1/29 - Encapsulation 2 and Polymorphism/inheritance

Thursday, January 29, 2015

2:04 PM

Encapsulation part 2: separate exposed behavior

* A class should mark all of its instance fields as private so that code outside the class doesn't screw things up or change these field values unexpectedly
* The exposed stuff, including getters, setters, and constructors, is called the **interface**
  + The interface allows you to alter an object
* Make constructors as public for now in the class
* Interfaces can be declared by themselves and go in their own .java file
  + Should have the same name as file
  + Body of interface is just a list of method signatures that can be implemented or applied to a class
  + Ex:

public interface InterfaceName {

Type method1(parameters);

Type method2(parameters);

//Etc…….

}

* Then if you want to make a class that uses or implements this interface, you do this:

public class ClassName implements InterfaceA, InterfaceB {}

* Interface naming must be different from class names that use the interface
  + Convention A
    - Start all interface names with "I" for interface
    - Ex: Itriangle, Ipoint
  + Convention B
    - Name the interface generally, and then the classes that implements this interface as more specific
    - Ex: with an interface called Triangle, classes that implement this could be called PointTriangle, AngleTriangle, PolarTriangle
* **An interface does not have an implementation!!!**
  + **This means you have to attach this interface to an actual class that implements the methods listed in the interface**
  + **Thus when you create a new object, you have to do this:**

Point p1 = new PolarPoint(x,y,z);

* You can't say 'new Point(blah blah blah)', because 'Point' is just an interface, not a class!
* **So ideally, when you define a class, you should really put everything that is exposed(public) within an interface, then implement that interface through a class**
  + **Everything that is private should be marked private**
* What to make private, what to make public
  + Public: class declaration, interface methods(getters/setters), constructors
  + Private: fields, helper methods for internal behavior

Polymorphism = many forms

* General principle of providing access to an abstraction or method in many forms
* In Object Oriented programming, polymorphism can exist in a number of places:
  + Constructor overloading, method overloading
* If you don't define a constructor, java will create an invisible/implicit default constructor with no arguments.
* **Constructor Overloading**
  + Creating multiple versions of a constructor that may create different versions of an object
  + The way to distinguish between them is in the number or type of the parameters
  + The name of the constructors are all the same, but the parameters will differ
  + Once you define at least one constructor, there is no longer an automatically created one by Java
* You can also call other constructors within one constructor, which is called constructor chaining
  + To do this, the first line of code in the constructor must be the '*this'* keyword used as if a function with parameter
    - Form: this(parameter1, parameter2, etc);
    - Ex:

public Triangle(Point[] points){

//can't put anything here

this(points[0], points[1], points[2]);

//any code here will run after the call to the chained constructor

}

* Method overloading
  + Works the same as constructor overloading, same name but different parameters so the compiler knows which one to use
  + Usually the return type is still the same
  + Method type must be the same among them: all must be static or non-static
  + One version can call another(chaining)
    - No restrictions like constructor chaining