Java Interfaces

Concept

- **`interface`** is a way to describe what classes should do, without specifying how they should do it.
- It's not a class but a set of requirements for classes that want to conform to the interface

```
public interface Comparable
{
        int compareTo(Object
  otherObject);
}
```

this requires that any class implementing the `Comparable` interface contains a `compareTo` method, and this method must take an `Object` parameter and return an integer

Interface declarations

- The declaration consists of a keyword `interface`, its name, and the members
- Similar to classes, interfaces can have three types of members
  - constants (fields)
  - methods
  - nested classes and interfaces

Interface member – constants

- An interface can define named constants, which are `public`, `static` and `final` (these modifiers are omitted by convention) automatically. Interfaces never contain instance fields.
- All the named constants MUST be initialized

An example interface

```
interface Verbose {
        int SILENT = 0;
        int TERSE = 1;
        int NORMAL = 2;
        int VERBOSE = 3;

        void setVerbosity (int level);
        int getVerbosity();
}
```

Interface member – methods

- They are implicitly `abstract` (omitted by convention). So every method declaration consists of the method header and a semicolon.
- They are implicitly `public` (omitted by convention). No other types of access modifiers are allowed.
- They can't be `final`, nor `static`

Modifiers of interfaces itself

- An interface can have different modifiers as follows
  - `public/package(default)`
  - `abstract`
    - all interfaces are implicitly `abstract`
    - omitted by convention

To implement interfaces in a class

- Two steps to make a class implement an interface
  1. declare that the class intends to implement the given interface by using the `implements` keyword

```
class Employee implements Comparable { . . . }
```

  2. supply definitions for all methods in the interface

```
public int compareTo(Object otherObject) {
     Employee other = (Employee) otherObject;
     if (salary < other.salary) return -1;
     if (salary > other.salary) return 1;
     return 0; }
```

note: in the `Comparable` interface declaration, the method `compareTo()` is `public` implicitly but this modifier is omitted. But in the `Employee` class design, you cannot omit the `public` modifier, otherwise, it will be assumed to have `package` accessibility

- If a class leaves any method of the interface undefined, the class becomes `abstract` class and must be declared `abstract`
- A single class can implement multiple interfaces. Just separate the interface names by comma

```
class Employee implements Comparable, Cloneable {. . .}
```

Instantiation properties of interfaces

- Interfaces are not classes. You can never use the `new` operator to instantiate an interface.

```
public interface Comparable {
        . . . }
Comparable x = new Comparable( );
```

- You can still declare interface variables

```
Comparable x;
```

but they must refer to an object of a class that implements the interface

```
class Employee implements Comparable {
                . . .
}
x = new Employee( );
```

Extending interfaces

- Interfaces support multiple inheritance – an interface can extend more than one interface
- Superinterfaces and subinterfaces

Example

```
public interface SerializableRunnable extends
java.io.Serializable, Runnable {
        . . .
}
```

Extending interfaces – about constants (1)

- An extended interface inherits all the constants from its superinterfaces
- Take care when the subinterface inherits more than one constants with the same name, or the subinterface and superinterface contain constants with the same name — always use sufficient enough information to refer to the target constants

- When an interface inherits two or more constants with the same name
  - In the subinterface, explicitly use the superinterface name to refer to the constant of that superinterface

E.g.

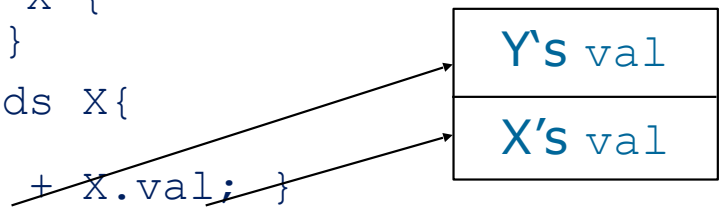
```
interface A {
        int val = 1;
}
interface B {
        int val = 2;
}

interface C extends A, B {
        System.out.println("A.val = "+
A.val);
System.out.println("B.val = "+ B.val);
}
```

- If a superinterface and a subinterface contain two constants with the same name, then the one belonging to the superinterface is **hidden**
  1. in the subinterface
     - access the subinterface-version constants by directly using its name
     - access the superinterface-version constants by using the superinterface name followed by a dot and then the constant name

E.g

```
interface X {
    int val = 1; }
interface Y extends X{
    int val = 2;
    int sum = val + X.val; }
```



  2. outside the subinterface and the superinterface
     - you can access both of the constants by explicitly giving the interface name.

E.g. in previous example, use `Y.val` and `Y.sum` to access constants `val` and `sum` of interface `Y`, and use `X.val` to access constant `val` of interface `X`.