method `getColorName`.
- This can be a good way to reduce code duplication.
- If it's not what you want, subclasses can "override" methods (or variables — but this is not usually a good idea). In C++, this may require "virtual functions"; in Java, all functions/methods are virtual.
- Or a superclass can leave methods unimplemented; subclasses must then define — for `Shape`, `area()` could be abstract.

Inheritance and Subtypes

- In the "shapes" example, class `Shape` defines a type, and `Circle` and `Rectangle` are subtypes. Anywhere we need a `Shape`, we can use a `Circle`.

  ```
  Shape s = new Circle();
  ```

  (but not `Circle = new Shape()`)

Slide 5

Multiple Inheritance Versus Interfaces

- What if you want a class to inherit from multiple classes? C++ allows this ("multiple inheritance"). To avoid possible confusion/ambiguity, Java doesn't.
- Instead, define "interfaces" — classes in which *all* methods are abstract.
- In `Shape` example, we could define a `HasColorName` interface with methods `getColorName` and `setColorName`.
- A class can "implement" as many interfaces as you like.

Slide 6

Interfaces and Types

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

  ```
  HasColorName o = new Shape();
  ```

- 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 7

More Code

- We could write a class `TestShapes` to test `Shape` and its subclasses . . .

Slide 8

Minute Essay

- Try writing a `Square` class to fit in with the `Shape` example.