Comparable和Comparator区别

1、Comparable

> Comparable是**内部比较器**（即用于实现了Comparable接口的类与自身进行比较），称自然排序；
>
> 至于一个类与实现了Comparable接口的另一个类的比较则依赖于CompareTo方法的实现。
>
> 返回值有3个

```
/**
 * @Author AlexanderBai
 * @data 2019/3/14 15:20
 */
public class Domain implements Comparable<Domain> {
    private String str;

    public Domain(String str) {
        this.str = str;
    }

    public String getStr() {
        return str;
    }
    public void setStr(String str) {
        this.str = str;
    }

    @Override
    public int compareTo(Domain domain) {
        if (this.str.compareTo(domain.str)>0) {
            return 1;
        }
        else if (this.str.compareTo(domain.str) == 0) {
            return 0;
        }
        else
            return -1;
    }

    public static void main(String[] args) {
        Domain domain = new Domain("a");
        Domain domain1 = new Domain("a");
        Domain domain2 = new Domain("B");
        Domain domain3 = new Domain("c");
        System.out.println("domain.compareTo(domain1) = " + domain.compareTo(domain1));
        System.out.println("domain.compareTo(domain2) = " + domain.compareTo(domain2));
        System.out.println("domain2.compareTo(domain3) = " + domain2.compareTo(domain3));
    }
}
```

> 运行结果：
>
> domain.compareTo(domain1) = 0 domain.compareTo(domain2) = 1 domain2.compareTo(domain3) = -1

1.1、基于Comparable实现的二叉树操作

Comparable比较器的排序过程是**二叉树**的排序方法，再利用中序遍历依次读取

```
package testfile;

/**
 * @Author AlexanderBai
 * @data 2019/3/14 15:52
 */
class BinaryTree{
    class Node{
        private Comparable data;
        private Node left;
        private Node right;

        /**
         * 添加子节点
         * @param newNode
         */
        public void addNode(Node newNode) {
            if (newNode.data.compareTo(this.data)<0) {
                if (this.left == null) {
                    this.left=newNode;
                }else {
                    this.left.addNode(newNode);
                }
            }
            if (newNode.data.compareTo(this.data)>=0){
                if (this.right == null) {
                    this.right = newNode;
                }else {
                    this.right.addNode(newNode);
                }
            }
        }

        //中序遍历输出
        public void printNode() {
            if (this.left != null) {
                this.left.printNode();
            }
            System.out.println(this.data+"\t");
            if (this.right != null) {
                this.right.printNode();
            }
        }
    }
    private Node root;
```

```
    /**
     * 添加根节点
     * @param data
     */
    public void add(Comparable data) {
        Node newNode=new Node();
        newNode.data=data;
        if (root == null) {
            root=newNode;
        }else {
            root.addNode(newNode);
        }
    }

    public void print() {
        this.root.printNode();
    }
}

public class Test {
    public static void main(String[] args) {
        BinaryTree binaryTree=new BinaryTree();
        binaryTree.add(8);
        binaryTree.add(87);
        binaryTree.add(2456);
        binaryTree.add(76);
        binaryTree.add(1);
        binaryTree.add(44);
        System.out.println("排序之后的结果是：");
        binaryTree.print();
    }
}
```

运行结果：

```
"C:\Program Files\Java\jdk1.8.0_92\bin\java.exe" ...
排序之后的结果是：
1
8
44
76
87
2456

Process finished with exit code 0
```

2、Comparator

一般使用Comparator有两种方式

①、不支持自己和自己比较

②、对自然排序方法ComparaTo不满意，自己重写

假设对以上的ComparaTo方法不满意,DoMian类不变

```
import java.util.Comparator;

/**
 * @Author AlexanderBai
 * @data 2019/3/14 17:30
 */
public class TestComparator implements Comparator<Domain> {

    @Override
    public int compare(Domain domain1, Domain domain2) {
        if (domain1.getStr().compareTo(domain2.getStr()) > 0) {
            return 1;
        }
        else if (domain1.getStr().compareTo(domain2.getStr()) == 0){
            return 0;
        }
        else
            return -1;
    }

    public static void main(String[] args) {
        Domain domain = new Domain("a");
        Domain domain1 = new Domain("a");
        Domain domain2 = new Domain("B");
        Domain domain3 = new Domain("c");
        TestComparator testComparator=new TestComparator();
        System.out.println("testComparator.compare(domain,domain1) = " +
testComparator.compare(domain, domain1));
        System.out.println("testComparator.compare(domain,domain2) = " +
testComparator.compare(domain1, domain2));
        System.out.println("testComparator.compare(domain2,domain3) = " +
testComparator.compare(domain2, domain3));
    }
}
```

> 运行结果：
>
> testComparator.compare(domain,domain1) = 0 testComparator.compare(domain,domain2) = 1
> testComparator.compare(domain2,domain3) = -1