Comparator

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

SYNTAX:

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

```
public class 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 void setName(String name) {
        this.name = name;
    }

    public int getAge() {
```

```
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

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

```
import java.util.Comparator;

public class NameComparator implements Comparator<Employee> {

    @Override
    public int compare(Employee emp1, Employee emp2) {
        return emp1.getName().compareTo(emp2.getName()); //
sorting by name
    }
}
```

```
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("Adarsh",20));
        employees.add(new Employee("Bishal", 40));

        Collections.sort(employees,new NameComparator());  //
using comparator

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

```
        }
    }
}
```

O/P:

Employee{name='Adarsh', age=20}

Employee{name='Bishal', age=40}

Employee{name='Krishna', age=30}

TASK:

Student Class

Attributes: name, rollNo, marks

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

```
package studentExample;

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;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getRollNo() {
        return rollNo;
    }

    public void setRollNo(int rollNo) {
        this.rollNo = rollNo;
    }

    public double getMarks() {
        return marks;
    }

    public void setMarks(double marks) {
        this.marks = 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 roll No
    }
}
```

```java
package studentExample;

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 the descending order
    }
}
```

```java
package studentExample;

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", 101, 95.5));
        students.add(new Student("Govind", 100, 85.5));
        students.add(new Student("Gopal", 102, 90.5));

        System.out.println("Sorting by Roll No.: ");
        Collections.sort(students);
        for (Student std : students){
            System.out.println(std);
        }



System.out.println("++++++++++++++++++++++++++++++++++++++++++++++
++++++++++");

        System.out.println("Sorting by marks in descending
order: ");
        Collections.sort(students,new MarksComparator());
        for (Student std : students){
            System.out.println(std);
        }

    }
}
```

o/p:

Sorting by Roll No.:

Student{name='Govind', rollNo=100, marks=85.5}

Student{name='Krishna', rollNo=101, marks=95.5}

Student{name='Gopal', rollNo=102, marks=90.5}

++

Sorting by marks in descending order:

Student{name='Krishna', rollNo=101, marks=95.5}

Student{name='Gopal', rollNo=102, marks=90.5}

Student{name='Govind', rollNo=100, marks=85.5}

BOOK TASK:

Book Class: title, author, price

Implement comparable to sort book by title

Create comparator to sort by book price.

Thread

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

Threads are used for multitasking and improve the performance.

How to create threads:

There are 2 ways to create threads:

1. By extending the thread class Thread (Java SE 21 & JDK 21)
2. By implementing the runnable interface. Runnable (Java SE 21 & JDK 21)

What is thread:

- A thread is a smallest unit of a process
- It can be executed independently
- It is a lightweight sub process
- Each thread has its own path of execution
- It means it can perform tasks concurrently with other threads

Lifecycle of a thread:

New: the thread is created but not yet started

Runnable: The thread is ready to run, waiting for CPU to execute it.

Blocked/Waiting: The thread is temporarily inactive, waiting for a resource or another thread to complete.

Timed Waiting: The thread is waiting for a specific amount of time.(e.g. using sleep())

Terminated: The thread has completed its execution and is no longer running.