Java Inheritance In-Depth

Superclass vs Subclasses
Protected Variables
Inheritance vs Composition

Inheritance

- Recall that classes can inherit from one another
  - A Textbook is a type of Book
    - public class Textbook extends Book
  - A Chihuahua is a type of Dog
    - public class Chihuahua extends Dog

Subclasses have access to superclass data/methods

```
class Dog {
   public void bark() {
      System.out.println(“Bark bark!”);
   }
   // Cannot call .yip() on a Dog object / ref
}

class Chihuahua extends Dog {
  public void yip() {
      System.out.println(“Yip yip yip!”);
  }
  // Can also call .bark() on a Chihuahua object / ref
}
```

Overriding and Overloading

- You can override a method on a subclass, which will cause that method to run *instead* of the one on the superclass

```
class Chihuahua extends Dog {
   public void bark() {
      System.out.println(“high-pitched barking”);
   }
}
```

Note that overriding is different than overloading, which is using the same method name with different parameter lists.

Overloading Example

```
public class Addition {
    public static int add(int a, int b) {
        return a + b;
    }
    public static float add(float a, float b) {
        return a + b;
    }
    public static int add(int a) {
        return a;
    }
}
```

super - Using Superclass methods in a subclass

- What if we wanted to use the superclass's method, but also add a little bit on to it?

- We can use super to gain access to the superclass's methods!

super()

- If you want to use super in a constructor, there is a special form, super(). Otherwise super.methodName().
- super() must be the first statement in a constructor!
- super() can contain arguments, e.g. super(7);

instanceof

- You can check an object's type with instanceof
- It will return true if that object is an "instance of" a class, false otherwise
- Note that it checks the OBJECT, not the reference!

Why is this useful?

- Getting something from an ArrayList (which can any object), we can check if it's the kind of object we want! Otherwise, if we just use explicit casting, it could break.

- Note: There's ANOTHER, EVEN COOLER way to do work around this you'll be learning about soon, called Generics. STAY TUNED!

protected

- So far, we have only worked with public and private methods / attributes, which works fine.
- Can subclasses access private variables? NOPE
- But sometimes we want subclasses to have access to “private” variables and methods of superclasses.
- We can make these *protected*.

protected means “private, except for subclasses*”

Kind of in-between public and private - “private except for certain circumstances”

Imagine my son is a subclass of me - I am OK with giving him the keys to my house, but not any of you (no offense)

```
protected int _numKeys = 6;
```

protected in packages

- A slight caveat: protected variables are also accessible by other classes in the same package. Since we haven't discussed packages, yet, don't worry about this right now, but know that there is another exception coming!

Inheritance vs Composition

- Inheritance - an “is-a” relationship (extends)
- Composition - a “has-a” relationship (variables)

Example

- Is-A
  - A St. Bernard is-a Dog
  - A TextBook is-a Book
  - A Professor is-a Human (probably)
- Has-A
  - A Human has-a Name
  - A Hand has-a Set of Fingers
  - A Book has-a Set of Pages

Deciding Whether To Use Inheritance vs Composition

- Depends on what you are using it for. In general, if something is a _kind_ of something else, use inheritance. If it contains or is linked to something else, use composition.