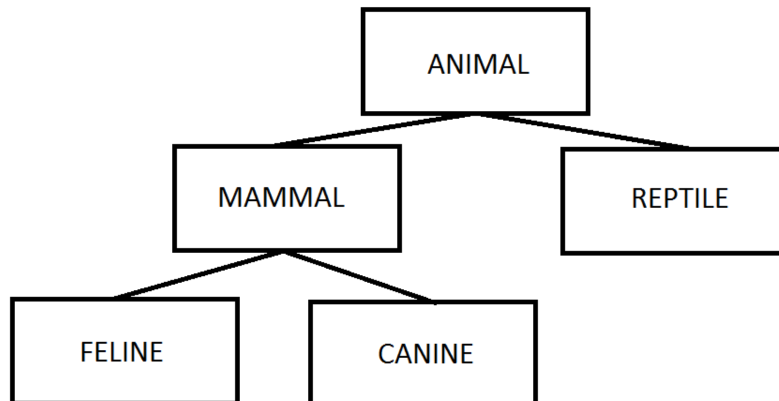
Brian Ahmann, Christian Schulz, Bryan Rose
CSSE 221 – Section 1
Dr. Delvin Defoe
16 September 2013

Inheritance

Inheritance is the interactions between super and sub classes. Each class can only have one super class, but can have any number of subclasses. A class uses the extends keyword to represent inheritance. The super-sub relationship is represented with this chart:



As you can tell, inheritance has an "is-a" relationship: A mammal "is-an" animal, and a feline "is-a" mammal. In this example, the Mammal class would inherit the public and protected fields and methods of Animal, and the Feline class would inherit the public and protected fields and methods of Mammal, including those from Animal. Constructors are also inherited through the use of "super()" in the subclass's constructor. (In other words, one can call super() within the child's constructor to inject the code from the parent's constructor). [Note that the super(…) call must be the first line of code in the subclass's constructor.]

Instead of using an inherited method, the subclass can override that method. This is done by creating a method in the subclass with the same descriptors, types, parameters, and name as a method in the superclass but a different body. Put the "@Override" notation before the new method. Then, in the subclass, whenever the method is called, it will go to the new version. To access the superclass's version, simply put a "super()" before the call (super.methodCall()). This does not work with static, or class, methods, as described in the next section.

If you attempt to override a static method, instead it is "hidden". If the superclass itself is referenced, the superclass's version of the method is called, whereas an overridden method would still call the subclass's method.

When creating objects, it is possible to create something with only partial capabilities through casting, i.e. Animal ani = new Mammal(). This will result in ani only being able to call methods that have already been created in Animal, and not methods created in Mammal. For example, if the method animalSound was the only method in Animal, and Mammal overrode animalSound and created furColor, ani would only be able to call animalSound and not furColor. This is because ani is in fact an Animal object reference and not a Mammal object reference. However, the constructor called for ani is in fact Mammal's constructor.

Sometimes a method will take an Animal as one of its parameters (e.g. weapon.hunt(Animal ani)). Instead of passing a Mammal directly, one must use casting, e.g. if mamm is a Mammal, call weapon.hunt((Animal)mamm);

Source: Oracle pages The Java Tutorials
http://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html