Collection API
++++++++++++++

interfaces
++++++++++
1. Collection
2. List

3. Set
   Implementation class :: HashSet,LinkedHashSet
4. SortedSet
5. NavigableSet
   Implementation class :: TreeSet

When to use Set(I) implementation class Objects?
Ans. Duplicates are not allowed
     Insertion Order is not preserved

Note: To iterate any type of Collection object,the Collection Object should be "Iterable".
      For Collection Interface the Parent interface is "Iterable",so Iterator interface can be used for all the Objects.

Note: Map(I) is not a part of Collection.

6. Map(<K,V>)
   Implementation :: HashMap,WeakHashMap,IdentityHashMap
                     Dictionary,Hashtable,Properties
7. SortedMap
8. NavigableMap
   Implementation :: TreeMap

9. Queue
   Implementation class : BlockingQueue,PriorityBlockingQueue,...

When to use Queue(I) implementation class Objects?
Ans. if we want to represent group of individual objects prior to processing then we should go for Queue.

Name the cursors available for iterating the Collection Objects
 a. Enumeration(I) => Legacy Cursor.
 b. Iterator(I)   => Universal Cursor as it can be iterated on every collection Object.
 c. ListIterator(I) => It can be used only on List(I) implementation class Object.

Name the interfaces available for Sorting the Objects
 a. Comparable(I) :: meant for Default Natural Sorting Order.
 b. Comparator(I) :: meant for Customized Sorting Order.

TreeSet
=======
 Underlying Datastructure: BalancedTree
 duplicates : not allowed
 insertion order : not preserved
 heterogenous element: not possible,if we try to do it would result in "ClassCastException".
 inserting null   : NullPointerException.
 Implements Serializable and Cloneable interface,but not RandomAccess.

```
 All Objects will be inserted based on "some sorting order" or "customized sorting
order".

Constructor
============
TreeSet t=new TreeSet();//All objects will be inserted based on some default
natural sorting order.

TreeSet t=new TreeSet(Comparator);//All objects will be inserted based on some
customized sorting order.

TreeSet t=new TreeSet(Collection c);
TreeSet t=new TreeSet(SortedSet);

Note::
Comparable => Default natural sorting order.
Comparator => Customized sorting order.

eg#2.
import java.util.TreeSet;
class TreeSetDemo {
public static void main(String[] args) {
            TreeSet t = new TreeSet();
            t.add(new StringBuffer("A"));
            t.add(new StringBuffer("Z"));
            t.add(new StringBuffer("L"));
            t.add(new StringBuffer("B"));
            System.out.println(t);
      }
}

Output:ClassCastException
Reason:: TreeSet t=new TreeSet()
             a. we inform jvm to use default natural sorting order
                      To sort the elements must be
                           a. Homogenous
                     b. Comparable(class should implement Comparable)
                  otherwise it would result in "ClassCastException".

Note:: Object is said to be Comparable, iff the corresponding class implements
"Comparable".
       All Wrapper class and String class implements Comparable so we can compare
the objects.

Comparable(I)
 => It is a part of java.lang package
 => It contains only one method compareTo.
         public int compareTo(Object o)

     => obj1.compareTo(obj2)
               returns -ve iff obj1 has to come before obj2[B-Negative]
               returns +ve iff obj1 has to come after obj2[A-Postivie]
               returns 0 if both are equal[Both are equal]


eg#1.
System.out.println("A".compareTo("Z"));//A should  come before Z so -ve
System.out.println("Z".compareTo("K"));//Z should  come after  K so +ve
System.out.println("A".compareTo("A"));//Both are equal zero
System.out.println("A".compareTo(null));//NullPointerException

eg#2.
import java.util.TreeSet;
public class TestApp{
      public static void main(String... args){
            TreeSet ts= new TreeSet();

            ts.add("K");
            ts.add("Z");//internally  "Z".compareTo("K") +ve
            ts.add("A");//internally  "A".compareTo("K") -ve
            ts.add("A");//internally  "A".compareTo("K") -ve
                        //internally  "A".compareTo("A")  0

            System.out.println(ts);//[A K Z]
      }
}
  Rule: obj1.compareTo(obj2)
        obj1 => The object which needs to be inserted.
        obj2 => The object which is already inserted.
```

Whenever we are depending on default natural sorting order,if we try to insert the elements
then internally it calls compareTo() to IdentifySorting order.

```
Comparable
   => compareTo()
      It is meant for default natural sorting order.
Comparator
   => compare()
      It is meant for customized sorting order.
```

Write a program to insert integer objects into the TreeSet where sorting order is descending order?

```
import java.util.TreeSet;
import java.util.Comparator;
class MyComparator implements Comparator{
      public int compare(Object obj1,Object obj2){
            Integer i1=(Integer)obj1;
            Integer i2=(Integer)obj2;
            if (i1<i2)
                  return 1;
            else if (i1>i2)
                  return -1;
            else
                  return 0;
      }
}
public class TestApp{
      public static void main(String... args){
            TreeSet ts= new TreeSet(new MyComparator());
            ts.add(10);
            ts.add(0);
            ts.add(15);
            ts.add(5);
            ts.add(20);
            ts.add(20);
            System.out.println(ts);//[0,5,10,15,20]
      }
}

Various Possible combination implementation of compare()
========================================================
class MyComparator implements Comparator{
      public int compare(Object obj1,Object obj2){
            Integer i1=(Integer)obj1;
            Integer i2=(Integer)obj2;
            return i1.compareTo(i2);//ascending order
            return -i1.compareTo(i2);//descending order
            return i2.compareTo(i1);//descending order
                return -i2.compareTo(i2);//ascending order

            return +1;//insertion order is preserved
            return -1;//reverse of insertion order
            return 0;//only first elements is added,remaining all duplicates
      }
}



=> Insert String object into treeset, perform sorting in reverse of Alphabetical
Order.

eg#1.
import java.util.*;

class MyComparator implements Comparator
{
      @Override
      public int compare(Object obj1,Object obj2)
      {
            String s1=(String)obj1 ;
            String s2=obj2.toString();
            return -s1.compareTo(s2);
      }
}

public class Test
{
      public static void main(String[] args)
      {
            //TreeSet[Balanced Tree] -> Comparable :: DNS
            //public int compareTo(Object obj)
            TreeSet ts1 =new TreeSet();
            ts1.add("sachin");
            ts1.add("saurav");
            ts1.add("dhoni");
            ts1.add("kohli");
            ts1.add("yuvi");

            System.out.println(ts1);//[dhoni,kohli,sachin,saurav,yuvi]

            System.out.println();

            //TreeSet[Balanced Tree] -> Comparator :: CSO
            //public abstract int compare(Object obj1,Object obj2);
            //public abstract boolean equals(java.lang.Object);
            TreeSet ts2 =new TreeSet(new MyComparator());
            ts2.add("sachin");
            ts2.add("saurav");
            ts2.add("dhoni");
            ts2.add("kohli");
            ts2.add("yuvi");
            System.out.println(ts2);//[yuvi, saurav, sachin, kohli, dhoni]
      }
}
```

=> Insert StringBuffer object into treeset, perform sorting in Alphabetical Order.

eg#1.

```
import java.util.*;

class MyComparator implements Comparator
{
      @Override
      public int compare(Object obj1,Object obj2)
      {
            String s1=obj1.toString();
            String s2=obj2.toString();
            return -s1.compareTo(s2);

      }
}
public class Test
{
      public static void main(String[] args)
      {
            //TreeSet[Balanced Tree] -> Comparable :: DNS
            //public int compareTo(Object obj)
            TreeSet ts1 =new TreeSet();
            ts1.add(new StringBuffer("sachin"));
            ts1.add(new StringBuffer("saurav"));
            ts1.add(new StringBuffer("dhoni"));
            ts1.add(new StringBuffer("kohli"));
            ts1.add(new StringBuffer("yuvi"));

            System.out.println(ts1);//[dhoni,kohli,sachin,saurav,yuvi]

            System.out.println();

            //TreeSet[Balanced Tree] -> Comparator :: CSO
            //public abstract int compare(Object obj1,Object obj2);
            //public abstract boolean equals(java.lang.Object);
            TreeSet ts2 =new TreeSet(new MyComparator());
            ts2.add(new StringBuffer("sachin"));
            ts2.add(new StringBuffer("saurav"));
            ts2.add(new StringBuffer("dhoni"));
            ts2.add(new StringBuffer("kohli"));
            ts2.add(new StringBuffer("yuvi"));
            System.out.println(ts2);//[yuvi, saurav, sachin, kohli, dhoni]
      }
}
```

Write a java program to insert the String and StringBuffer object into TreeSet where sorting order is in increasing length order.
    if 2 objects have same length then consider their Alphabetical order

```
sample::
ts.add(new StringBuffer("A"));
ts.add(new StringBuffer("ABC"));
ts.add(new StringBuffer("AA"));
ts.add("XX");
ts.add("ABCE");
ts.add("A");

eg#1.
import java.util.*;

class MyComparator implements Comparator
{
      @Override
      public int compare(Object obj1,Object obj2)
      {
            String s1=obj1.toString();
            String s2=obj2.toString();


            int i1= s1.length();
            int i2= s2.length();

            //increasing length order
            if (i1<i2)
            {
                  //obj1 should come before obj2
                  return -1;
            }
            else if(i1>i2)
            {
                  //obj1 should come after obj2
                  return +1;
            }
            else
            {
                  //same length :: Alphabetical Order
                  return s1.compareTo(s2);
            }
      }
}
public class Test
{
      public static void main(String[] args)
      {
            //TreeSet[Balanced Tree] -> Comparator :: CSO
            //public abstract int compare(Object obj1,Object obj2);
            //public abstract boolean equals(java.lang.Object);
            TreeSet ts1 =new TreeSet(new MyComparator());
            ts1.add(new StringBuffer("A"));
            ts1.add(new StringBuffer("ABC"));
            ts1.add(new StringBuffer("AA"));
            ts1.add("XX");
            ts1.add("ABCE");
            ts1.add("A");
            System.out.println(ts1);//
      }
}
```

```
Output
[A, AA, XX, ABC, ABCE]

Note:
Comparable  :: By default the object we add into treeSet, the  corresponding class
should implement "Comparable" interface and
             the object should be homogenous Otherwise it would result in
"ClassCastException".


Comparator :: This interface is meant for Custom sorting, so the objects added to
TreeSet need not be "Homogenous" and need not
            implement "Comparable".
            We can add Non-Homogenous and Non-Comparable objects into TreeSet.




Scenario
========
When to go for Comparable and Comparator?
 1st category
    Predefined Comparable classes like String and Wrapper class
             => Default natural sorting order is already available
             => If not satisfied, then we need to go for Comparator

 2nd Category
    Predefined NonComparable classes like StringBuffer
             => Default natural sorting order not available so go for Comparator
only aways

 3rd Category
    Our Own classes like Employee,Student,Customer
            =>Person who is writing this classes are responsible for implementing
comparable
              interface to promote Natural sorting order.
            =>Person who is using this class,can define his own natural sorting
order
              by implementing Comparator interface.


Write a Program to Insert Employee Objects into the TreeSet where DNSO is Based on
Ascending Order of EmployeeId
and Customized Sorting Order is Based on Alphabetical Order  of Names.

eg#1.
import java.util.*;

class Employee implements Comparable
{
      int eid;
      String ename;

      //Parameterized Constructor
      Employee(int eid,String ename)
      {
            this.eid   = eid;
            this.ename = ename;
      }
      @Override
      public String toString()
      {
            return "{ "+eid + "----> " + ename+"}";
      }

      @Override
      public int compareTo(Object obj1)
      {
            //logic for Sorting based on eid
            int id1 = this.eid;

            Employee emp1 = (Employee)obj1;
            int id2 = emp1.eid;

            if (id1<id2)
            {
                  return -1;
            }
            else if(id1>id2)
            {
                  return +1;
            }
            else
            {
                  return 0;
            }

      }
}

class MyComparator implements Comparator
{
      @Override
      public int compare(Object obj1,Object obj2)
      {
            //logic for Sorting based on ename
            Employee e1 = (Employee) obj1;
            Employee e2 = (Employee) obj2;

            String s1 = e1.ename;
            String s2 = e2.ename;

            //DefaultNatural Sorting Order based on Alphabetical Order
            return s1.compareTo(s2);
      }
}

public class Test
{
      public static void main(String[] args)
      {
            Employee e1 = new Employee(10,"sachin");
            Employee e2 = new Employee(9,"lara");
            Employee e3 = new Employee(14,"ponting");
            Employee e4 = new Employee(7,"dhoni");
            Employee e5 = new Employee(18,"kohli");
            TreeSet ts1 = new TreeSet();
            ts1.add(e1);
            ts1.add(e2);//e2.compareTo(e1)
            ts1.add(e3);
            ts1.add(e4);
            ts1.add(e5);
            System.out.println("Default Natural Sorting Order:: Based on ID");
            System.out.println(ts1);

            System.out.println();


            TreeSet ts2 = new TreeSet(new MyComparator());
            ts2.add(e1);
            ts2.add(e2);
            ts2.add(e3);
            ts2.add(e4);
            ts2.add(e5);
            System.out.println("Customized Sorting Order:: Based on NAME");
            System.out.println(ts2);
      }
}
Output
Default Natural Sorting Order:: Based on ID
[{ 7----> dhoni}, { 9----> lara}, { 10----> sachin}, { 14----> ponting}, { 18---->
kohli}]

Customized Sorting Order:: Based on NAME
[{ 7----> dhoni}, { 18----> kohli}, { 9----> lara}, { 14----> ponting}, { 10---->
sachin}]

Comparable and Comparator
=========================
Comparable => Meant for default natural sorting order
Comparator => Meant for customized sorting order

Comparable => part of java.lang package
Comparator => part of java.util package

Comparable => only one method compareTo()
Comparator => 2 methods compare(),equals()

Comparable => It is implemented by Wrapper class and String class
Comparator => It is implemented by Collator and RuleBaseCollator(GUI based API)

Comparsion table of Set implemented Classes
============================================
 HashSet       => underlying data structure is HashTable
              duplicates not allowed
              insertion order not preserved
              Sorting order not preserved
              duplicates not allowed
              heterogenous elements allowed
              null  allowed

 LinkedHashset => underlying data structure is linkedhashset and HashTable
              duplicates not allowed
              inserted order preserved
                  Sorting order not preserved
```

duplicates not allowed
heterogenous elements allowed
null allowed

TreeSet => underlying data structure is balanced Tree
duplicates not allowed
insertion order not preserved
Sorting order not preserved
duplicates not allowed
heterogenous elements not allowed by default
null not allowed.

Difference b/w Iterator and Iterable
====================================
The target element in for-each loop should be Iterable object/array/Collection.

```
      for(datatype item:target){
            ..........
            .........
      }
```

=> An object is set to be iterable iff corresponding class implements java.lang.Iterable interface.
=> Iterable interface introduced in 1.5 version and it's contains only one method iterator().
Syntax : public Iterator iterator();

Note: Every array class and Collection interface already implements Iterable interface.

Difference between Iterable and Iterator:

Iterable
1. It is related to forEach loop
2. The target element in forEach loop should be Iterable.
3. Iterator present in java.lang package.
4. contains only one method iterator().
5. Introduced in 1.5 version.

Iterator
1. It is related to Collection.
2. We can use Iterator to get objects one by one from the collection.
3. Iterator present in java.util package.
4. contains 3 methods hasNext(), next(), remove()
5. Introduced in 1.2 version.