TreeSet:
 => The Underlying Data Structure is Balanced Tree.
 => Insertion Order is Not Preserved and it is Based on Some Sorting Order.
 => Heterogeneous Objects are Not Allowed. If we are trying to Insert we will get Runtime Exception Saying ClassCastException.
 => Duplicate Objects are Not allowed.
=> null Insertion is Possible (Only Once).
=> Implements Serializable and Cloneable Interfaces but Not RandomAccess Interface.

Constructors:
1) TreeSet t = new TreeSet();
      Creates an Empty TreeSet Object where all Elements will be Inserted According to Default Natural Sorting Order.
2) TreeSet t = new TreeSet(Comparator c);
      Creates an Empty TreeSet Object where all Elements will be Inserted According to
      Customized Sorting Order which is described by Comparator Object.
3) TreeSet t = new TreeSet(Collection c);
4) TreeSet t = new TreeSet(SortedSet s);

eg#1.
```
import java.util.TreeSet;
classTreeSetDemo {
      public static void main(String[] args) {
                  TreeSet t = new TreeSet();
                  t.add("A");
                  t.add("a");
                  t.add("B");
                  t.add("Z");
                  t.add("L");
                  t.add(new Integer(10));//ClassCastExcpetion
                  t.add(null);//RE: Exception in thread "main" java.lang.NullPointerException
                  System.out.println(t); //[A, B, L, Z, a]
      }
}
```

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);
       }
}
```
RE: Exception in thread "main" java.lang.ClassCastException: java.lang.StringBuffer cannot be cast to java.lang.Comparable

Note:
If we are Depending on Default Natural Sorting Order Compulsory Objects should be Homogeneous and Comparable.
Otherwise we will get RE: ClassCastException.
An object is said to be Comparable if and only if corresponding class implements Comparable interface.

All Wrapper Classes, String Class Already Implements Comparable Interface. But StringBuffer Class doesn't Implement Comparable Interface.
Hence we are ClassCastException in the Above Exampl

Comparable (I):
Comparable Interface Present in java.lang Package and it contains Only One Method compareTo().

obj1.compareTo(obj2)
Returns –ve if and Only if obj1 has to Come Before obj2.
Returns +ve if and Only if obj1 has to Come After obj2.
Returns 0 if and Only if obj1 and obj2 are Equal.

eg#1.

```
System.out.println("A".compareTo("Z")); //-ve value
System.out.println("Z".compareTo("K")); // +value
System.out.println("Z".compareTo("Z")); // zero
System.out.println("Z".compareTo(null));//NPE
```

Wheneverwe are Depending on Default Natural Sorting Order and if we are trying to Insert Elements then Internally JVM will
Call compareTo() to IdentifySorting Order.

```
TreeSet t = new TreeSet();
      t.add(“K”);
      t.add(“Z”); “Z”.compareTo(“K”);
      t.add(“A”); “A”.compareTo(“K”);
      t.add(“A”); “A”.compareTo(“A”);
      System.out.println(t);
```

Note: If we are Not satisfied with Default Natural Sorting Order OR if Default Natural Sorting Order is Not Already Available then
we can Define Our Own Sorting by using Comparator Object.

Comparator (I):
This Interface Present in java.util Package.
Methods: It contains 2 Methodscompare() and equals().

```
public  int compare(Object obj1, Object obj2);
```

Returns –ve if and Only if obj1 has to Come Before obj2.
Returns +ve if and Only if obj1 has to Come After obj2.
Returns 0 if and Only if obj1 and obj2 are Equal.

```
public boolean equals(Object o);
```

Whenever we are implementing Comparator Interface Compulsory we should Provide Implementation for compare().
Implementing equals() is Optional because it is Already Available to Our Class from Object Class through Inheritance.

```
import java.util.*;
class     TreeSetDemo {
public static void main(String[] args) {
            TreeSet t = new TreeSet(new MyComparator());//line-1
                  t.add(10);
                  t.add(0);
                  t.add(15);
                  t.add(5);
                  t.add(20);
                  t.add(20);
                  System.out.println(t);//[20, 15, 10, 5, 0]
      }
```

```
}
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;
            }
}
```

At Line 1 if we are Not Passing Comparator Object as an Argument then Internally JVM will Call compareTo(),
Which is Meant for Default Natural Sorting Order (Ascending Order). In this Case the Output is [0, 5, 10, 15, 20].

At Line 1 if we are Passing Comparator Object then JVM will Call compare() Instead of compareTo().
Which is Meant for Customized Sorting (Descending Order). In this Case the Ouput is [20, 15, 10, 5, 0]

Various Possible Implementations of compare():

```
public int compare(Object obj1, Object obj2) {
            Integer I1 = (Integer)obj1;
            Integer I2 = (Integer)obj2;
            return I1.compareTo(I2);
            return -I1.compareTo(I2);
            return I2.compareTo(I1);
            return -I2.compareTo(I1);
            return +1;
            return -1;
            return 0;
}
```

Output:

1. Ascending order
2. Descending order
3. Descending order
4. Ascending order
5. insertion order
6. reverse of insertion order
7. only first element will be inserted.

Write a Program to Insert String Objects into the TreeSet where the Sorting Order is of
Reverse of Alphabetical Order:

```
import java.util.*;
class  TreeSetDemo {
      public static void main(String[] args) {
                  TreeSet t = new TreeSet(new MyComparator());
                  t.add("sachin");
                  t.add("ponting");
                  t.add("sangakara");
                  t.add("fleming");
                  t.add("lara");
                  System.out.println(t);
```

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

Write a Program to Insert StringBuffer Objects into the TreeSet where Sorting Order is Alphabetical Order:

```
import  java.util.*;
class  TreeSetDemo {
	public static void main(String[] args) {
			TreeSet t = new TreeSet(new MyComparator1());
			t.add(new StringBuffer("A"));
			t.add(new StringBuffer("Z"));
			t.add(new StringBuffer("K"));
			t.add(new StringBuffer("L"));
			System.out.println(t);
	}
}
class MyComparator1 implements Comparator {
	publicint compare(Object obj1, Object obj2) {
			String s1 = obj1.toString();
			String s2 = obj2.toString();
			return s1.compareTo(s2); //[A, K, L, Z]
	}
}
```

Write a Program to Insert String and StringBuffer Objects into the TreeSet where Sorting Order is Increasing Length Order.
If 2 Objects having Same Length then Consider their Alphabetical Order:

```
import  java.util.*;
class  TreeSetDemo {
public static void main(String[] args) {
		TreeSet t = new TreeSet(new MyComparator());
		t.add("A");
		t.add(new StringBuffer("ABC"));
		t.add(new StringBuffer("AA"));
		t.add("XX");
		t.add("ABCE");
		t.add("A");
		System.out.println(t);
	}
}
class  MyComparator implements Comparator {
      public      int compare(Object obj1, Object obj2) {
	String s1 = obj1.toString();
	String s2 = obj2.toString();
	int i1 = s1.length();
	int i2 = s2.length();
	if(i1 < i2)return -1;
	else if(i1 > i2)return 1;
	else  return s1.compareTo(s2);
```

```
	}
}
```
Note:
If we are Depending on Default Natural Sorting Order Compulsory Objects should be Homogeneous and Comparable Otherwise we will
get RE: ClassCastException.

If we defining Our Own Sorting by Comparator then Objects Need Not be Homogeneous and Comparable. That is we can
Add Heterogeneous Non Comparable Objects to the TreeSet

When we go for Comparable and When we go for Comparator:
Comparable Vs Comparator:

=> For Predefined Comparable Classes (Like String) Default Natural Sorting Order is Already Available. If we are Not satisfied with that we
	can Define Our Own Sorting by Comparator Object.
=> For Predefine Non- Comparable Classes (Like StringBuffer) Default Natural Sorting Order is Not Already Available.
	If we want to Define Our Own Sorting we can Use Comparator Object.
=> For Our Own Classes (Like Employee) the Person who is writing Employee Class he is Responsible to Define Default Natural Sorting
	Order by implementing Comparable Interface.
=> The Person who is using Our Own Class if he is Not satisfied with Default Natural Sorting Order he can Define his Own Sorting by using Comparator Object.
	If he is satisfied with Default Natural Sorting Order then he can Use Directly Our Class.

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:

```
import  java.util.*;
class Employee implements Comparable {
            String name;
            int   eid;
            Employee(String name, inteid) {
                  this.name = name;
                  this.eid = eid;
            }
      public String toString() { return name+"-----"+eid;}
      public int compareTo(Object obj) {
            int eid1 = this.eid;
            Employee e = (Employee)obj;
            int eid2 = e.eid;
            if(eid1 < eid2) return -1;
            else if(eid1 > eid2) return 1;
            else return 0;
      }
}
class     Test {
      public static void main(String[] args) {

            Employee e1 = new Employee("sachin", 10);
            Employee e2 = new Employee("ponting", 14);
            Employee e3 = new Employee("lara", 9);
            Employee e4 = new Employee("flintoff", 17);
            Employee e5 = new Employee("anwar", 23);
```

```
            TreeSet t = new TreeSet();
            t.add(e1);
            t.add(e2);
            t.add(e3);
            t.add(e4);
            t.add(e5);
            System.out.println(t);

            TreeSet t1 = new TreeSet(new MyComparator());
            t1.add(e1);
            t1.add(e2);
            t1.add(e3);
            t1.add(e4);
            t1.add(e5);
            System.out.println(t1);
      }
}
class    MyComparator implements Comparator {
      public      int compare(Object obj1, Object obj2) {
            Employee e1 = (Employee) obj1;
            Employee e2 = (Employee) obj2;
            String s1 = e1.name;
            String s2 = e2.name;
            return s1.compareTo(s2);
      }
}
```

Comparison of Comparable and Comparator:

Comparable
Present in java.lang Package
It is Meant for Default Natural Sorting Order.
Defines Only One Method compareTo()
All Wrapper Classes and String Class implements Comparable Interface.

Comparator
Present in java.util Package
It is Meant for Customized Sorting Order.
Defines 2 Methods compare() and equals().
The Only implemented Classes of Comparator are Collator and RuleBaseCollator.