

JAVA INTERFACES

=====

Java has an "interface" keyword which refers to something quite different than the interfaces I defined in Lecture 8, even though the two interfaces are related. Henceforth, when I say "interfaces" I mean public fields, public method prototypes, and the behaviors of public methods. When I say "Java interfaces" I mean Java's "interface" keyword.

- A Java interface is just like an abstract class, except for two differences.
- (1) In Java, a class can inherit from only one class, even if the superclass is an abstract class. However, a class can "implement" (inherit from) as many Java interfaces as you like.
 - (2) A Java interface cannot implement any methods, nor can it include any fields except "final static" constants. It only contains method prototypes and constants.

The distinction between abstract classes and Java interfaces exists because of technical reasons that you might begin to understand if you take CS 164 (Compilers). Some languages, like C++, allow "multiple inheritance," so that a subclass can inherit from several superclasses. Problems arise when two superclasses define differing methods or fields with the same name; but this is not the only problem that can occur. Multiple inheritance is responsible for some of the scariest tricks and traps of the C++ language, subtleties that cause much wailing and gnashing of teeth. Java avoids these problems by not having multiple inheritance, except in a very limited form (Java interfaces).

```
public interface Nukeable {           // In Nukeable.java
    public void nuke();
}

public interface Comparable {        // In java.lang
    public int compareTo(Object o);
}

public class SList extends List implements Nukeable, Comparable {
    [Previous stuff here.]

    public void nuke() {
        head = null;
        size = 0;
    }

    public int compareTo(Object o) {
        [Returns a number < 0 if this < o,
         0 if this.equals(o),
         > 0 if this > o.]
    }
}
```

Observe that the method prototypes in a Java interface may be declared without the "abstract" keyword, because it would be redundant; a Java interface cannot contain a method implementation.

Because an SList is a Nukeable and a Comparable, we can assign it to variables of these types.

```
Nukeable n = new SList();
Comparable c = (Comparable) n;
```

The cast is required because not every Nukeable is a Comparable.

"Comparable" is a standard interface in the Java library. By having a class implement Comparable, you immediately gain access to Java's sorting library. For instance, the Arrays class in java.util includes a method that sorts arrays of Comparable objects.

```
public static void sort(Object[] a)    // In java.util
```

The parameter's type is Object[], but a run-time error will occur if any item stored in a is not a Comparable.

Interfaces can be extended with subinterfaces. A subinterface can have multiple superinterfaces, so we can group several interfaces into one.

```
public interface NukeAndCompare extends Nukeable, Comparable { }
```

We could also add more method prototypes and constants, but in this example I don't.