1. What is the compareTo() method in Java

`compareTo()` method is defined in interface **java.lang.Comparable** and it is used to implement natural sorting on java classes. natural sorting means the sort order which naturally applies on object e.g. lexical order for String, numeric order for Integer or Sorting employee by there ID, etc. most of the java core classes including **String and Integer implements CompareTo()** method and provide natural sorting.

2. Why do you need CompareTo()

Sorting is an essential part of application development, which you often required to implement in your system. in Java sorting is implemented using Comparator and Comparable in Java.

Since we store java objects in Collection there are also certain `Set` and `Map` which provides automating sorting when you insert element on that e.g. TreeSet and TreeMap. to implement sorting you need to override either `compareTo(Object o)` method or Comparable class or `compare(Object o1, Object o2)` method of Comparator class.

Most of the classes implement Comparable to implement natural order. for example if you are writing Employee object you probably want to implement Comparable interface and override `compareTo`() method to compare current employee with other employees based on ID.

So essentially you need to override `compareTo()` because you need to `sort elements in ArrayList` or any other `Collection`.

How to implement compareTo in Java

There are certain rules and important points to remember while overriding `compareTo` method:

1) `CompareTo` method must return negative number if current object is less than other object, positive number if current object is greater than other object and zero if both objects are equal to each other.

2) `CompareTo` must be in consistent with equals method e.g. if two objects are equal via `equals()` , there `compareTo()` must return `zero` otherwise if those objects are stored

in `SortedSet` or `SortedMap` they will not behave properly.
Since `SortedSet` or `SortedMap` use `compareTo()` to check the object if two unequal object are returned equal by `compareTo` those will not be added into Set or Map if they are not using external Comparator.

One example where `compareTo` is not consistent with equals in JDK is `BigDecimal` class. two `BigDecimal` number for which compareTo returns zero, equals returns false as clear from following `BigDecimal` comparison example:

```
BigDecimal bd1 = new BigDecimal("2.0");
BigDecimal bd2 = new BigDecimal("2.00");

System.out.println("comparing BigDecimal using equals: " +
bd1.equals(bd2));
System.out.println("comparing BigDecimal using compareTo: " +
bd1.compareTo(bd2));

Output:
comparing BigDecimal using equals: false
comparing BigDecimal using compareTo: 0
```

How does it affect `BigDecimal` ? well if you store these two `BigDecimal` in `HashSet` you will end up with duplicates (violation of Set Contract) i.e. two elements while if you store them in `TreeSet` you will end up with just 1 element because HashSet uses equals to check duplicates while `TreeSet` uses compareTo to check duplicates. That's why its suggested to keep **`compareTo` consistent with equals method in java**.

3) `CompareTo()` must throw `NullPointerException` if current object get compared to `null` object as opposed to `equals()` which return false on such scenario.

4) Another important point to note is **don't use subtraction for comparing integral values** because result of subtraction can overflow as every int operation in Java is modulo 2^32. use either `Integer.compareTo()` or `logical operators` for comparison.

There is one scenario where you can use subtraction to reduce clutter and improve performance. As we know `compareTo` doesn't care magnitude, it just care whether result is positive or negative. While comparing two integral fields you can use subtraction if you are absolutely sure that both operands are positive integer or more precisely there different must be less than `Integer.MAX_VALUE`. In this case there will be no overflow and your compareTo will be concise and faster.

5. Use relational operator to compare integral numeric value i.e. `<` or `>` but use `Float.compareTo()` or `Double.compareTo()` to compare floating point number as relational operator doesn't obey contract of `compareTo` for floating point numbers.

6. `CompareTo()` method is for comparison so **order in which you compare two object matters**. If you have more than one significant field to compare than always *start comparing from most significant field* to least significant field.

here **`compareTo` is different with `equals`** because in case of equality check order doesn't matter. like in above *example of compareTo* if we don't consider Id and compare two student by its name and age than name should be first compare and than age, so if two student have same name one that has higher age should result in greater.

```
Student john12 = new Student(1001, "John", 12);
Student john13 = new Student(1002, "John", 13);

//compareTo will return -1 as age of john12 is less than john13
System.out.println("comparing John, 12 and John, 13 with compareTo :" +
john12.compareTo(john13));

Output:
comparing John, 12 and John, 13 with compareTo :-1
```

7. Another important point while comparing `String` using `compareTo` is to consider case. just like `equals()` doesn't consider case, `compareTo` also do not consider case, if you want to compare regardless of case than use `String.compareToIgnoreCase()` as we have used in above example.