

Interfaces, Lambda Expressions, and Inner Classes

Yangtao Ge

June 24, 2019

1 Interfaces

1.1 The Interface Concept

Definition: **Interface** is not a class but a set of *requirements* for the classes that we want to conform to the interface.

e.g.

```
public interface Comparable<T>{  
    int compareTo(T other); // para has type T  
}
```

Some Notice to Interface:

- all methods of an interface are automatically public – we don't add *public* in the signature
- interfaces can define constants
- interfaces cannot have instance fields
- method are never implemented in interface (we could now, but it's **BAD**)

To make a class implement an interface:

1. declare that your class intends to *implement* the given interface
2. supply definitions for all methods in the interface

e.g.

```
class Employee implements Comparable<Employee>{
    public int compareTo(Employee other){
        return Double.compare(salary , other.salary );
    }
    ...
}
```

N.B. Try to use *Generic type*, less using ‘type cast’.

Regarding to *compareTo* method:

- how to compare:
 - subtraction: if we know the maximum bounday is less than ‘maximum of Integer’.
 - compareTo: don’t care
- inheritance Problems: (solve like *equal()* method in Chapter 5)
 - different notations of comparison: add a same class test


```
if(getClass() != other.getClass){
    throw new ClassCastException;
}
```
 - common algorithm: provide a single *compareTo* method, and declare it as *final*

1.2 Properties of Interfaces

Some properties of Interface:

- Interfaces are not classes – can’t do ‘*x = new Comparable(...);*’
- we can declare interface variables

- can do ‘*Comparable x*;’
- can do ‘*x = new Employee(...)*;’ (since ‘Employee’ implements Comparable)
- we can check whether an object implements an interface by ‘*instanceof*’ keyword
- we can extend Interfaces

```
public interface Moveable{
    void move (double x, double y);
}
public interface Powered extends Moveable{
    double milesPerGallon ();
}
```

- we can add *constants* in the interface. This method is automatically ‘**public static final**’

```
public interface Powered extends Moveable{
    double milesPerGallon ();
    // public static final constant
    double SPEED_LIMIT = 95;
}
```

- classes can implement *multiple* interfaces – we can do ‘*class Employee implements Person, Comparable*’ (but one class can only have one superclass)

1.3 Interfaces and Abstract Classes

Key: A class can only extend a single class, but can implement several interfaces.

We can think it as:

- abstract classes: tends to stress what it is (inheritance – ‘is-a relationship’)
- interface: tends to illustrate what can it do (properties).

N.B. Remember it by – **things can only belong to one class, but it can have several properties.**

1.4 Static and Private Methods

We can add '*static* method' since Java 8, and '*private* method' since Java 9. This is not very useful.

Ref: p.306

1.5 Default Methods

some useful situation for *default* modifier:

- implement 'iterator': providing an exception

```
public interface Iterator<E>{
    boolean hasNext();
    E next();
    default void remove(){
        throw new UnsupportedOperationException(“remove”)
    }
}
```

- implement 'collection': call other methods

```
public interface Collection{
    int size(); // an abstract method
    default boolean isEmpty(){
        return size() == 0;
    }
}
```

- *interface evolution* for adding class in the future.

1.6 Resolving Default Method Conflicts