One of the major drawbacks of using a Comparable interface is that the comparing logic gets fixed. If we take the example of the `Vehicle` class that we looked at in the previous lesson, then it can be sorted either on the basis of the brand or the production year depending on the implementation of the `compareTo()` method.

If we need some flexibility in sorting, we should use the Comparator interface instead of the Comparable interface. The Comparator interface has a method, `compare(T o1, T o2)`, which takes two objects, o1 and o2 as parameters. It returns -1 if $o1 < o2$, 1 if $o1 > o2$ and 0 if o1 is equal to o2.

If we need to use the Comparator interface, then we can't use the `Collections.sort(List<T> t)` method as `T` should implement the Comparable interface. There is another overloaded method, `sort(List<T> list, Comparator<? super T> c)`, that takes the list as well as a Comparator object as input. It then sorts the List on the basis of logic, which is provided in the Comparator implementation.

The below code shows how to create a custom Comparator. We will create two custom comparators: one for sorting by brand and one for sorting by year.

```
import java.util.Comparator;

public class BrandComparator implements Comparator<Vehicle> {

    @Override
    public int compare(Vehicle o1, Vehicle o2) {
        return o1.brand.compareTo(o2.brand);
    }
}

```

```
import java.util.Comparator;

public class MakeYearComparator implements Comparator<Vehicle>{

    @Override
    public int compare(Vehicle o1, Vehicle o2) {
        return o1.makeYear.compareTo(o2.makeYear);
    }
}

```

In the below example, we have used both the Comparators to sort on the basis of brand and production year.

BrandComparator.java

Vehicle.java

ArrayListDemo.java

MakeYearComparator.java

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

public class ArrayListDemo {

    public static void main(String args[]) {
        List<Vehicle> list = new ArrayList<>();
        list.add(new Vehicle("Volkswagen", 2010));
        list.add(new Vehicle("Audi", 2009));
        list.add(new Vehicle("Ford", 2001));
        list.add(new Vehicle("BMW", 2015));

        System.out.println("Sorting by brand name.");
        Collections.sort(list, new BrandComparator());
        for (Vehicle vehicle : list) {
            System.out.println("Vehicle Brand: " + vehicle.brand + ", Vehicle
        }

        System.out.println("Sorting by make year.");
        Collections.sort(list, new MakeYearComparator());
        for (Vehicle vehicle : list) {
            System.out.println("Vehicle Brand: " + vehicle.brand + ", Vehicle
        }
    }
}

```

Run Save Reset

We can also use an anonymous class in the sort method instead of creating a separate class that implements Comparator. This is shown in the below example.

ArrayListDemo.java

Vehicle.java

```
public class Vehicle {

    String brand;
    Integer makeYear;

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

```

Run Save Reset

The above code can be further simplified if we use lambda expressions instead of anonymous classes. Lambda expressions were introduced in Java 8.

ArrayListDemo.java

Vehicle.java

```
public class Vehicle {

    String brand;
    Integer makeYear;

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

```

Run Save Reset