# Comparable introduction

In the previous lesson, we saw that the `Collections.sort()` method sorts the given List in ascending order. But the question is, how does the `sort()` method decide which element is smaller and which one is larger?

Each wrapper class(Integer, Double, or Long), String class, and Date class implements an interface called **Comparable**. This interface contains a `compareTo(T o)` method which is used by sorting methods to sort the Collection. This method returns a negative integer, zero, or a positive integer if **this** object is less than, equal to, or greater than the object passed as an argument.

> If we use the `Collections.sort(List<T> list)` method to sort an **ArrayList**, then the class whose objects are stored in the **ArrayList** must implement the Comparable interface. If the **ArrayList** stores an Integer, a Long, or a String, then we don't need to worry as these classes already implement the Comparable interface. But if the **ArrayList** stores a custom class object, then that class must implement the Comparable interface.

In the below example, we have a custom class called `Employee`. We have stored some `Employee` objects in an **ArrayList**, and we need to sort it. The below example will not compile as the `Employee` class does not implement the Comparable interface.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

class Employee {

    String name;
    int age;

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

}

public class ArrayListDemo {

    public static void main(String args[]) {
        List<Employee> list = new ArrayList<>();
        list.add(new Employee("Jane", 29));
        list.add(new Employee("Alex", 54));

        Collections.sort(list);
        System.out.println("ArrayList in asc order: " + list);
    }
}
```

Run Save Reset

In the below example, the `Employee` class implements the `Comparable` interface. The code will run successfully and will sort the `Employee` objects in ascending order of their age.

ArrayListDemo.java
Employee.java

```
class Employee implements Comparable<Employee> {

    String name;
    int age;

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

    @Override
    public int compareTo(Employee emp) {
        //We will sort the employee based on age in ascending order
        //returns a negative integer, zero, or a positive integer as this emp
        //is less than, equal to, or greater than the specified object.
        return (this.age - emp.age);
    }
}
```

Run Save Reset

# How to write implementation of the `compareTo()` method

Let's say you have a custom class, and you need to write the implementation of the `compareTo()` method.

The first step will be to select the fields within that class where you need to sort the objects. For example, if you have a `Vehicle` class then you would like to sort it on the basis of the year it was sold.

Once you have decided the field where the sorting will be done, then the second step will be to write the implementation of the `compareTo(T o)` method. The `compareTo(T o)` method takes only one object as an input. The comparison is made with the calling object. Let's say we have two `Vehicle` class objects:

```
Vehicle v1 = new Vehicle();
Vehicle v2 = new Vehicle();
```

Then `v1.compareTo(v2)` should return:

1. **-1** if the production year of `v1` is less than the production year of `v2`
2. **1** if the production year of `v1` is greater than the production year of `v2`
3. **0** if the production year of `v1` is equal to the production year of `v2`

If we need to sort the `Vehicle` class on the basis of the year it was made, the logic will look as below:

Vehicle.java
ArrayListDemo.java

```
public class Vehicle implements Comparable<Vehicle> {

    String brand;
    Integer makeYear;

    public Vehicle(String brand, Integer makeYear) {
        super();
        this.brand = brand;
        this.makeYear = makeYear;
    }

    @Override
    public int compareTo(Vehicle o) {
        return this.makeYear - o.makeYear;
        // We can also use the compareTo() method of the Integer class.
        //return this.makeYear.compareTo(o.makeYear);
    }
}
```

Run Save Reset

If we need to sort the `Vehicle` class on the basis of the brand name, the logic will be as below:

Vehicle.java
ArrayListDemo.java

```
public class Vehicle implements Comparable<Vehicle> {

    String brand;
    Integer makeYear;

    public Vehicle(String brand, Integer makeYear) {
        super();
        this.brand = brand;
        this.makeYear = makeYear;
    }

    @Override
    public int compareTo(Vehicle o) {
        //Using the compareTo() method of String class.
        return this.brand.compareTo(o.brand);
    }
}
```

Run Save Reset