Интерфейсы Comparable и Comparator.

Проблема

Все мы знаем, как сравнивать числа между собой. Мы знаем, что пять - это больше трех, но меньше шести. Мы знаем, как расставить набор чисел в *натуральном порядке:*

-4 -2 0 1 8 10 .....

Но, допустим, нам надо расставить по порядку объекты класса Dog, или класса Auto, или класса Person …. Как сравнить две машины: по году выпуска, по скорости, по цене? Прежде всего, *нам надо*

определить, как мы сравниваем объекты между собой.

Интерфейс Comparable

Как следует из названия, цель реализации интерфейса **java.lang.Comparable** - определить стратегию сравнения объектов класса между собой, иными словами определить *естественный порядок сортировки* (natural sort order) объектов.

Допустим у нас есть класс Student и мы хотим сравнивать одного студента с другим. Например, хотим отсортировать массив студентов по возрасту. Для этого необходимо:

1. Делаем класс Student “сравниваемым”, т.е. имплементирующим Comparable<Student>

```
public class Student implements Comparable<Student>
```

2. Для реализовать Comparable необходимо реализовать единственный метод:

```
public int compareTo(Student o){ }
```

3. Реализовав интерфейс Comparable, мы можем отсортировать массив Student с помощью стандартного метода Arrays.sort(). При этом, метод sort() “знает” как сортировать массив, т.к. может вызвать метод compareTo и сравнить элементы массива между собой.

```
Arrays.sort(students);
```

Метод `compareTo()` реализуемый при имплементации интерфейса Comparable определяет правила сравнения объекта this (т.е объект, от которого вызван метод) с объектом, переданным в параметрах. При реализации метода `compareTo` необходимо соблюдать следующее соглашение, относительно результата:

- `если объект this больше объекта o переданного в параметрах, метод должен вернуть любое число больше нуля.`
- `если объект this меньше объекта o переданного в параметрах, метод должен вернуть любое число меньше нуля.`
- `если объект this равен объекту o переданного в параметрах, метод должен вернуть ноль.`

Пример кода

(хотим сравнивать по возрасту)

```
public class Student implements Comparable<Student>{
   private String name;
   private int age;

   public Student(String name, int age) {
       this.name = name;
       this.age = age;
   }

   @Override
   public int compareTo(Student o) {
       if (this.age>o.age) return 100;
       if (this.age<o.age) return -100;
       return 0;
   }
}
```

Т.к. age это целочисленное значение, метод CompareTo можно реализовать проще:

```
public int compareTo(Student o) {
   return this.age-o.age;
}
```

Такая реализация тоже выполняет “контракт”: если this.age больше - результат положительный. Если o.age больше - результат отрицательный. Если возрасты равны - 0.

Недостатки Comparable

1. Мы можем реализовать compareTo() единственным способом. Т.е. нет возможности где то использовать один способ сравнения а где то другой.
2. Если класс мы не имеем возможности редактировать класс (например стандартные классы JDK: String, Integer ….) у нас нет возможности определить порядок сравнения.

Интерфейс Comparator

Интерфейс `java.util.Comparator` предлагает альтернативный подход к сравнению объектов.

Допустим у нас есть класс Student и мы хотим сравнивать одного студента с другим. Например, хотим отсортировать массив студентов по имени.

1. Необходимо создать класс-comparator студентов, т.е. класс, который реализует интерфейс `Comparator<Student>`. Это может быть любой класс, единственное условие, он должен быть Comparator<Student>.
2. Для реализации интерфейса Comparator достаточно реализовать единственный метод `int compare(Student student1, Student student2)`.
3. Чтобы выполнить сортировку массива студентов вызываем метод Arrays.sort() передавая в качестве параметра объект-comparator:
   `Arrays.sort(students, new ComparatorByName())`

Метод `compare()` реализуемый при имплементации интерфейса Comparator очень похож на метод `compareTo()` интерфейса Comparable, но в отличие от последнего, метод compare() оба сравниваемых объекта получает в парраметрах. Логика метода реализовать стратегию сравнения student1 и student2. Соглашения относительно интерпретации результата, аналогичны:

- `Если student1 > student2 метод должен вернуть любое положительное число`
- `если student1 < student2 результат должен быть отрицательным.`
- `В случае равенства студентов необходимо вернуть 0.`

Comparator VS Comparable

1. Comparable позволяет задать единственный вариант сравнения. Comparator'ов может быть несколько, каждый реализует свой вариант сравнения.
2. Comparable реализуется в самом классе, объекты которого сравниваем. Соответственно, если изменение класса не доступно, мы не можем реализовать Comparable Comparator - отдельный, независимый класс, реализуемый по необходимости.
3. Метод compareTo реализуемый Comparable находится непосредственно в классе и имеет доступ к полям "сравниваемого" класса, метод compare интерфейса Comparator находится в стороннем объекте и должен обращаться к полям сравниваемых объектов через методы-геттеры.
4. Идея Comparable - определить порядок объектов по умолчанию, идея Comparator - реализовать необходимы здесь и сейчас вариант сравнения объектов

Пример кода

```
public class Student{
   private String name;
   private int age;

   public Student(String name, int age) {
       this.name = name;
       this.age = age;

   }

   public int getAge() {
      return age;
   }

   public String getName() {
       return name;
   }
}
```

```
import java.util.Comparator;
public class StudentComparatorByName implements Comparator<Student> {

   @Override
   public int compare(Student student1, Student student2) {
       return student1.getName().compareTo(student2.getName());
   }
}
```

```
import java.util.Comparator;
public class StudentComparatorByAge implements Comparator<Student> {

   @Override
   public int compare(Student student1, Student student2) {
       return student1.getAge()-student2.getAge();
   }
}
```

```
public class Main {
        public static void main(String[] args) {
           Student[] students = {
                   new Student("Jack",20),
                   new Student("Ann",23),
                   new Student("Nick",22),
           };
           Arrays.sort(students,new StudentComparatorByName()); // сортируем по имени
           Arrays.sort(students,new StudentComparatorByAge()); // сортируем по возрасту
        }
}
```

—----------

Дополнительно:

- https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Comparable.html
- https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Comparator.html
- https://www.baeldung.com/java-comparator-comparable