Inheritance and Polymorphism – 5 classes

Day 1

* Introduce ideas of object composition.
* Class designs a show dog class that is derived from the normal dog class.
* What behaviors/fields should a showdog do that a regular dog cannot?
  + Methods – prance, tapdance, jump through hoop.
  + Fields – quality\_of\_breed, points\_earned, beauty, prize\_won, levelOfAgility, arraylist of skills.
* Make showdog class.
  + Make sure it has at least one new method, and one new instance variable.
  + When we write the method, make it use the name of the dog.
  + But how do we access the name? (can use getname, or make protected in base class).
* Class does parrot, petparrot example.
* Comments from s17:
  + Do not do constructors! Make sure dog and parrot don’t have constructors.
  + When showing dog class and turning into showdog, first make showdog completely empty! Then write a dog in main and a showdog, illustrate how showdog can do everything that a dog can do.
  + Then start adding things to showdog.

Day 2

* Review isa/has-a.
* Review terminology.
* Review Dog/showdog, parrot/petparrot class from other day, including protected.
* Main goals – overriding and constructors.
* Show overriding.
  + Override toString() in showdog, to display number of trophies.
  + Have pet\_parrot override fly(), so pet parrots fly by using their names.
  + Talk about the keyword super.
  + Override speak in ShowDog so after the dog speaks regularly (woof woof), it then speaks in French.
* Add constructors to dog.
  + First add no-arg constructor to dog with print statement.
  + Add no-arg constructor to showdog with print statement.
  + Look at the order.
  + Change constructor in dog to take name.

Class lab work – do cars example. (on slides).

Day 3 (changed to lab day, so this is all old)

* Take 5-10 mins to finish up cars example.
* [Ice cream inheritance is more of the same, so skip if we lose a day for urcas and go straight to polymorph]
* Bring back parrot example.
* With pointers, show

Day 4 (done after lab, so they've seen a little bit of polymorphism)

* Introduce idea of polymorphism.
* From Greek πολύς, polys, "many, much" and μορφή, morphē, "form, shape."
* ***The ability for a derived class to substitute in code where a base class is needed.***
* Start with dog/showdog
* construct 3 dogs
  + make all dogs speak
  + Emphasize how the third dog is declared as a DOG reference, but uses the SHOWDOG functions.
* Code for dogs speaking, code for dogs talking. Mention why speak is different than dance. Emphasize why the polymorph dog can speak but not dance. Might show casting here.
* Polymorphism appears in multiple places.
  + So you can declare an object to be one class and substitute a derived class.
  + **You can also call functions that expect base classes and pass in derived classes**.
  + Demo speaktwice.
  + If you have a function that returns a base class, you can also have the function return a derived class.
  + Demo createRandomDog.
* Point out that with polymorphism, we are changing our understanding of function calls a little.
  + In 141, and up to this point in 142, whenever we called a function, we always knew ahead of time what code would be run. We could say "Java will jump to this specific line of code and start executing this specific function."
  + But now we can't say this anymore. We literally don't know ahead of time which function this is going to call.
* Another place to see polymorphism is in objects that hold other objects.
* Go through dog park example.
  + dancing example - show instanceof and casting.
  + Make a stuck up dog that will only play with other show dogs.
  + Show instanceof for dogs to detect only playing with showdogs.
* Introduce the Object hierarchy.
  + If you don't specify anything after extends when creating a class, Java assumes you are inheriting from Object.
* Object
  + String, Scanner, Color
  + ArrayList derives from other stuff first
  + toString, equals (show the little override symbol)

ABSTRACT CLASSES

* Skip to poly2
* We introduce the Pet class.
  + Go through pet, dog, showdog, cat.
  + Go through MyPets
    - arraylist of Pet.
    - But I want pet to speak! Why can't it speak?
    - (because animal doesn't have a speak method, it only lives in Dog/Cat)
    - Solutions? Use instanceof to figure out whether each one is a dog or cat,

and cast it, and then call speak.

* Or, declare speak in animal.
* But then what does speak in animal DO?
* NEW CONCEPT: As we examine the object hierarchy in any language, not just java, as you go higher and higher up, the classes become more and more abstract --- they represent more general concepts.
* Like Pet is more general than Dog and Cat, Dog is more general than ShowDog. This is the way inheritance works, because it expresses an IS-A relationship. A Dog IS-A animal, Cat is-a pet, etc.
* Often we reach a point in the object hierarchy where we find a class that is SO general, we want it to have a certain method so its subclasses will inherit it, but we can't write code for the method in the base class because each subclass does it differently.
* Speak fits this idea. We know every Pet (at least common ones) have a way to speak (or make some kind of noise). But they all do it differently. What sort of code should we even have inside this pet.speak() method?
* We can't. What we'd like to do is tell Java, "Hey, all Pets speak, but each one does it differently. I need to tell this Pet class that I want each Pet to be able to speak, but each subclass of Pet should write its own speak method.
* Java has a special way to handle this situation:
* This can be done with something called an ABSTRACT method.
* When you declare a method ABSTRACT, it means that the method is so general, we can't write the code for it here in the base class. Instead, we will delegate the responsibility of writing that method to each derived class. Derived class must override abstract methods and provide their own code.
  + Show syntax.
  + The whole class must now be marked ABSTRACT.
  + And now the class cannot be instantiated.
  + The opposite of ABSTRACT is CONCRETE.
* Go through shape example. Shape getArea, getPerimeter.
  + Circle, triangle, rectangle.
* Instrument Example:
  + Instrument with play() method.
  + piano, guitar, flute, etc.