

COMPUTER PROGRAMMING IN JAVA

COLUMBIA UNIVERSITY HIGH SCHOOL SCIENCE HONORS PROGRAM

Object Oriented Programming, Part 2

Lecture

2007 Nov 17 Sat

Review

The OO paradigm

Classes

Constructors

Objects/instances

Getters/Setters

Driver/main method

Inheritance

The OO programming paradigm allows for a very powerful 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” or inherited by 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? [Discuss]

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). [Construct a class hierarchy together]

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. That is, you can do the following [example]. 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 unfortunately we will not have time to go into.

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. [example]

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

In Java polymorphism is exhibited in two basic ways: overriding and overloading a method.

Overriding a method is essentially modifying a method that exists in the parent class. For example, given a Shape class, we have a method that returns the area. Each child class (Triangle, Circle, etc), then overrides this method with its own way to calculate the area.

Overloading a method is essentially adding multiple methods with the same name, but different number of arguments. For example, again using the Shape class, we can have a method setColor, that can take in either 1 color, or 2 colors. Taking two colors then gives a mix of the colors.