Slide 1

Administrivia

- Reminder: Project requirements analysis due today (5pm). Accepted without penalty through 5pm Friday; after that, 10% off per working day.
- Reminder: Ethics presentations next week. Topics assigned by e-mail to group leaders. (Some) details on Web. Questions?

Slide 2

Project Status / Next Step

- At this point you should have a reasonably clear idea of what you want your project/program to do.
- Now what? Code? Probably not — probably better to do some planning ("design") first. SO, next project milestone is a design for a prototype implementation, preferably an object-oriented design (a chance to practice that!)
- So . . .

  (Credit where credit is due, and a note about where to read more:

  Most material from today's lecture comes one of the books mentioned in the syllabus, Page-Jones's *Fundamentals of Object-oriented Design in UML*. Nice book — the author has a sense of humor, and it shows.

  Additions to "useful links" page coming soon.)

Slide 3

What is "Object-Oriented" Anyway?

- Ask a dozen experts, and you may get a dozen answers — different lists of characteristics / properties / principles.

  (Maybe this is "I know it when I see it, but I can't define it"?)
- On just about everybody's list — encapsulation.
- Can get a sense of what this is about, though, by reviewing list of key ideas from Page-Jones's book.
- Also may be useful to think about an example familiar to most of you — Dr. Lewis's PAD II game framework.

Slide 4

Encapsulation

- What is it? grouping related ideas into one unit, which can thereafter be referred to by a single name.
- Idea has a long history, going back to first work on packaging frequently-repeated code as a module / procedure / subroutine / function / method.
- In object orientation — packaging operations and attributes into an "object type", so attributes are modifiable only via interface provided by encapsulation.

Slide 5

Information/Implementation Hiding

- What is it? use of encapsulation to restrict some information or implementation details, so they're not visible "from outside".
- Also has a history — "modularity".
- Key benefit — isolates parts of a system from each other.

Slide 6

State Retention

- What is it? idea that objects, unlike procedural modules, have "state" (attributes — in Java terms, values of instance variables).
- Also has a history — "abstract data type".

Slide 7

Object Identity

- What is it? each object (regardless of class or current state) can be identified and treated as a distinct entity — each has an "object handle" (in Java terms, a value for a reference variable).
- Idea is that all other parts of the system can communicate with object via its "handle".

Slide 8

Messages

- What is it? vehicle by which sender object O_1 conveys to target object O_2 a demand for O_2 to apply one of its methods (in Java terms, an invocation of an instance method).
- Message includes target object's handle, name of operation it should execute, parameters/arguments if needed.
- Can think in terms of different kinds of messages — informative ("here is some info about something that happened"), interrogative ("give me some info"), imperative ("do something to yourself").

Slide 9

Classes

- What are they? stencils from which objects are “instantiated”.
- (You’re probably familiar with the distinction between classes and objects from Java.)

Slide 10

Inheritance

- What is it? using Java terms, facility by which a subclass has implicitly defined in it all attributes and operations of superclass, as if they were defined in subclass itself.
- (This also should seem familiar from Java.)
- One benefit is less (or no) duplication of code.
- Sometimes seems to make sense, if classes fall into some sort of hierarchy.
- Some situations seem to call for “multiple inheritance” (e.g., if modeling a university, a student worker is both a student and an employee).
- Use with care, though — often misused, especially multiple inheritance.

Slide 11

Polymorphism

- What is it? two definitions:
  - Facility by which a single operation or attribute name can be defined in more than one class, with different implementations.
  - Property by which an attribute or variable can point to objects of different classes at different times.

  (These also should be familiar from Java.)
- More terminology:

  Overriding — redefining a class's method in a subclass.

  Overloading — multiple class operations with the same name.

Slide 12

Genericity

- What is it? construction of a class so that one or more classes it uses internally are supplied only at instantiation time.
- Obvious example of where this is useful — container classes (linked lists, hash tables, binary trees, etc., etc.).
- (Aside: Java 5.0 (a.k.a. 1.5) includes lots of new and useful support for this idea — Java answer to C++ templates, though not entirely equivalent.)

Slide 13

How to Approach Object-Oriented Design

- First step is almost surely to decide how to encapsulate needed operations and attributes — i.e., identify classes.
- Usually useful to think in terms of real-world (or semi-real-world) objects being simulated / modeled / used.
- Example — Lewis game framework.

Slide 14

Kinds of Classes (One View)

- Foundation classes — fundamental (integer), structural (list), semantic (date, time, money, point).
- Architecture classes — for communicating with machine, manipulating database, HCI, etc. — "utility" classes useful in many applications.
- Business-domain classes — attribute (bank balance), role (customer), relationship (account ownership).

  (Probably a lot of the classes you design will be in this domain.)
- Application-domain classes — event recognizers, event managers. (This part isn't clear to me either!)

Slide 15

Minute Essay

- None — sign in.