CSCI 2120:
Software Design & Development II

UNIT2: Data management

Collections Framework & Generics

Comparable

Overview

1. Introduction
2. When to use Comparable interface in Java?
3. Comparable interface declaration
4. compareTo(Object obj) Method in Java
5. Comparable Examples

Introduction

Comparable interface in Java defines `compareTo()` method for comparing objects of the same type.

In other words, `Comparable` interface defines a standard way in which two objects of the same class will be compared.

It is mainly used to provide the natural sorting order of objects (elements). Natural ordering means usual ordering.

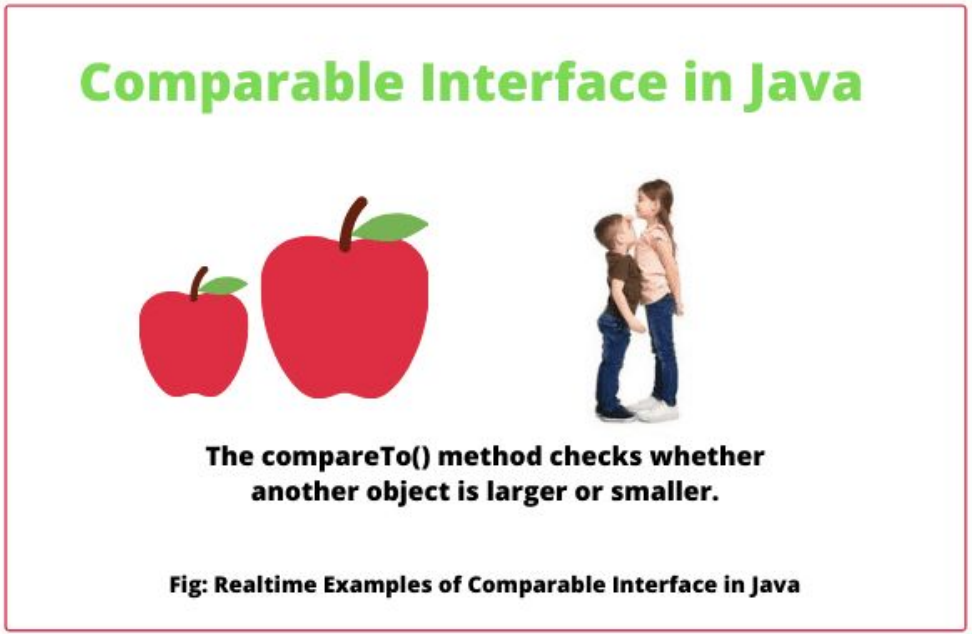
For example, the natural order of strings is alphabetical (i.e. A before B, B before C, and so on). For numeric values, it is numeric order (i.e. 1 before 2, 2 before 3, and so on).

Java `Comparable` interface can be implemented by any custom class or user-defined class if you want to use Arrays or Collections sorting methods.

The sorted collections use the natural sorting order defined by a class to determine the order of objects. So, if you want to store objects in a sorted collection, its class must implement `Comparable` interface.

`Comparable` interface was introduced in Java 1.0 version. It is present in `java.lang.Comparable` package and contains only one method named `compareTo(Object)`.

When to use `Comparable` interface in Java?



Suppose we want to find the larger or smaller of two objects of the same type, such as two students, two employees, two dates, two rectangles, two squares, or two circles.

In order to compare it, the two objects must be comparable. To accomplish this purpose, Java provides `Comparable` interface.

The purpose of using `Comparable` interface and its `compareTo()` method is to compare one object with another object in a way less than (<), equal to (=), or greater than (>).

Comparable interface declaration

Comparable in Java is a generic interface that can be declared like this in the general form:

```
public interface Comparable<T>
```

Here, T represents the type of objects that has to be compared.

Several classes in Java library such as `BigDecimal, BigInteger, Boolean, Byte, ByteBuffer, Calendar, String`, etc implement `Comparable` interface in java to define a natural order for objects.

compareTo(Object obj) Method in Java

The Comparable interface provides only one abstract method that is used to determine the natural ordering of instances of a class. The signature of this method is as follows:

```
public int compareTo(Object obj)

// This method is defined in Comparable interface for comparing objects
// (defined in java.lang package) like this:

package java.lang;
public interface Comparable<E> {
   public int compareTo(Object obj);
}
```

compareTo(Object obj) Method in Java

The `compareTo()` method is used to compare the current object with the specified object. It returns an indication that is as follows:

- positive integer value, if the current object is greater than the specified object (this > object).
- negative integer value, if the current object is less than the specified object (this < object).
- zero, if the current object is equal to the specified object (this = object).

compareTo(Object obj) Method in Java

Let's understand it with the following examples.

1. `a.compareTo(b)`
   returns a negative number if `a` < `b`, zero if `a` == `b`, and a positive number if `b` < `a`.

2. `System.out.println(new Integer(3).compareTo(new Integer(5)));`
   returns a negative value because 3 is less than 5.

3. `System.out.println("ABC".compareTo("ABE"));`
   returns negative value because `ABC` is less than `ABE`.

4. ```
   java.util.Date date1 = new java.util.Date(2021, 1, 1);
   java.util.Date date2 = new java.util.Date(2020, 1, 1);
   System.out.println(date1.compareTo(date2));
   ```
   return a positive value because `date1` is greater than `date2`.

Thus, numbers, strings, dates are comparable. We can use the `compareTo()` method to compare two numbers, two strings, and two dates.
   ```

compareTo(Object obj) Method in Java

In general, we can sort elements of String objects, Wrapper class objects, and User-defined class objects.

If the two objects are not compatible with each other, the `compareTo()` method will throw an exception named **`ClassCastException`**.

Comparable Examples

Example 1: Comparable Student

1. Let's take an example program where we will define a user-defined class `Student` that will implement Comparable interface to sort the list of elements on the basis of roll numbers. Look at the program source code to understand better.

Example 1: Comparable Student - Sort in Natural Order

```
class Student implements Comparable<Student> {
   // Declaration of variables.
   String name;
   int rollno;
   int age;

   Student(String name, int rollno,int age){
       this.name = name;
       this.rollno = rollno;
       this.age = age;
   }
   // Compare two students based on their roll numbers.

   @Override// Implementing the compareTo method defined in Comparable interface.
   public int compareTo(Student st) {
   // Logic for sorting elements in ascending order.
       if(rollno == st.rollno)
           return 0;
       else if(rollno > st.rollno)
           return 1;
       else
           return -1;
   }
}
```

Example 1: Comparable Student - Sort in Natural Order

```
import java.util.ArrayList;
import java.util.Collections;
public class ComparableTester1 {
   public static void main(String[] args) {
       // Creates objects of Student class and passes the parameters to their constructors.
       Student st1 = new Student("John", 20, 15);
       Student st2 = new Student("Peter", 15, 16);
       Student st3 = new Student("Deep", 25, 15);

       // Create an object of ArrayList of type Student.
       ArrayList<Student> al = new ArrayList<>();

       // Adds elements (references) to the array list.
       al.add(st1);
       al.add(st2);
       al.add(st3);

       // Display name of students, sorted by rollnos.
       System.out.println("Displaying student's name sorted by rollnos:");
       Collections.sort(al); // This method is used to sort elements of list.
       for(Student st:al){
           System.out.println(st.name+" "+st.rollno+" "+st.age);
       }
   }
}
```

Example 1: Comparable Student - Sort in Natural Order

Output:

```
Displaying student's name sorted by rollnos:
Peter 15 16
John 20 15
Deep 25 15
```

Example 2: Comparable Person - Sort in Reverse Order

2. Let’s take the same example program where we will sort the list of elements on the basis of rollnos in reverse order. We only need to change the particular code in the Student class.

Example 2: Comparable Person - Sort in Reverse Order

```
class Person implements Comparable<Person> {
   // Declaration of variables.
   String name;
   int rollno;
   int age;

   Person(String name, int rollno,int age){
       this.name = name;
       this.rollno = rollno;
       this.age = age;
   }
   // Compare two persons based on their roll numbers.

   @Override// Implementing the compareTo method defined in Comparable interface.
   public int compareTo(Person p) {
   // Logic for sorting elements in ascending order.
       if(rollno == p.rollno)
           return 0;
       else if(rollno < p.rollno)
           return 1;
       else
           return -1;
   }
}
```

Example 2: Comparable Person - Sort in Reverse Order

```
import java.util.ArrayList;
import java.util.Collections;
public class ComparableTester2 {
   public static void main(String[] args) {
       // Creates objects of Person class and passes the parameters to their constructors.
       Person p1 = new Person("John", 20, 15);
       Person p2 = new Person("Peter", 15, 16);
       Person p3 = new Person("Deep", 25, 15);

       // Create an object of ArrayList of type Person.
       ArrayList<Person> al = new ArrayList<>();

       // Adds elements (references) to the array list.
       al.add(p1);
       al.add(p2);
       al.add(p3);

       // Display name of person, sorted by rollnos.
       System.out.println("Displaying person's name sorted by rollnos:");
       Collections.sort(al); // This method is used to sort elements of list.
       for(Person p : al){
           System.out.println(p.name+" "+p.rollno+" "+p.age);
       }
   }
}
```

Example 2: Comparable Person - Sort in Reverse Order

Output:

```
Displaying person's name sorted by rollnos:
Deep 25 15
John 20 15
Peter 15 16
```

Example 3: Comparable Employee - Sort by Salary

3. Let’s take one more example program where we will learn how to sort employee objects in java using comparable interface. In this example program, we will create a class Employee that will implement Comparable interface and will sort employee information based on their hike salary.

Example 3: Comparable Employee - Sort by Salary

```
class Employee implements Comparable<Employee> {
   // Declaration of encapsulated variables.
   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 hikeSalary(double byPercent){
       double hike = salary * byPercent / 100;
       salary += hike;
   }
   @Override
   public int compareTo(Employee emp)
   {
       if (salary < emp.salary) return -1;
       if (salary > emp.salary) return 1;
       return 0;
   }
}
```

Example 3: Comparable Employee - Sort by Salary

```
import java.util.Arrays;
public class ComparableTester3 {
   public static void main(String[] args)
   {
       // Create an array.
       Employee[] staff = new Employee[3];
       staff[0] = new Employee("Harry", 30000);
       staff[1] = new Employee("Carl", 70000);
       staff[2] = new Employee("Tony", 39000);

       Arrays.sort(staff); //  This method is used to sort list of elements in Arrays.

       // Display information about all Employees, sorted by their salaries.
       for (Employee e : staff)
           System.out.println("name: " + e.getName() +", "+"salary = " + e.getSalary());
   }
}
```

Example 3: Comparable Employee - Sort by Salary

Output:

```
name: Harry, salary = 30000.0
name: Tony, salary = 39000.0
name: Carl, salary = 70000.0
```

END