Comparable & Comparator

These are interfaces which are used for sorting objects.

- Comparable is used to define natural ordering of objects.
  Natural Ordering:
  - Strings -> alphabetical order(A-Z, a-z)
  - Numbers-> ascending order(1,2,3,……)
- Comparator is used to define the custom ordering.

1. Comparable:
   It is an interface which is a part of java.lang package.
   It has a single method called compareTo, which is used to define the natural ordering of objects.

   SYNTAX:
   public class Employee implements Comparable<Employee> {

   ```
       @Override
       public int compareTo(Employee obj){
           //We can define comparison logic here.
           }
   }
   ```

   Example:

```
package exampleC;

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

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

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

```
    public int getAge() {
        return age;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }

    @Override
    public int compareTo(Employee emp) {
        return this.age - emp.age;  // Natural sorting with
age
    }
}
```

```
package exampleC;

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

public class Main {
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();
        employees.add(new Employee("Krishna",30));
        employees.add(new Employee("Gopal",20));
        employees.add(new Employee("Govinda",50));

        Collections.sort(employees);

        for (Employee emp: employees){
            System.out.println(emp);
        }

    }
}
```

[Comparable (Java SE 21 & JDK 21)]

2. Comparator Interface
    - It is a part of java.util package.
    - It provides compare method
    - It allows custom ordering for objects.

    SYNTAX:

```
public class MyComparator implements
Comparator<MyClass > {
        @override
        Public int compare(MyClass obj1, MyClass obj2){
        // will define comparison logic here
  }
}
```

```
package exampleC;

import java.util.Comparator;

public class NameComparator implements Comparator<Employee> {
    @Override
    public int compare(Employee emp1, Employee emp2){
        return emp1.getName().compareTo(emp2.getName());   //
sort by name
    }
}
```

```
package exampleC;

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

public class Main {
    public static void main(String[] args)  {
        List<Employee> employees = new ArrayList<>();
        employees.add(new Employee("Krishna",30));
        employees.add(new Employee("Gopal",20));
        employees.add(new Employee("Govinda",50));

        Collections.sort(employees, new NameComparator()); //
Uses Comparator

        for (Employee emp: employees){
            System.out.println(emp);
        }
    }
}
```

Comparator (Java SE 21 & JDK 21)

	Comparable	Comparator
Package	java.lang	Java.util
Method	compareTo(T obj)	Compare(T obj1, T obj2)
Sorting logic	Natural ordering	Custom ordering
Class modification	Modifies the original class	Does not modify the original class

- Use Comparable when we want a default natural ordering.
- Use Comparator when we need multiple ways to sort objects.

TASK:

Student Class

Attributes: name, rollNo, marks.

Implement comparable to sort students by rollNo and use comparator to sort students by marks.(sort marks by descending order).

```
package combineEx;

public class Student implements Comparable<Student>{

    private String name;
    private int rollNo;
    private double marks;

    public Student(String name, int rollNo, double marks) {
        this.name = name;
        this.rollNo = rollNo;
        this.marks = marks;
    }

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

```java
    public int getRollNo() {
        return rollNo;
    }

    public double getMarks() {
        return marks;
    }

    @Override
    public String toString() {
        return "Student{" +
                "name='" + name + '\'' +
                ", rollNo=" + rollNo +
                ", marks=" + marks +
                '}';
    }

    @Override
    public int compareTo(Student student){
        return this.rollNo - student.rollNo; // Natural
Sorting with rollNo
    }

}
```

```java
package combineEx;

import java.util.Comparator;

public class MarksComparator implements Comparator<Student> {
    @Override
    public int compare(Student s1, Student s2){
        return Double.compare(s2.getMarks(),s1.getMarks()); //
Sort by marks in descending order.
    }
}
```

```java
package combineEx;

import java.sql.SQLOutput;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<Student> students =  new ArrayList<>();
        students.add(new Student("Krishna", 100, 95.5));
        students.add(new Student("Govind", 101, 85.0));
        students.add(new Student("Gopal", 102, 90.0));

        System.out.println("List of student sorted by roll
No(Natural Order):");
        Collections.sort(students);
        for (Student student: students){
            System.out.println(student);
        }


System.out.println("++++++++++++++++++++++++++++++++++++++++++
++++++++");
        System.out.println("List of students sorted by
marks(Descending order): ");
        Collections.sort(students, new MarksComparator());
        for (Student student: students){
            System.out.println(student);
        }

    }
}
```

Task:

Book Class : title, author, price

Implement comparable to sort book by title

Create comparator to sort books by price.

Steps:

1. Create a book class -> attributes-> constructors->getters -> toString
2. Implement(in book class) Comparable to sort books by title(override the compareTo)
3. Test it in Main Class

Threads:

A thread is a lightweight sub-process which runs independently within a program.

Threads are used for multitasking and improve the performance.

2 ways to create threads

- By extending the thread class
- By implementing the runnable interface.

………