Slide 1

Administrivia

- Slides from class will be on Web — preliminary version shortly before class, final version usually later that day. The final version will include an answer to minute essays for which there is a right answer (such as the one today).
- Example code from class will also usually be on the Web sometime after class, linked from the “Sample programs” page (here).)
- Practice problems on the Web, linked from the “Useful links” page (here).)

Slide 2

Minute Essay From Last Lecture

- (Review question from last time’s notes.)
- Would it be better to store one point and width/length for a rectangle, or four points? Why?
- Should you store additional information such as area?

Slide 3

Java Basics, Review/Continued — Control Structures

- Most control structures are the same as C — `if` , `while`, `do`, `switch`, `for`, etc. Also a simplified `for`, as of Java 5.0 (a.k.a. 1.5), called “for-each”. More about it later.
- Also have “exceptions” — a way to deal with unusual or error conditions, break out of current flow of control. Can be “thrown” and “caught” (or not caught, in which case the program crashes). More about them later.

Slide 4

Packages and Importing

- Packages are simply a way of grouping related code and providing restricted scope for class names. Package names are (somewhat) hierarchical, with levels separated by dots — look at Java library API for examples.
- For classes in `java.lang` and current package reference using the class name only (e.g., `System`). For other classes, can use full name (e.g., `java.util.Vector`), or use `import`. (`import` looks like `#include`, but works differently.)

Packages, Continued

- You can define your own packages. Convention is to start with your e-mail/Web address, in reverse order (e.g., Dr. Lewis's framework is `edu.trinity.cs.gamecore`).
- Packages and filesystem hierarchy are related — for an example, create a package in BlueJ/Eclipse and then use another tool to look at the resulting directories and files.

Another Example

- Let's start sketching another example — `Account` class representing bank accounts.
- What variables seem useful? what methods?

Slide 7

UML Class Diagrams

- "Unified Modeling Language" — formal graphic representation of software analysis and design.

  Many types of diagrams, some of which you'll probably encounter in other courses. Tools exist for drawing them, but worth noting that they were designed to be whiteboard-friendly.
- We will mainly use class diagrams:
  - Box representing a class has name, attributes, operations.
  - Different kinds of arrows showing relationships among classes and interfaces.

Slide 8

Inheritance (Short Version)

- Given a class, it can be useful to define specialized versions — "subclasses".
- A subclass inherits attributes and operations from its superclass (which can in turn have a superclass . . .).
- Subclasses also form "subtypes" — e.g., if `PersonalAccount` is a subclass of `Account`, can use a `PersonalAccount` anywhere we need an `Account`.

Polymorphism (Short Version)

- “Many shapes” — something that works with many types.
- E.g., a function that works on `Account`s should work on `PersonalAccount`s, `CorporateAccount`s, ...

Inheritance and Code Reuse

- If class `Account` defines

  ```
  private String accountID;
  public void print();
  ```

  then if `CorporateAccount` is a subclass of `Account`, `CorporateAccount` also has variable `accountID` and method `print`.
- 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).
- Or a superclass can leave methods unimplemented; subclasses must then define (maybe differently for different classes). E.g., for `Account`, if we make `deposit` and `withdraw` abstract, each subclass must provide its own code.

Slide 11

Inheritance and Subtypes

- In the bank-account example, class `Account` defines a type, and `CorporateAccount` and `PersonalAccount` are subtypes. Anywhere we need a `Account`, we can use a `CorporateAccount` — e.g.,

  `Account s = new CorporateAccount();`

  (but not `CorporateAccount s = new Account()`;)
- Let's write more code for that example . . .

Slide 12

Minute Essay

- Sketch a method for the `Account` class that would allow you to pay interest — its name, parameters, possibly code . . .

Minute Essay Answer

- There are actually several things you might want to think about first . . .
- Is the rate the same every time you pay interest (monthly?), or does it change from month to month? (If it stays the same, maybe it should be a variable within the object; if it changes, maybe it should be a parameter to the method.)
- Is the rate the same for all accounts, or different for different accounts? if the former, it could be a class (static) variable.