Lecture 4, Reviewing Java
Students' questions

Q. Can we add more methods and other objects in an extended class or can we only do super?
A. An extended class can add as much stuff as it wants. It benefits from the methods and variables of the class it extends. So, if I have a class A with method a1() and a2(), then the extended class B can use these methods a1() and a2() and also add methods such as b1() and b2(). Note furthermore that you can redefine methods of the class you're extending: for example, a `toString()` can be define for A but a different one defined for B. There is a special situation about protections. When something is `private` then it really does not leave the class: you don't have an access to it when you're extending. If you want to have the benefits of private so that people outside your class can't access, but you can do it if you're extending, then you use the type `protected`. Here's an example:

```
public class A{
        protected int value ;
        public A(int v){
                value = v ;
        }
}
```

```
public class B extends A{
        public A(int v){
                super() ;
                value *= 10 ;
        }
}
```

In this example, we have a class A of which the variable is protected. So we can create it like "`A blurp = new A(10)`" but it is forbidden to access the variable and do things such as `blurp.value = 20`. However, since it's protected, class B can use it and it chooses to multiply it by 10. In the extended class' constructor, you must initialize the class that you're extending and that's what `super()` does.

Q. What do you mean by "get something back from array" and "garbage"? And what are the advantages of giving array lists a type?
A. ArrayList is a Data Structure: essentially, it stores things and we retrieve them. As shown in the Javadoc, all elements in an ArrayList without a type will be considered of type Object; that's what I call garbage, you can really store anything you want in such an ArrayList. We know what is the type of an object when we store it into a list: for example, it can be our custom class A, or a type such as String. However, when we retrieve the element (*i.e.* get it back), we lost the type: the ArrayList contains only the type Object so that's what it returns to you. This is a problem because you may want to call some function specific to the object that you stored, but the compiler will tell you that if cannot guarantee if this function exists. Here's an example:

```
ArrayList myArray = new ArrayList();
myArray.add(new A(10));     // we add an object of type A, that has a method getValue
myArray.get(0).getValue(); //illegal: you didn't get back an A but an Object (getValue isn't defined on Object)
```

If all the things you add in an ArrayList are of the same type, then you should assign this type to the ArrayList. That way, when you get things back, they kept their type and you can call methods relevant to this type. So if you have `ArrayList<A>` myArray then you can retrieve the element and ask it its value, since this is a valid operations for instances of class A. ArrayList is a very common thing in Java so here are a few more examples to see it in action:

http://www.idevelopment.info/data/Programming/java/collections/ArrayListExample.java
http://www.anyexample.com/programming/java/java_arraylist_example.xml
http://www.java-examples.com/simple-java-arraylist-example
http://www.roseindia.net/java/beginners/array_list_demo.shtml
http://www.java-samples.com/showtutorial.php?tutorialid=234