Lecture 14: Lambda Expression with Collection:

Comparator is an interface is a SAM and has compare method. In compare method we can implement customized sorting.

Method:

@Override

**public** **int** compare(Object obj1, Object obj22) {

}

* It returns +value if Obj1 has to come after Obj2;
* It returns – value if Obj1 has to come before Obj2;
* It return 0 if both Obj1 and Obj2 are equal

Lecture 15: Sorting elements of List without Lambda Expressions:

**public** **class** Example1 {

**public** **static** **void** main(String[] args) {

ArrayList<Integer> numbers = **new** ArrayList<Integer>();

numbers.add(10);

numbers.add(0);

numbers.add(15);

numbers.add(5);

numbers.add(20);

System.***out***.println("Before sorting "+numbers);

MyComparator myComparator = **new** MyComparator();

numbers.sort(myComparator);

System.***out***.println("After sorting "+numbers);

}

}

**class** MyComparator **implements** Comparator<Integer>{

@Override

**public** **int** compare(Integer o1, Integer o2) {

// **TODO** Auto-generated method stub

**return** -(o1-o2);

}

}

Lecture 16: Sorting elements of List with Lambda Expressions:

**public** **class** Example1 {

**public** **static** **void** main(String[] args) {

ArrayList<Integer> numbers = **new** ArrayList<Integer>();

numbers.add(10);

numbers.add(0);

numbers.add(15);

numbers.add(5);

numbers.add(20);

System.***out***.println("Before sorting "+numbers);

Comparator<Integer> myComparator = (num1, num2) -> -(num1-num2);

numbers.sort(myComparator);

System.***out***.println("After sorting "+numbers);

}

}

Or we need not even create the variable myComparator and it can be further reduced to:

**public** **class** Example1 {

**public** **static** **void** main(String[] args) {

ArrayList<Integer> numbers = **new** ArrayList<Integer>();

numbers.add(10);

numbers.add(0);

numbers.add(15);

numbers.add(5);

numbers.add(20);

System.***out***.println("Before sorting "+numbers);

numbers.sort((num1, num2) -> -(num1-num2));

System.***out***.println("After sorting "+numbers);

}

}

Lecture 17: Sorting Elements of TreeSet with Lambda Expression:

TreeSet: to have elements in TreeSet in natural sorting order then the code is as below.

TreeSet<Integer> treeSet = **new** TreeSet<Integer>();

If we need to have customized sorting then

TreeSet<Integer> treeSet = **new** TreeSet<Integer>(Comparator);

**public** **class** Example1 {

**public** **static** **void** main(String[] args) {

MyComparator myComparator = **new** MyComparator();

TreeSet<Integer> treeSet = **new** TreeSet<Integer>(myComparator);

treeSet.add(10);

treeSet.add(0);

treeSet.add(15);

treeSet.add(25);

treeSet.add(5);

treeSet.add(20);

System.***out***.println(treeSet);

}

}

**class** MyComparator **implements** Comparator<Integer>{

@Override

**public** **int** compare(Integer o1, Integer o2) {

// **TODO** Auto-generated method stub

**return** -(o1-o2);

}

}

With Lambda Expression:

Comparator<Integer> myComparator = (num1, num2 ) -> -(num1-num2);

TreeSet<Integer> treeSet = **new** TreeSet<Integer>(myComparator);

treeSet.add(10);

treeSet.add(0);

treeSet.add(15);

treeSet.add(25);

treeSet.add(5);

treeSet.add(20);

System.***out***.println(treeSet);

Lecture 18: Sorting Elements of TreeMap with Lambda Expression:

With out lambda Expression

**public** **class** Example1 {

**public** **static** **void** main(String[] args) {

MyComparator myComparator = **new** MyComparator();

TreeMap<Integer, String> treeMap = **new** TreeMap<Integer, String>(myComparator);

treeMap.put(10, "Ten");

treeMap.put(0, "Zero");

treeMap.put(15, "Fifteen");

treeMap.put(5, "Five");

treeMap.put(20, "Twenty");

System.***out***.println(treeMap);

}

}

**class** MyComparator **implements** Comparator<Integer>{

@Override

**public** **int** compare(Integer o1, Integer o2) {

// **TODO** Auto-generated method stub

**return** -(o1-o2);

}

}

With Lambda Expression:

Comparator<Integer> myComparator = (num1, num2) -> -(num1 - num2);

TreeMap<Integer, String> treeMap = **new** TreeMap<Integer, String>(myComparator);

treeMap.put(10, "Ten");

treeMap.put(0, "Zero");

treeMap.put(15, "Fifteen");

treeMap.put(5, "Five");

treeMap.put(20, "Twenty");

System.***out***.println(treeMap);

Lecture 19: Sorting of Our own class objects with Lambda Expression:

Without Lambda Expression

**class** Employee{

**int** eNumber;

String eName;

**public** Employee(**int** eNumber, String eName) {

// **TODO** Auto-generated constructor stub

**this**.eNumber=eNumber;

**this**.eName= eName;

}

@Override

**public** String toString() {

// **TODO** Auto-generated method stub

**return** "\nEmployee Number "+**this**.eNumber+" "+"Employee Name "+**this**.eName;

}

}

**class** SortByEmployeeNumber **implements** Comparator<Employee>{

@Override

**public** **int** compare(Employee emp1, Employee emp2) {

// **TODO** Auto-generated method stub

**return** -(emp1.eNumber - emp2.eNumber);

}

}

**public** **class** Example1 {

**public** **static** **void** main(String[] args) {

Employee emp1 = **new** Employee(200,"Emp200");

Employee emp2 = **new** Employee(400,"Emp400");

Employee emp3 = **new** Employee(300,"Emp300");

Employee emp4 = **new** Employee(100,"Emp100");

ArrayList<Employee> employees = **new** ArrayList<Employee>();

employees.add(emp1); employees.add(emp2);

employees.add(emp3); employees.add(emp4);

System.***out***.println("Before Sorting "+employees);

employees.sort(**new** SortByEmployeeNumber());

System.***out***.println("After Sorting "+employees);

}

}

With Lambda Expression:

Comparator<Employee> customComparator =

(employee1, employee2) -> -(employee1.eNumber - employee2.eNumber);

employees.sort(customComparator);