Creating Abstract Relationships with Interfaces

Jim Wilson

@hedgehogjim | blog.jwhh.com | jimw@jwhh.com

What to Expect in This Module

What an interface is

Implementing an interface

Implementing multiple interfaces

Declaring an interface

What Is an Interface?

An interface defines a contract

Provides no implementation

Classes implement interfaces

Expresses that the class conforms to the contract

Interfaces don't limit other aspects of the class' implementation

Implementing an Interface

java.lang.Comparable

Used for determining relative order

One method: compareTo

Receives item to compare to

Return indicates current instance relative sequence

- Negative value: before
- Positive value: after
- Zero value: equal

```
public class Passenger implements  Comparable {
  // others members elided for clarity
  private int memberLevel; // 3(1st priority), 2, 1
  private int memberDays;
  public int compareTo(Object o) {
    Passenger p = (Passenger) o;
    if(memberLevel > p.memberLevel)
      return -1;
    else if(memberLevel < p.memberLevel)
      return 1;
    else {
     if(memberDays > p.memberDays)
       return -1;
     else if(memberDays < p.memberDays)
       return 1;
     else
       return 0;
    }
  }
}
```

Implementing an Interface

```java
Passenger bob = new Passenger();
bob.setLevelAndDays(1, 180);
Passenger jane = new Passenger();
jane.setLevelAndDays(1, 90);
Passenger steve = new Passenger();
steve.setLevelAndDays(2, 180);
Passenger lisa = new Passenger();
lisa.setLevelAndDays(3, 730);

Passenger[] passengers =
  {bob, jane, steve, lisa};
Arrays.sort(passengers);
```

lisa steve bob jane

```java
public class Passenger implements Comparable {
  // others members elided for clarity
  private int memberLevel; // 3(1st priority), 2, 1
  private int memberDays;
  public int compareTo(Object o) {
    Passenger p = (Passenger) o;
    if(memberLevel > p.memberLevel)
      return -1;
    else if(memberLevel < p.memberLevel)
      return 1;
    else {
     if(memberDays > p.memberDays)
       return -1;
     else if(memberDays < p.memberDays)
       return 1;
     else
       return 0;
    }
  }
}
```

Implementing an Interface

```
public class Flight implements Comparable {
  // others members elided for clarity
  private int flightTime; // minutes past midnight

  public int compareTo(Object o) {
    Flight f = (Flight) o;
    if(flightTime < f.flightTime)
      return -1;
    else if(flightTime > f.flightTime)
      return 1;
    else
      return 0;
  }
}
```

Implementing an Interface

```
Flight lax045 = new Flight();
lax045.setFlightTime(45);

Flight slc015 = new Flight();
slc015.setFlightTime(15);

Flight nyc030 = new Flight();
nyc030.setFlightTime(30);


Flight[] flights =
  {lax045, slc015, nyc030};
Arrays.sort(flights);
```

slc015 nyc030 lax045

```
public class Flight implements Comparable {
  // others members elided for clarity
  private int flightTime; // minutes past midnight

  public int compareTo(Object o) {
    Flight f = (Flight) o;
    return flightTime - f.flightTime;
  }
}
```

What Is an Interface?

An interface defines a contract

Provides no implementation

Classes implement interfaces

Expresses that the class conforms to the contract

Interfaces don't limit other aspects of the class' implementation

Some interfaces require additional type information

Uses a concept known as generics

Implementing a Generic Interface

```
public interface Comparable<T> {
    int compareTo(T o);
}
```

```
public class Flight implements Comparable<Flight> {
  // others members elided for clarity
  private int flightTime; // minutes past midnight

  public int compareTo(Flight f) {
    Flight f = (Flight) o;
    return flightTime - f.flightTime;
  }
}
```

Implementing a Generic Interface

```
public class Passenger implements  Comparable<Passenger> {
  // others members elided for clarity
  private int memberLevel; // 3(1st priority), 2, 1
  private int memberDays;
  public int compareTo( Passenger p) {
    Passenger p = (Passenger) o;
    if(memberLevel > p.memberLevel)
      return -1;
    else if(memberLevel < p.memberLevel)
      return 1;
    else {
     if(memberDays > p.memberDays)
       return -1;
     else if(memberDays < p.memberDays)
       return 1;
     else
       return 0;
    }
  }
}
```

What Is an Interface?

An interface defines a contract

Provides no implementation

Classes implement interfaces

Expresses that the class conforms to the contract

Interfaces don’t limit other aspects of the class’ implementation

Some interfaces require additional type information

Uses a concept known as generics

Classes are free to implement multiple interfaces

Implementing Multiple Interfaces

```
public class Person {
  // other members elided for clarity
  private String name;
}
```

```
public class CrewMember extends Person {
  // members elided for clarity
}
```

```
public class Passenger extends Person
        implements Comparable<Passenger> {
  // members elided for clarity
}
```

```
public class Flight
   implements Comparable<Flight>{
   // others members elided for clarity
   private int flightTime;
   private CrewMember[] crew;
   private Passenger[] roster;
   public int compareTo(Flight f) {
     Flight f = (Flight) o;
     return flightTime - f.flightTime;
   }
}
```

Implementing Multiple Interfaces

```
public interface Iterable<T> {
  Iterator<T> iterator();
}
```

```
public interface Iterator<T> {
  boolean hasNext();
  T next();
}
```

```
public class Flight
  implements Comparable<Flight>, Iterable<Person> {
  // others members elided for clarity
  private int flightTime;
  private CrewMember[] crew;
  private Passenger[] roster;
  public int compareTo(Flight f) {
    Flight f = (Flight) o;
    return flightTime - f.flightTime;
  }
  public Iterator<Person> iterator() {


  }

}
```

Implementing Multiple Interfaces

```
public class FlightIterator
 implements Iterator<Person> {
  private CrewMember[] crew;
  private Passenger[] roster;
  private int index = 0;
  public FlightIterator(
   CrewMember[] crew, Passenger[] roster) {
    this.crew = crew;
    this.roster = roster;
  }
  boolean hasNext() {
   return index < (crew.length + roster.length);
  }
  public Person next() {
   Person p = (index < crew.length) ?
     crew[index] : roster[index - crew.length];
   index++;
   return p;
  }
}
```

```
public class Flight
  implements Comparable<Flight>, Iterable<Person> {
  // others members elided for clarity
  private int flightTime;
  private CrewMember[] crew;
  private Passenger[] roster;
  public int compareTo(Flight f) {
   Flight f = (Flight) o;
   return flightTime - f.flightTime;
  }
  public Iterator<Person> iterator() {
    return new FlightIterator(crew, roster);
  }
}
```

Implementing Multiple Interfaces

```
Flight lax045 = new Flight(45);
// Add crew members:
//   Pilot Patty, CoPilot Karl, Marshal Mary
// Add Passengers:
//   Bob, Jane, Steve, Lisa
for(Person p:lax045)
  System.out.println(p.getName());
```

```
Pilot Patty
CoPilot Karl
Marshal Mary
Bob
Jane
Steve
Lisa
```

```
Iterable<Person> laxIterable = lax045;
Iterator<Person> persons = laxIterable.iterator();
while(persons.hasNext()) {
  Person p = persons.next();
  System.out.println(p.getName());
}
```

Declaring an Interface

Declaring an interface is similar to declaring a class

Use the interface keyword

Supports a subset of the features available to classes

Methods	Constants	Extending interfaces
Name, parameters, and return type	Typed named values	An interface can extend another interface
Implicitly public	Implicitly public, final, static	Implementing extended interface implies implementation of base

Demo
Dynamically Extending CalcEngine

Summary

- An interface defines a contract
  - Provides no implementation
  - Can include methods and constants
- Classes implement interfaces
  - Classes are able to implement multiple interfaces
- Interfaces are able to extend other interfaces
  - Implementing an extended interface implicitly implements the base