2/19

Thursday, February 19, 2015

2:06 PM

* a subclass has access to public and protected fields and methods
* subclass constructors:
  + should call superclass constructor with super() as first line
    - or, chain to a different constructor
    - or, rely on the implicit call to super() constructor with no parameters
* subclass can overload methods in superclass
  + meaning that a subclass can have a method that has the same name but diff parameters as a method within the superclass
* class Polymorphism
  + a class follows an "is-a" relationship with all classes and interfaces above it
  + covariance(up casting)
    - treating an instance of a subclass as if it were a parent class
    - can be checked at compile type
  + contravariance(down casting)
    - forcing a parent class instance into being a subclass
    - cannot be type checked in advance at compile time
      * Ex:

instanceOfSubA = (SubA)instanceOfA;

* problem: what should happen when an overridden method is called on a covariant reference?
  + AKA: when you upcast a subclass into a parent class, but call a method on the new casted object that was overriden by the subclass
  + two possibilities:
    - if methods are non-virtual, then the method will be called on whatever type the object is(so if a subclass is upcasted into a superclass, then the superclass's version of the method is called)
    - virtual: the overriden method is called, no matter what the object is casted into
  + C++,C# use non-virtual solution, and programmers can force virtual solution by marking a method with virtual
  + Java: all methods are virtual, no special keyword needed

* there is a drawback to "always virtual" approach
  + consider the situation in which a subclass just needs a method to do just a little more
    - in other words, it wants to execute a mtehod as defined in the superclass and then tweak the result, but can't do it because in virtual langauges, the subclass can't d
  + Ex:
    - class C extends B{
      * public int m(){ return ((B)this.m()) +1};
        + when you call the m() method on a casted B, it calls back to the m() method defined in class C, so it's an infinite loop
  + solution: the super keyword
    - this tells java to only use the superclass's method, not the overridden method
* use inheritence when a subclass uses common internals, including both fields and methods
  + if there's not common implementation between subclasses of a specific method within an interface, but the rest of the methods are all common , the subclasses have to say they extend the parent class and implement the interface as well
  + or, do this by using the 'abstract' keyword within the parent class
    - Ex:

abstract public void speak();

* then, subclasses then can implement speak in their own way and allows the parent class to abide by the interface
* 'abstract' = any subclasses must provide the missing implementation of this method