Interfaces

Part1 – Interface Basics

Chapter 6, Core Java Volume I

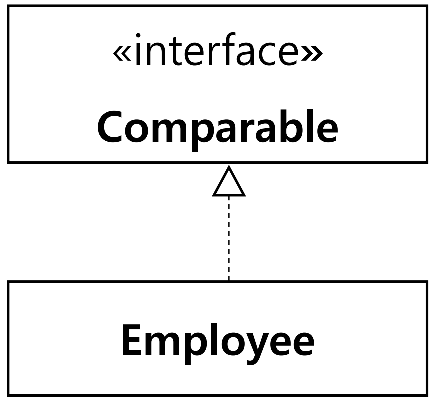
Chapter 10, Java How to Program, 10th ed.

Table of Contents

The Concept of Interface

- Interfaces (in general)
  - external views (behavior) of a class
  - contracts (a set of requirements) for a class

- Interfaces (in Java)
  - a set of abstract methods (a way to achieve abstraction)
  - a reference type



- A class can implement one or more interfaces.
  - The class conforms to the interfaces. (c*onforms-to* relationship)
  - The class is the subtypes of the interfaces. (*is-a* relationship)

- Confroms-to Relationship
  - Service provider: “If your class conforms to a particular interface, then I'll perform the service.”
  - Example: `Arrays.sort` sorts an array if the element class conforms to the `Comparable` interface.

The Concept of Interface

- Defines Comparalbe Interface :

```
public interface Comparable
{
   int compareTo(Object other); // public by default
}
```

```
// as of Java 5
public interface Comparable<T>
{
   int compareTo(T other);
}
```

- Implements Comparable Interface:

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

Listing 6.1: interfaces/EmployeeSortTest.java

```
package interfaces;

import java.util.*;

 // This program demonstrates the use of the Comparable interface.

public class EmployeeSortTest
{
   public static void main(String[] args)
   {
      Employee[] staff = new Employee[3];

      staff[0] = new Employee("Harry Hacker", 35000);
      staff[1] = new Employee("Carl Cracker", 75000);
      staff[2] = new Employee("Tony Tester", 38000);

      Arrays.sort(staff);

      // print out information about all Employee objects
      for (Employee e : staff)
         System.out.println("name=" + e.getName() + ",salary=" + e.getSalary());
   }
}
```

Listing 6.1: interfaces/EmployeeSortTest.java

```
package interfaces;
public class Employee implements Comparable<Employee>
{
   private String name;
   private double salary;
   public Employee(String name, double salary) {
      this.name = name;
      this.salary = salary;
   }
   public String getName() {
      return name;
   }
   public double getSalary() {
      return salary;
   }
   public void raiseSalary(double byPercent) {
      double raise = salary * byPercent / 100;
      salary += raise;
   }
   public int compareTo(Employee other) {  // Compare employees by salary
      return Double.compare(salary, other.salary);
   }
}
```

Properties of Interfaces

- An interface can have one or more methods(public and abstract by default)
- An interface can have zero or more constant fields(public static final by default);
- Interfaces never have instance fields.
- An interface never have constructors.
- The methods of interface are never implemented in the interface(before java 8)
  - After java 8, interface can have simple *static methods* with implementation code.
  - After java 8, interfaces can have non-static methods with *default implementation*.
- The class that *implements* an interface, but does not implement all of abstract methods of the interface is an abstract class.

Properties of Interfaces

- Since an interface is not a class, we can't instantiate it:

```
Comparable x = new Comparable(. . .);  // Error
```

- You can declare variables of interface type:

```
Comparable x ; // OK
```

- An interface variable refer to an object of a class that implements the interface(*is-relationship*):

```
Comparable x = new Employee(. . .); // OK if Employee implements Comparable
```

- You can use instanceOf to check whether an object implements an interface(*subtyping*):

```
if (anObject instanceOf Comparable) { . . . }
```

Properties of Interfaces

An interface can extend another:

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

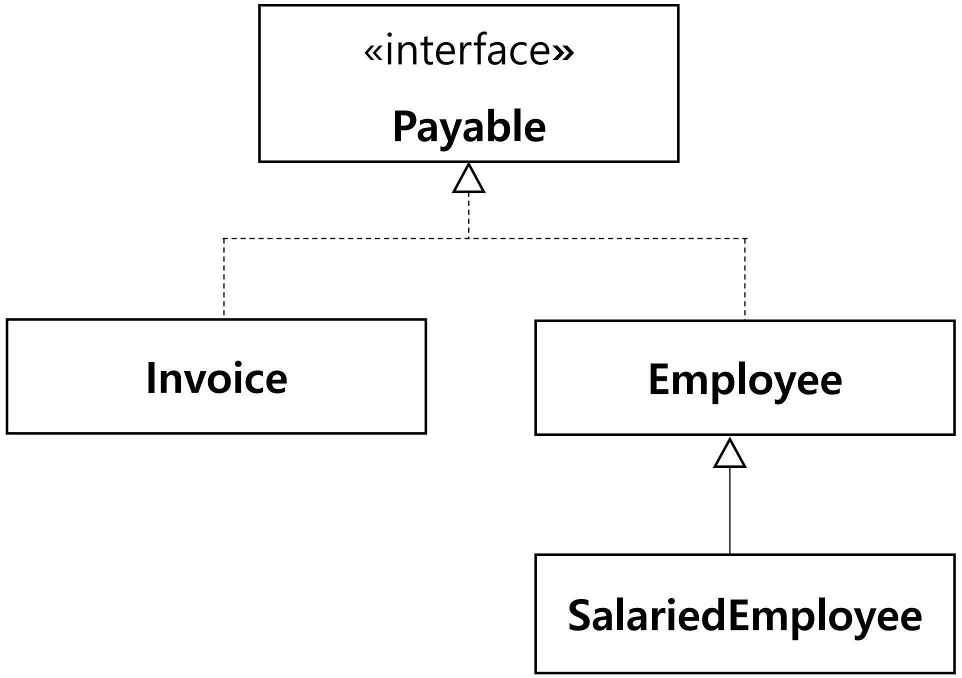
- An interface can have fields:

```
public interface Moveable
{
   . . .
   double SPEED_LIMIT = 95; // automatically public static final
}
```

- A class can implement multiple interfaces:

```
public class Employee implements Comparable<Employee>, Moveable { ... }
```

Example : Payable Hierarchy



```
public interface Payable
{
   double getPaymentAmount();
}
```

Example : Payable Hierarchy

```
public class Invoice implements Payable
{
   private final String partNumber;
   private final String partDescription;
   private int quantity;
   private double pricePerItem;

   public Invoice(String partNumber, String partDescription,
int quantity,
      double pricePerItem)
   {
      this.quantity = quantity;
      this.partNumber = partNumber;
      this.partDescription = partDescription;
      this.pricePerItem = pricePerItem;
   }

   // get/set methods here …

  @Override
   public double getPaymentAmount()
   {
      return getQuantity() * getPricePerItem();
   }

  // toString method here
} // end class Invoice
```

```
public abstract class Employee implements Payable
{
   private final String firstName;
   private final String lastName;
   private final String socialSecurityNumber;

   public Employee(String firstName, String lastName,
      String socialSecurityNumber)
   {
      this.firstName = firstName;
      this.lastName = lastName;
      this.socialSecurityNumber = socialSecurityNumber;
   }

   // get/set methods here
   // getPaymentAmount() is not overridden

   public String toString()
   {
      return String.format("%s %s%nsocial security number: %s",
         getFirstName(), getLastName(),
getSocialSecurityNumber());
   }
}
```

Example : Payable Hierarchy

```
public class SalariedEmployee extends Employee
{
   private double weeklySalary;
   public SalariedEmployee(String firstName, String
lastName,
      String socialSecurityNumber, double weeklySalary)
   {
      super(firstName, lastName, socialSecurityNumber);
      this.weeklySalary = weeklySalary;
   }

   // get/set methods here

  @Override
   public double getPaymentAmount()
   {
      return getWeeklySalary();
   }
  @Override
   public String toString()
   {
      return String.format("salaried employee: %s%n%s:
$%,.2f",
         super.toString(), "weekly salary", getWeeklySalary());
   }
} // end class SalariedEmployee
```

```
public class PayableInterfaceTest
{
   public static void main(String[] args)
   {
      Payable[] payableObjects = new Payable[4];

      payableObjects[0] = new Invoice("01234", "seat", 2, 375.0);
      payableObjects[1] = new Invoice("56789", "tire", 4, 79.95);
      payableObjects[2] =
         new SalariedEmployee("John", "Smith", "111-11-1111", 800.0);
      payableObjects[3] =
         new SalariedEmployee("Lisa", "Barnes", "888-88-8888", 1200.0);

      System.out.println(
         "Invoices and Employees processed polymorphically:");

      // generically process each element in array payableObjects
      for (Payable currentPayable : payableObjects)
      {
         // output currentPayable and its appropriate payment amount
         System.out.printf("%n%s %n%s: $%,.2f%n",
            currentPayable.toString(), // could invoke implicitly
            "payment due", currentPayable.getPaymentAmount());
      }
   } // end main
} // end class PayableInterfaceTest
```

Interfaces and Abstract Classes

- Why not make **Comparable** into an abstract class?

```
abstract class Comparable // why not?
{
   public abstract int compareTo(Object other);
}
```

- Then **Employee** would simply extend it:

```
class Employee extends Comparable // why not?
{
   public int compareTo(Object other) { . . . }
}
```

- Problem: What if **Employee** already has a superclass?

```
class Employee extends Person, Comparable // Error
```

- Java has no “multiple inheritance”.

Interfaces and Abstract Classes

	Abstract Classes	Interfaces
variables	- all kinds of variables	- final static variables (constants)
methods	- abstract or concrete methods - static or non-static	- abstract methods only - default implementation (ver. 8) - static methods (ver. 8)
access modifiers	- all kinds of access modifiers public, protected, private, etc.	- public only
constructor	- can be defined	- cannot be defined
multiple inheritance	- can extends only one class - but, implements multiple interfaces	- extends multiple interfaces - but, cannat extends classes
is-relationship	- can be used as supertype or subtype	- can be used as supertype or subtype
instantiation	- no	- no
when to use	- abstract superclass for some related classes (with shared variables and methods)	- abstract supertype for some possibly unrelated classes (with shared public behavior)