**Lambda Expressions with Collections**

**Collection is nothing but a group of objects represented as a single entity. The important Collection types are:**

**1. List(I)**

**2. Set(I)**

**3. Map(I)**

**1. List(I):**

**If we want to represent a group of objects as a single entity where duplicate objects are allowed and insertion order is preserved then we shoud go for List.**

**1. Insertion order is preserved**

**2. Duplicate objects are allowed**

**The main implementation classes of List interface are:**

**1. ArrayList 2. LinkedList 3. Vector 4. Stack**

**Demo Program to describe List Properties:**

|  |
| --- |
| **1) import java.util.ArrayList;** |
| **2) class Test** |
| **3) {** |
| **4) public static void main(String[] args)** |
| **5) {** |
| **6) ArrayList<String> l = new ArrayList<String>();** |
| **7) l.add("Sunny");** |
| **8) l.add("Bunny");** |
| **9) l.add("Chinny");** |
| **10) l.add("Sunny");** |
| **11) System.out.println(l);** |
| **12) }** |
| **13) }** |

**Output:[Sunny, Bunny, Chinny, Sunny]**

**Note: List(may be ArrayList,LinkedList,Vector or Stack) never talks about sorting order. If we want sorting for the list then we should use Collections class sort() method.**

**Collecttions.sort(list)==>meant for Default Natural Sorting Order**

**Collections.sort(list,Comparator)==>meant for Customized Sorting Order**

**2. Set(I):**

**If we want to represent a group of individual objects as a single entity where duplicate objects are not allowed and insertion order is not preserved then we should go for Set.**

**1. Insertion order is not preserved**

**2. Duplicates are not allowed.If we are trying to add duplicates then we won't get any error, just add() method returns false.**

**The following are important Set implementation classes**

**1. HashSet 2.TreeSet etc**

**Demo Program for Set:**

|  |
| --- |
| **1) import java.util.HashSet;** |
| **2) class Test** |
| **3) {** |
| **4) public static void main(String[] args)** |
| **5) {** |
| **6) HashSet<String> l = new HashSet<String>();** |
| **7) l.add("Sunny");** |
| **8) l.add("Bunny");** |
| **9) l.add("Chinny");** |
| **10) l.add("Sunny");** |
| **11) System.out.println(l);** |
| **12) }** |
| 13) **}** |

**Output: [Chinny, Bunny, Sunny]**

**Note: In the case of Set, if we want sorting order then we should go for: TreeSet**

**3. Map(I):**

**If we want to represent objects as key-value pairs then we should go for**

**Map**

**Eg:**

**Rollno-->Name**

**mobilenumber-->address**

**The important implementation classes of Map are:**

**1. HashMap 2. TreeMap etc**

**Demo Program for Map:**

|  |
| --- |
| **1) import java.util.HashMap;** |
| **2) class Test** |
| **3) {** |
| **4) public static void main(String[] args)** |
| **5) {** |
| **6) HashMap<String,String> m= new HashMap<String,String>();** |
| **7) m.put("A","Apple");** |
| **8) m.put("Z","Zebra");** |
| **9) m.put("Durga","Java");** |
| **10) m.put("B","Boy");** |
| **11) m.put("T","Tiger");** |
| **12) System.out.println(m);** |
| **13) }** |
| 14) **}** |

**Output: {A=Apple, B=Boy, T=Tiger, Z=Zebra, Durga=Java}**

**Sorted Collections:**

**1. Sorted List**

**2. Sorted Set**

**3. Sorted Map**

**1. Sorted List:**

**List(may be ArrayList,LinkedList,Vector or Stack) never talks about sorting order. If we want sorting for the list then we should use Collections class sort() method.**

**Collecttions.sort(list)==>meant for Default Natural Sorting Order**

**For String objects: Alphabetical Order**

**For Numbers : Ascending order**

**Instead of Default natural sorting order if we want customized sorting order then we should go for**

**Comparator interface.**

**Comparator interface contains only one abstract method: compare() Hence it is Functional interface.**

**public int compare(obj1,obj2)**

**returns -ve iff obj1 has to come before obj2 returns +ve iff obj1 has to come after obj2 returns 0 iff obj1 and obj2 are equal**

**Collections.sort(list,Comparator)==>meant for Customized Sorting Order**

**Demo Program to Sort elements of ArrayList according to Defaut Natural Sorting**

**Order(Ascending Order):**

|  |
| --- |
| **1) import java.util.ArrayList;** |
| **2) import java.util.Collections;** |
| **3) class Test** |
| **4) {** |
| **5) public static void main(String[] args)** |
| **6) {** |
| **7) ArrayList<Integer> l = new ArrayList<Integer>();** |
| **8) l.add(10);** |
| **9) l.add(0);** |
| **10) l.add(15);** |
| **11) l.add(5);** |
| **12) l.add(20);** |
| **13) System.out.println("Before Sorting:"+l);** |
| **14) Collections.sort(l);** |
| **15) System.out.println("After Sorting:"+l);** |
| **16) }** |
| **17) }** |

**Output:**

**Before Sorting:[10, 0, 15, 5, 20]**

**After Sorting:[0, 5, 10, 15, 20]**

**Demo Program to Sort elements of ArrayList according to Customized Sorting**

**Order(Descending Order):**

**1) import java.util.TreeSet;**

**2) import java.util.Comparator;**

**3) class MyComparator implements Comparator<Integer>**

**4) {**

**5) public int compare(Integer I1,Integer I2)**

**6) {**

**7) if(I1<I2)**

**8) {**

**9) return +1;**

**10) }**

**11) else if(I1>I2)**

**12) {**

**13) return -1;**

**14) }**

**15) else**

**16) {**

**17) return 0;**

**18) }**

**19) }**

**20) }**

**21) class Test**

|  |
| --- |
| **22) {** |
| **23) public static void main(String[] args)** |
| **24) {** |
| **25) TreeSet<Integer> l = new TreeSet<Integer>(new MyComparator());** |
| **26) l.add(10);** |
| **27) l.add(0);** |
| **28) l.add(15);** |
| **29) l.add(5);** |
| **30) l.add(20);** |
| **31) System.out.println(l);** |
| **32) }** |
| **33) }** |

**//Descending order Comparator**

**Output: [20, 15, 10, 5, 0]**

**Shortcut way:**

**1) import java.util.ArrayList;**

**2) import java.util.Comparator;**

**3) import java.util.Collections;**

**4) class MyComparator implements Comparator<Integer>**

**5) {**

**6) public int compare(Integer I1,Integer I2)**

**7) {**

**8) return (I1>I2)?-1:(I1<I2)?1:0;**

**9) }**

**10) }**

**11) class Test**

**12) {**

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

**14) {**

**15) ArrayList<Integer> l = new ArrayList<Integer>();**

**16) l.add(10);**

**17) l.add(0);**

**18) l.add(15);**

**19) l.add(5);**

**20) l.add(20);**

**21) System.out.println("Before Sorting:"+l);**

**22) Collections.sort(l,new MyComparator());**

**23) System.out.println("After Sorting:"+l);**

**24) }**

**25) }**

**Sorting with Lambda Expressions:**

**As Comparator is Functional interface, we can replace its implementation with Lambda Expression**

**Collections.sort(l,(I1,I2)->(I1<I2)?1:(I1>I2)?-1:0);**

**Demo Program to Sort elements of ArrayList according to Customized Sorting Order By using Lambda Expressions(Descending Order):**

|  |
| --- |
| **1) import java.util.ArrayList;** |
| **2) import java.util.Collections;** |
| **3) class Test** |
| **4) {** |
| **5) public static void main(String[] args)** |
| **6) {** |
| **7) ArrayList<Integer> l= new ArrayList<Integer>();** |
| **8) l.add(10);** |
| **9) l.add(0);** |
| **10) l.add(15);** |
| **11) l.add(5);** |
| **12) l.add(20);** |
| **13) System.out.println("Before Sorting:"+l);** |
| **14) Collections.sort(l,(I1,I2)->(I1<I2)?1:(I1>I2)?-1:0);** |
| **15) System.out.println("After Sorting:"+l);** |
| **16) }** |
| **17) }** |

**Output:**

**Before Sorting:[10, 0, 15, 5, 20]**

**After Sorting:[20, 15, 10, 5, 0]**

**2. Sorted Set**

**In the case of Set, if we want Sorting order then we should go for TreeSet**

**1. TreeSet t = new TreeSet();**

**This TreeSet object meant for default natural sorting order**

**2. TreeSet t = new TreeSet(Comparator c);**

**This TreeSet object meant for Customized Sorting Order**

**Demo Program for Default Natural Sorting Order(Ascending Order):**

|  |
| --- |
| **1) import java.util.TreeSet;** |
| **2) class Test** |
| **3) {** |
| **4) public static void main(String[] args)** |

|  |
| --- |
| **5) {** |
| **6) TreeSet<Integer> t = new TreeSet<Integer>();** |
| **7) t.add(10);** |
| **8) t.add(0);** |
| **9) t.add(15);** |
| **10) t.add(5);** |
| **11) t.add(20);** |
| **12) System.out.println(t);** |
| **13) }** |
| 14) **}** |

**Output: [0, 5, 10, 15, 20]**

**Demo Program for Customized Sorting Order(Descending Order):**

|  |
| --- |
| **1) import java.util.TreeSet;** |
| **2) class Test** |
| **3) {** |
| **4) public static void main(String[] args)** |
| **5) {** |
| **6) TreeSet<Integer> t = new TreeSet<Integer>((I1,I2)->(I1>I2)?-1:(I1<I2)?1:0);** |
| **7) t.add(10);** |
| **8) t.add(0);** |
| **9) t.add(15);** |
| **10) t.add(25);** |
| **11) t.add(5);** |
| **12) t.add(20);** |
| **13) System.out.println(t);** |
| **14) }** |
| **15) }** |

**Output: [25, 20, 15, 10, 5, 0]**

**3. Sorted Map:**

**In the case of Map, if we want default natural sorting order of keys then we should go for**

**TreeMap.**

**1. TreeMap m = new TreeMap();**

**This TreeMap object meant for default natural sorting order of keys**

**2. TreeMap t = new TreeMap(Comparator c);**

**This TreeMap object meant for Customized Sorting Order of keys**

**Demo Program for Default Natural Sorting Order(Ascending Order):**

|  |
| --- |
| **1) import java.util.TreeMap;** |
| **2) class Test** |
| **3) {** |
| **4) public static void main(String[] args)** |
| **5) {** |
| **6) TreeMap<Integer,String> m = new TreeMap<Integer,String>();** |
| **7) m.put(100,"Durga");** |
| **8) m.put(600,"Sunny");** |
| **9) m.put(300,"Bunny");** |
| **10) m.put(200,"Chinny");** |
| **11) m.put(700,"Vinny");** |
| **12) m.put(400,"Pinny");** |
| **13) System.out.println(m);** |
| **14) }** |
| 15) **}** |

**Output: {100=Durga, 200=Chinny, 300=Bunny, 400=Pinny, 600=Sunny, 700=Vinny}**

**Demo Program for Customized Sorting Order(Descending Order):**

|  |
| --- |
| **1) import java.util.TreeMap;** |
| **2) class Test** |
| **3) {** |
| **4) public static void main(String[] args)** |
| **5) {** |
| **6) TreeMap<Integer,String> m = new TreeMap<Integer,String>((I1,I2)->(I1<I2)?1:(I1>I2)?-**  **1:0);** |
| **7) m.put(100,"Durga");** |
| **8) m.put(600,"Sunny");** |
| **9) m.put(300,"Bunny");** |
| **10) m.put(200,"Chinny");** |
| **11) m.put(700,"Vinny");** |
| **12) m.put(400,"Pinny");** |
| **13) System.out.println(m);** |
| **14) }** |
| **15) }** |

**Output: {700=Vinny, 600=Sunny, 400=Pinny, 300=Bunny, 200=Chinny, 100=Durga}**

**Sorting for Customized class objects by using Lambda Expressions:**

|  |
| --- |
| **1) import java.util.ArrayList;** |
| **2) import java.util.Collections;** |
| **3) class Employee** |
| **4) {** |
| **5) int eno;** |
| **6) String ename;** |

**7) Employee(int eno,String ename)**

**8) {**

**9) this.eno=eno;**

**10) this.ename=ename;**

**11) }**

**12) public String toString()**

**13) {**

**14) return eno+":"+ename;**

**15) }**

**16) }**

**17) class Test**

**18) {**

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

**20) {**

**21) ArrayList<Employee> l= new ArrayList<Employee>();**

**22) l.add(new Employee(100,"Katrina"));**

**23) l.add(new Employee(600,"Kareena"));**

**24) l.add(new Employee(200,"Deepika"));**

**25) l.add(new Employee(400,"Sunny"));**

**26) l.add(new Employee(500,"Alia"));**

**27) l.add(new Employee(300,"Mallika"));**

**28) System.out.println("Before Sorting:");**

**29) System.out.println(l);**

**30) Collections.sort(l,(e1,e2)->(e1.eno<e2.eno)?-1:(e1.eno>e2.eno)?1:0);**

**31) System.out.println("After Sorting:");**

**32) System.out.println(l);**

**33) }**

**34) }**

**Output:**

**Before Sorting:**

**[100:Katrina, 600:Kareena, 200:Deepika, 400:Sunny, 500:Alia, 300:Mallika] After Sorting:**

**[100:Katrina, 200:Deepika, 300:Mallika, 400:Sunny, 500:Alia, 600:Kareena]**