

COMPUTER PROGRAMMING IN JAVA

COLUMBIA UNIVERSITY HIGH SCHOOL SCIENCE HONORS PROGRAM

Object Oriented Programming, Part 2

Lecture

2008 Apr 05

Review

Object Oriented Programming basics

- Classes
- Objects/instances
- Methods and constructors

Basic software principles

- abstraction
- encapsulation
- modularity
- reuse

Public versus Private

Variables and methods which are marked “public” means that other classes can access them; private variables and methods are inaccessible to other classes. Thinking about getters and setters, clearly we should make the class variables private and the getter and setter methods themselves public.

Static versus Non-static

A variable or method that is marked “static” means that it is a property of the entire class itself; non-static variables and methods are specific to a single instance.

Suppose we are designing a Person class, and one thing we want to keep track of is the total number of Person instances. This is an example where we could use a static variable to count.

Inheritance

The OO programming paradigm allows for a technique called inheritance. Inheritance is exactly what it sounds like: you have some “thing” which is somehow related to another “thing” in such a way that the “properties” of one is “passed down” to the other.

This is a very natural concept that everybody intuitively grasps. Consider any hierarchy, let's say of animals. We'll start at mammals. Represent this as a class, “Mammal”. What are some properties of all mammals? Now name some specific mammals. Are there any properties that they share?

In OO terminology, classes can inherit from other classes. A class can be both a parent class or a child class (synonyms: superclass and subclass).

At the root of the Java class hierarchy is the *Object* class. All other classes implicitly inherit from *Object* if they don't explicitly inherit from another class.

Methods and data members are also inherited (assuming you declare it correctly; we'll get into this later). Inheritance has a slightly different meaning in each case: when a method is inherited, it means that all subclasses automatically inherit this method. When a data member is inherited, it means that this variable automatically exists in every instance of that subclass (important: remember what it means to have multiple instances of a class!).

Now how do references work with inheritance? Answer: any reference can refer to any instance that is the same type as it, or if any of the subclasses of that instance match the type of the reference.

In Java, we are restricted to single inheritance: that is, any class can only inherit one class (C++ for example does not have this restriction; it (in)famously supports multiple inheritance). To provide the same functionality and flexibility as multiple inheritance, Java has the concept of interfaces, which we will also get into later.

Polymorphism

The other major concept OO brings to the table is polymorphism. Polymorphism

is essentially the ability for the system to dynamically (at runtime) determine what method to call.

Polymorphism is a very powerful concept whose use often results in clean, easier to debug, and easier to maintain code.

In Java, the primary way to achieve polymorphism is through overriding a method.

For example, thinking about shapes, we see that we can now have a base class Shape with a method “calcPerimeter()”, and the child classes Triangle, Circle, and Rectangle each override it in their own way. Then, given an array of Shape objects, Java will know which calcPerimeter() method to use.

An example of overriding methods is the toString() method, which is defined in the Object class, and which all child classes (essentially all classes) are free to override as needed.