Slide 1

Administrivia

- Linux accounts for new students should have been created, with passwords mailed to Trinity e-mail addresses. To change your password, `yppasswd` from the command line.
- If you're new to Linux and/or its command line: Links added to "Useful links" page to relevant information.
- Homework 1 design (description of your game) due Thursday at 11:59pm.

Slide 2

More Administrivia

- *Please* do not reboot the machines in this room (HAS 340); people rely on their being available for remote access.

  Also be careful not to inadvertently shut them down when trying to log off.

  If a previous user has left the machine's screen locked, use control-alt-backspace to restart the graphical subsystem.

Slide 3

About the Readings

- The textbook (*Java Software Structures*) is intended for a second-semester programming course, for people who already know Java.
- Since we teach our first-semester course in C — Dr. Lewis has been writing a "From C to Java" document.
- You should probably skim all assigned reading, but those with Java background will find some parts review.

Slide 4

A Little More About Homework 1

- Did you start reading the project description? Questions we should talk about now (briefly)?
- You're not committing yourself to anything at this point, but try to be as detailed as you can — so I can try to spot potential trouble. Also good to think in terms of a basic design (not too ambitious) plus extras. Keep in mind that what you do has to fit into an existing framework. (That's actually one of the pedagogical goals.)
- What you will actually turn in is HTML documentation of your planned game's main class — put it in your `Local/HTML-Documentation` and send me mail saying "ready to be graded". (Complete instructions in homework writeup.)

Slide 5

"Object Orientation"?

- A "programming paradigm" — contrast with procedural programming, functional programming, etc.
- No accepted-by-all definition, but most definitions mention encapsulation:
  - Data and functionality grouped together into "objects".
  - Some data/functionality is hidden.
- Origins in simulation/modeling, where the goal is to model complex systems consisting of many (real-world) objects.

Slide 6

What's An Object?

- Object — set of data (attributes) and associated functions (methods, behaviors, operations) that can act on data.
- Objects interact by calling each other's methods, or by sending each other messages.
- Often makes sense to have many similar objects — hence "classes".

Slide 7

What's a Class?

- Can be thought of as a blueprint for objects of a given type; individual objects are "instances" of the class.
- Defines attributes and methods each object will have (instance variables/methods), attributes and methods shared by all objects of a class (class variables/methods).
- Public interface — attributes and methods visible from outside the class.

Slide 8

Java and Object Orientation

- Java is not purely object-oriented — also includes "primitive types" for efficiency — but it's much more strongly object-oriented than a hybrid language such as C++.
- Java programs consist of definitions of classes. (No free-standing functions like the ones in C.)
- Java variables (except primitives) are references to objects, classes define types.
- Classes, attributes, methods have varying "visibilities" (from public to private).

Slide 9

Program Structure

- In Java, everything (variables and code) is part of a class. Typically have only one class per source code file (exception is inner/nested classes — more about them later).
- Any class can have a `main` method that can be launched by the runtime system (more about that later).

Slide 10

Defining a Class

- Each class is like a blueprint for objects of a particular kind, and can include:
  - Variables — instance (one copy per object) or static (one copy shared by all objects).
  - Methods — similar to C functions, but can be static or non-static ("instance methods"). Instance methods are "invoked on an object'.
  - Classes (more later).
- Variables and methods can be `public` or `private`. Good practice to define as private, except for constants that need to be used outside the class.

Slide 11

Naming Conventions

- Java library classes and methods follow these conventions:
  - If it's mixed-case and starts with uppercase, it's a class.
  - If it's mixed-case and starts with lowercase, it's a variable or method.
  - If it's all uppercase, it's a constant.
- You should follow them too, so your code will be easier for experienced Java programmers to read.

Slide 12

Tools

- Java programs are text, so you can write them with a text editor and compile and run them from the command line. (In fact I often do.)
- However, many professional programmers use an IDE (Integrated Development Environment), so we will too. It's Eclipse, and open-source, so you should be able to install a copy on your home machine if you like.

Slide 13

Example(s)

- Let's write a "hello world" program.
- We'll use Eclipse to
  - Define a project, a package, and a class with a `main` method.
  - Compile and run.
  - Generate HTML documentation.

Slide 14

Compiling and Running Programs — Java Versus C/C++

- With C/C++, your program ("source code") is transformed by a compiler into . . .

  "object code" (different for different processors), which is combined with library object code to produce . . .

  an "executable" (different for different operating systems) that can be run like other applications.
- With Java, your program (source code) is transformed by a compiler into . . .

  "byte code" (same on any processor), which is executed by . . .

  "Java virtual machine" (which has access to library byte code).

Slide 15

Java Basics — Recap

- Java programs consist of classes. Each class can contain
  - Variables — instance and static.
  - Methods — instance and static.
  - Classes (more about this later).
- Variables and methods can be `public` or `private`.
- Variables and methods can be `final`. (Use `static final` for constants.)

Slide 16

Variables

- Primitive types provided for efficiency (not purely object-oriented):
  - `boolean`, `short`, `int`, `long`, `float`, `double` are pretty much as in C.
  - `char` is 16-bit Unicode.
  - `byte` is 8-bit byte.
- All other variables are *references to objects*, similar to pointers:
  - `MyClass x` creates a *reference*, not an object — use `new` to create objects.

    Type of `x` is `MyClass` (just as type of an `int` variable is `int`).
  - Value of `null` means it doesn't point to anything.

Slide 17

Java Syntax

- Basic syntax based on C — variable declarations, method definitions, expressions — with some additions (as discussed in class and in "From C to Java").
- (This was by design.)

Slide 18

Creating and Deleting Objects

- Create object of class `MyClass` using `new` operator, e.g.,

  ```
  MyClass x = new MyClass();
  ```

  This object contains its own copy of all instance variables defined in `MyClass`.

  `new` above invokes no-parameters constructor for `MyClass`. Can have additional constructor(s) with parameters as desired.
- No need to explicitly free/delete objects — Java has "garbage collection". (This may not seem remarkable unless you've used a language without it — e.g., C or C++.)

Slide 19

Referencing Objects, Variables, and Methods

- Within `MyClass`, reference members of class (variables and methods) using just their names. If you have multiple objects of this class, which one is meant? "current object".
- In code using `MyClass`, reference as, e.g., `x.foo(parameters)` for instance methods, and `MyClass.staticFoo(parameters)` for static methods.

  Similar syntax for variables, but likely to be used less, since variables are normally private. (Exception is constants.)

Slide 20

Passing Parameters

- Syntax is like C.
- *Everything is passed by value* — but for reference variables, copying just creates two pointers to the same object, and the called method can change the object.

  (More about this later.)

Slide 21

Comments

- Can use C-style comments, C++-style comments.
- One type of C-style comments are special — "documentation comments" or "Javadoc comments". These start with `/**` and end with `*/`, and the command-line tool `javadoc` turns them into HTML documentation similar to what Sun provides for the library functions.
- Use documentation comments to describe what people using your class need to know. Use other types of comments to document code itself — something that would be useful to humans reading it.

Slide 22

Minute Essay

- Was there anything today that was particularly unclear?
- If you have programmed in Java before, what tool(s) did you use?