Interfaces II

Interfaces in the Java API

We'll look at some interfaces in the Java API

As we said before, it could be useful to declare otherwise unrelated classes and express that they do share some behavior, such as the ability for objects of the class to compare themselves with objects of the same type.

Luckily for us, there is an interface in the Java API which solves this for us.

The interface is called java.lang.Comparable and it has the capability to be used by unrelated classes.

Let’s look at java.lang.String

The complete class declaration in java.lang.String is:

```
public final class String implements java.io.Serializable,
Comparable<String>, CharSequence{...}
```

We now know what the different parts of that class declaration means. It is a public class which cannot be extended (final) and it implements a few interfaces.

The interface we’re interested in here is Comparable<String>

Comparable‹T›

The source code for java.lang.Comparable starts with the following declaration.

```
public interface Comparable<T>
```

The new thing about that declaration is the <T> thing. It just means “I can be used together with specified types”.

The interface declares just one abstract method:

```
public int compareTo(T o);  // there’s the T again!
```

The ‹T› is our friend

The `<T>` in the interface declaration allows us to specify for which class we intend to use this interface. Remember that the java.lang.String said in its declaration “`implements Comparable<String>`”?

Java will use “`String`” wherever it said `T` in the interface `Comparable`.

So the version of `compareTo(T o)` which String needs to implement must then be:

```
public int compareTo(String anotherString) {...}
```

Good! We now have a compareTo() implementation in String which can compare a String object to another String object!

Since String objects are Comparable...

...We can do this:

```
System.out.println(args[0].compareTo(args[1]);
```

Or even this (explained soon!):

```
    if(args.length != 0){
      Arrays.sort(args);
      System.out.println(Arrays.toString(args));
    }
```

What does the int we get from compareTo() mean

The compareTo method returns an int which holds the result of comparing an object to another object.

If the result is negative, then the object is to be considered less than the other object. If the result is 0 they are considered to be of equal order. If the result is positive, the object is considered greater than the other object:

```
String s =     "Ape";
System.out.println(s.compareTo("Zebra"));  // -25
System.out.println(s.compareTo("Ape"));    //   0
System.out.println(s.compareTo("Abraham"));//  14
```

One good use of comparisons

A good use case for comparing two objects, is when sorting many objects.

In order to sort a list (like an array), we must have a way to compare every object with the other objects.

There is a static utility method in the java.util.Arrays class called sort() which uses the compareTo method in order to sort an array:

```
if(args.length != 0){
  Arrays.sort(args);
  System.out.println(Arrays.toString(args));
}
```

Arrays.sort can only sort comparable objects

The version of Arrays.sort() which takes a single argument of a reference to an array, can only sort comparable objects (objects which can compare themselves to another object of the same Type), like String.

Since String implements the Comparable<String> interface, we know for a fact (via the contract) that there is a compareTo() method in String. Also, we have cheated and looked ;-)

That’s why it’s no problems for Arrays.sort() to sort an array of String references!