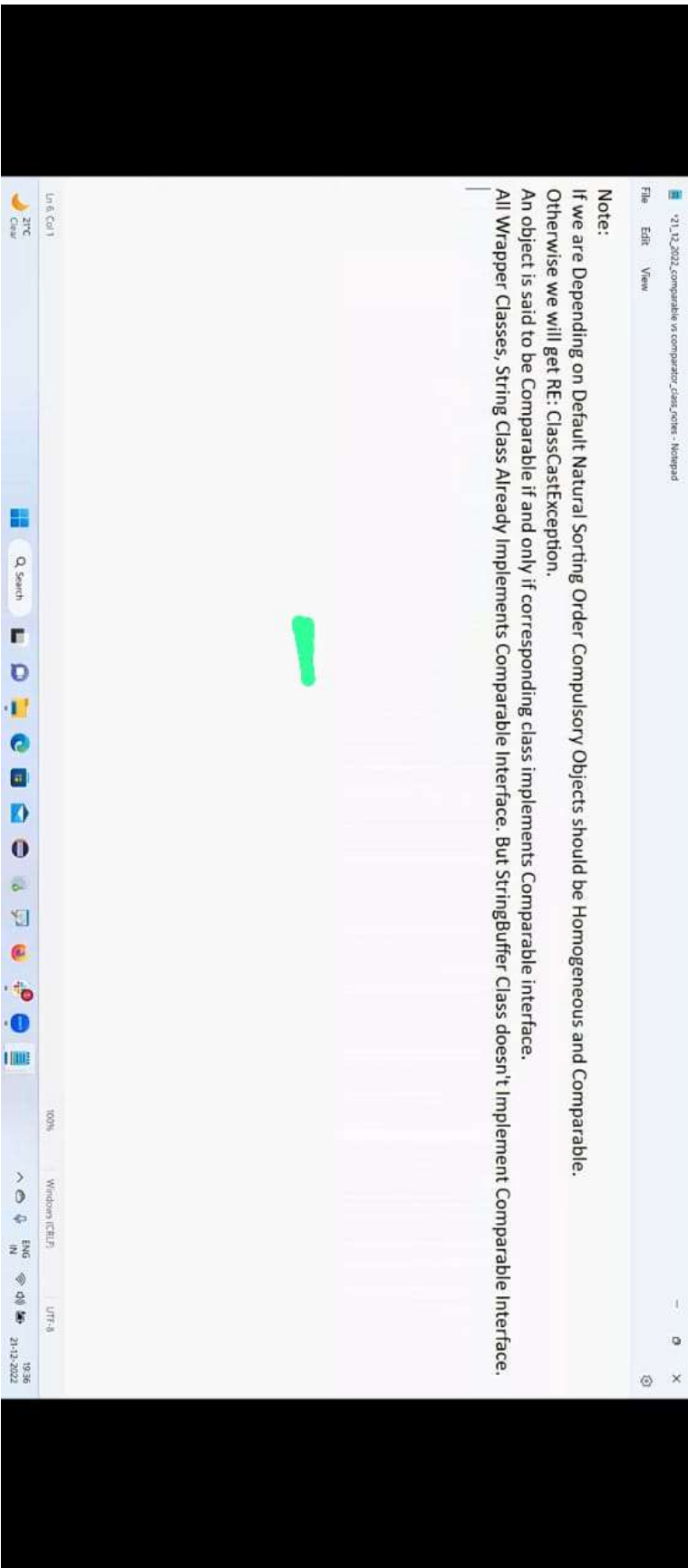



43Java_StreamApi



```
1 //
2 interface Comparable{
3     public int compareTo(Object obj);
4 }
5
6 If we keep objects inside TreeSet, internally JVM uses compareTo() and it sorts the
7 object, based on the sorting result the object will be stored in TreeSet.
8
9 class Test
10 {
11     public static void main(String[] args)
12     {
13         TreeSet ts = new TreeSet();
14         ts.add("A");
15         ts.add("Z");
16         System.out.println("A".compareTo("Z"));
17         System.out.println("Z".compareTo("K"));
18         System.out.println("Z".compareTo("Z"));
19         System.out.println("Z".compareTo(null));
20     }
21 }
```



```

7  object, based on the sorting result the object will be stored in TreeSet.
8
9  obj1.compareTo(obj2)
10      |=> returns -ve value, if obj1 has to come before obj2
11      |=> returns +ve value, if obj1 has to come after obj2
12      |=> returns 0 value, if both obj1 and obj2 are equal
13  */
14  class Test
15  {
16      public static void main(String[] args)
17      {
18          TreeSet ts = new TreeSet();
19          ts.add("A");
20          ts.add("Z");
21          ts.add("K");
22          System.out.println(ts);
23
24          System.out.println("A".compareTo("Z"));
25          System.out.println("Z".compareTo("K"));
26          System.out.println("Z".compareTo("Z"));
27          System.out.println("Z".compareTo(null));
28      }
29  }

```

FileViewTools

ClipboardImageToolsBrushesShapesSizeColors

100%21-12-2022

```
System.out.println("A".compareTo("Z"));// -ve value
System.out.println("Z".compareTo("K"));// +ve value
System.out.println("Z".compareTo("Z"));// 0

TreeSet ts = new TreeSet();
ts.add("A");
ts.add("Z");
ts.add("L");
ts.add("K");
ts.add("B");

System.out.println(ts); [A B K L Z]
```

Rules while constructing a binary tree

- ve means in binary tree the node should be to the left
- +ve means in binary tree the node should be to the right
- zero means in binary tree the nodes are duplicated

```
graph TD
    A((A)) -- +ve --> Z((Z))
    A -- -ve --> null1(( ))
    Z -- +ve --> null2(( ))
    Z -- -ve --> L((L))
    L -- +ve --> null3(( ))
    L -- -ve --> K((K))
    K -- +ve --> null4(( ))
    K -- -ve --> B((B))
```

```
System.out.println("A".compareTo("Z"));// -ve value
System.out.println("Z".compareTo("K"));// +ve value
System.out.println("Z".compareTo("z"));// 0
```

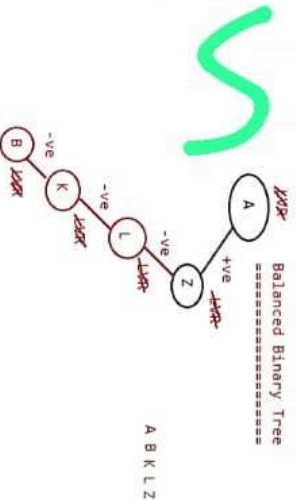
```
TreeSet ts = new TreeSet();
```

```
ts.add("A");      "Z": compareTo("A");
ts.add("Z");      "L": compareTo("A");
ts.add("L");      "K": compareTo("A");
ts.add("K");      "K": compareTo("L");

ts.add("B");      "B": compareTo("A");
                "B": compareTo("L");
                "B": compareTo("K");
```

Rules while constructing a binary tree

- ve means in binary tree the node should be to the left
- +ve means in binary tree the node should be to the right
- zero means in binary tree the nodes are duplicated



```
D:\Wrapper classes>javap java.util.Comparator
Compiled from "Comparator.java"
public interface java.util.Comparator<T> {
    public abstract int compare(T, T);
    public abstract boolean equals(java.lang.Object);

    public java.util.Comparator<T> reversed();
    public java.util.Comparator<T> thenComparing(java.util.Comparator<? super T>, ? extends java.util.Comparator<? super U>);
    public <U> java.util.Comparator<T> thenComparingInt(java.util.function.ToIntFunction<? super T>, ? extends U>);
    public java.util.Comparator<T> thenComparingLong(java.util.function.ToLongFunction<? super T>);
    public java.util.Comparator<T> thenComparingDouble(java.util.function.ToDoubleFunction<? super T>);
    public static <T> extends java.lang.Comparable<? super T> java.util.Comparator<T> naturalOrder();
    public static <T> java.util.Comparator<T> nullsFirst(java.util.Comparator<? super T>);
    public static <T> java.util.Comparator<T> nullsLast(java.util.Comparator<? super T>);
    public static <T, U> java.util.Comparator<T> comparing(java.util.function.Function<T, ? extends U>, java.util.Comparator<? super U>);
    public static <T> java.util.Comparator<T> comparingInt(java.util.function.ToIntFunction<? super T>);
    public static <T> java.util.Comparator<T> comparingLong(java.util.function.ToLongFunction<? super T>);
    public static <T> java.util.Comparator<T> comparingDouble(java.util.function.ToDoubleFunction<? super T>);
}
```

D:\Wrapper classes>



21_12_2022_comparable vs comparator_class_note - Notepad

File Edit View

}

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 (can be Ascending / Descending Order).
In this Case the Output is [20, 15, 10, 5, 0]

16

Ln 114, Col 45

19°C
Clear

Search



100%

Windows (CTRL)

ENG

IN

UTF-8

21-12-2022


```
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); // [A,K,Z] => Sorting is ascending order
```

Note:

For String default natural sorting order is "Ascending order".

For Number default natural sorting order is "Ascending order"

```
Comparator()
```

```
=====
```

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.

```
public interface java.util.Comparator<T> {
```

```
public abstract int compare(T, T);
```

```
public abstract boolean equals(java.lang.Object);
```

```
} I
```



```
21.12.2022_comparable vs comparator class, notes - Notepad
File Edit View

public interface java.util.Comparator<T>{
    public abstract int compare(T, T);
    public abstract boolean equals(java.lang.Object);
}


class MyComparator implements Comparator{

    @Override
    public int compare(Object obj1, Object obj2){
        ;;;;
    }

    compare(Object obj1, Object obj2) |
    |=> returns -ve value, if obj1 has to come before obj2
    |=> returns +ve value, if obj1 has to come after obj2
    |=> returns 0 value, if both obj1 and obj2 are equal

}

1
```



21_12_2022,comparator vs comparator.class,notes - Notepad

File Edit View

```
public interface java.util.Comparator<T> {  
    public abstract int compare(T, T);  
    public abstract boolean equals(java.lang.Object);  
}
```

Comparator (I):

This Interface Present in java.util Package.

Methods: It contains 2 Methods compare() 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.

Ln 86, Col 1

27°C
Clear

Search



100%

Windows (CTRL)

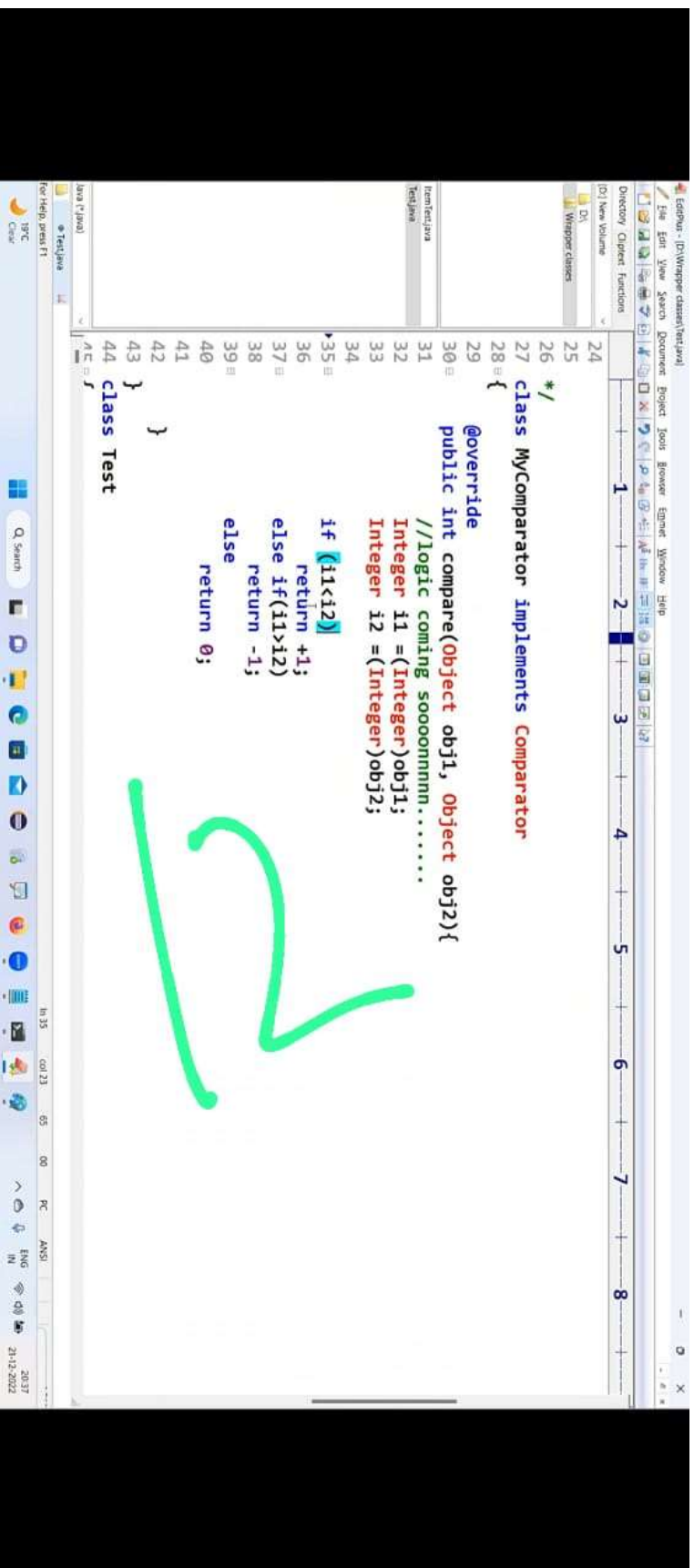
UTF-8

ENG
IN
21-12-2022
20:16

```
1 //  
2 //Natural Sorting order  
3 interface Comparable{  
4     public int compareTo(Object obj);  
5 }  
6 obj1.compareTo(obj2)  
7     |=> returns -ve value, if obj1 has to come before obj2  
8     |=> returns +ve value, if obj1 has to come after obj2  
9     |=> returns 0 value, if both obj1 and obj2 are equal  
10  
11  
12 //Customized sorting  
13 interface Comparator{  
14     public int compare(Object obj1, Object obj2);  
15     public boolean equals(Object obj);  
16 }  
17 compare(Object obj1, Object obj2)  
18     |=> returns -ve value, if obj1 has to come before obj2  
19     |=> returns +ve value, if obj1 has to come after obj2  
20     |=> returns 0 value, if both obj1 and obj2 are equal  
21  
22
```

Q

```
D:\Wrapper\classes>javap java.util.TreeSet
Compiled from "TreeSet.java"
public class java.util.TreeSet<E> extends java.util.AbstractSet<E> implements java.util.NavigableSet<E>, java.lang.Cloneable,
    java.io.Serializable {
    java.util.TreeSet(java.util.NavigableMap<E, java.lang.Object>);
    public java.util.TreeSet();
    public java.util.TreeSet(java.util.Comparator<? super E>);
    public java.util.TreeSet(java.util.Collection<? extends E>);
    public java.util.Iterator<E> iterator();
    public java.util.Iterator<E> descendingIterator();
    public java.util.NavigableSet<E> descendingSet();
    public int size();
    public boolean isEmpty();
    public boolean contains(java.lang.Object);
    public boolean add(E);
    public boolean remove(java.lang.Object);
    public void clear();
    public boolean addAll(java.util.Collection<? extends E>);
    public java.util.NavigableSet<E> subset(E, boolean, E, boolean);
    public java.util.NavigableSet<E> headSet(E, boolean);
    public java.util.NavigableSet<E> tailSet(E, boolean);
    public java.util.SortedSet<E> subSet(E, E);
    public java.util.SortedSet<E> headSet(E);
    public java.util.SortedSet<E> tailSet(E);
    public java.util.Comparator<? super E> comparator();
    public E first();
    public E last();
    public E lower(E);
    public E floor(E);
    public E ceiling(E);
```



21.12.2022, comparable vs comparator_images - Paint

FileView

Clipboard

Image

Tools

Brushes

Shapes

Size

Colors

```
TreeSet ts = new TreeSet(new MyComparator());
ts.add(10);
ts.add(0); compare(0,10)
ts.add(15); compare(15,10)
ts.add(5); compare(5,10)
ts.add(20); compare(20,10)
ts.add(20); compare(20,15)
ts.add(20); compare(20,20)
```

System.out.println(ts); [A B K L Z]

compare(Object obj1, Object obj2)
|=> returns -ve value, if obj1 has to come before obj2
|=> returns +ve value, if obj1 has to come after obj2
|=> returns 0 value, if both obj1 and obj2 are equal

If (11<12)
return +1;
else if(11>12)
return -1;
else
return 0;

20 15 10 5 0

```
graph TD
    10((10)) -- "+ve" --> 0((0))
    10 -- "-ve" --> 15((15))
    15 -- "-ve" --> 20((20))
    0 -- "-ve" --> 5((5))
```

```
graph LR
    K((K)) -- "-ve" --> B((B))
    B -- "-ve" --> 20((20))
```

100%

ENG

21-12-2022

```
D:\Wrapper classes>java Test
Every comparison the compare() is called
Every comparison the compare() is called
Every comparison the compare() is called
Every comparison the compare() is called
Every comparison the compare() is called
Every comparison the compare() is called
Every comparison the compare() is called
Every comparison the compare() is called
Every comparison the compare() is called
Every comparison the compare() is called
Every comparison the compare() is called
[20, 15, 10, 5, 0]
```

```
D:\Wrapper classes>javac Test.java
Note: Test.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

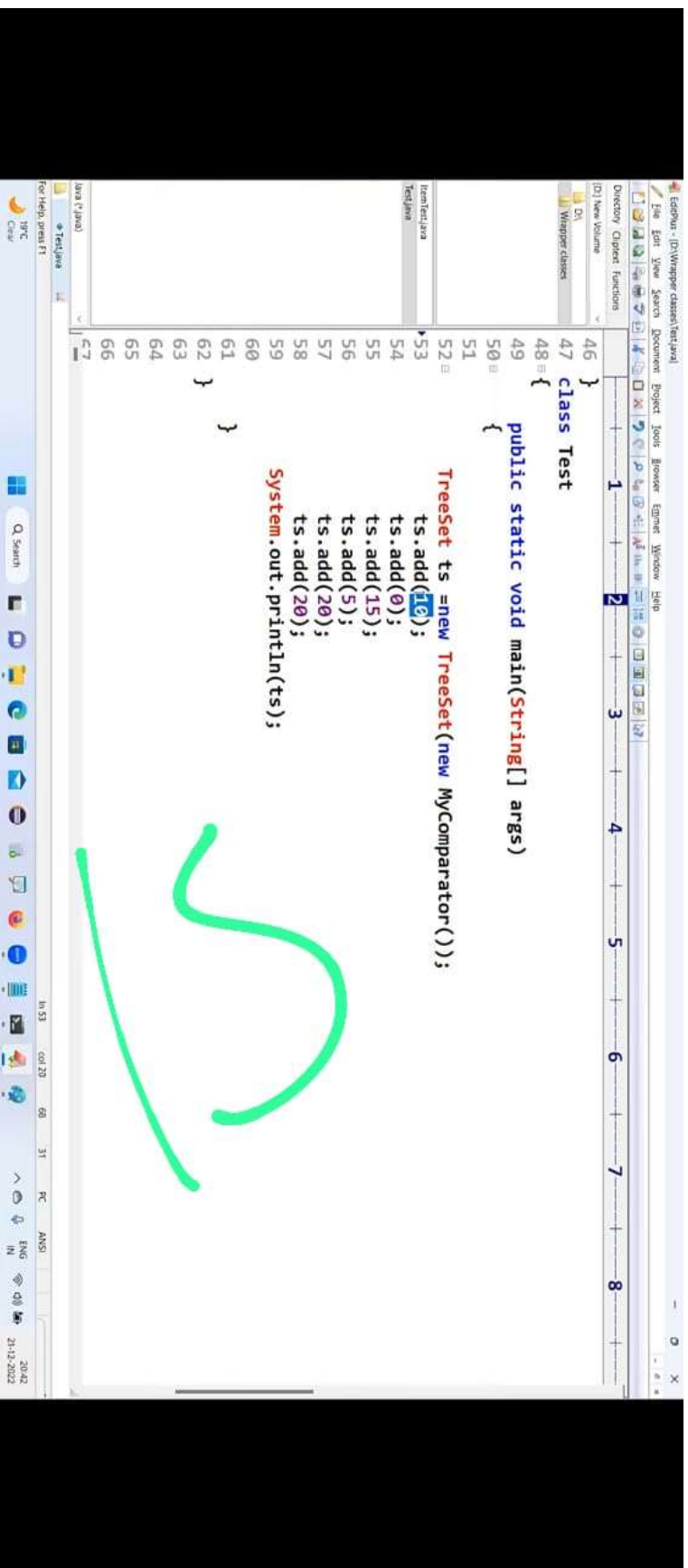
```
D:\Wrapper classes>java Test
Every comparison the compare() is called
Every comparison the compare() is called
Every comparison the compare() is called
Every comparison the compare() is called
Every comparison the compare() is called
Every comparison the compare() is called
Every comparison the compare() is called
Every comparison the compare() is called
Every comparison the compare() is called
Every comparison the compare() is called
Every comparison the compare() is called
[0, 5, 10, 15, 20]
```

4

D:\Wrapper classes>

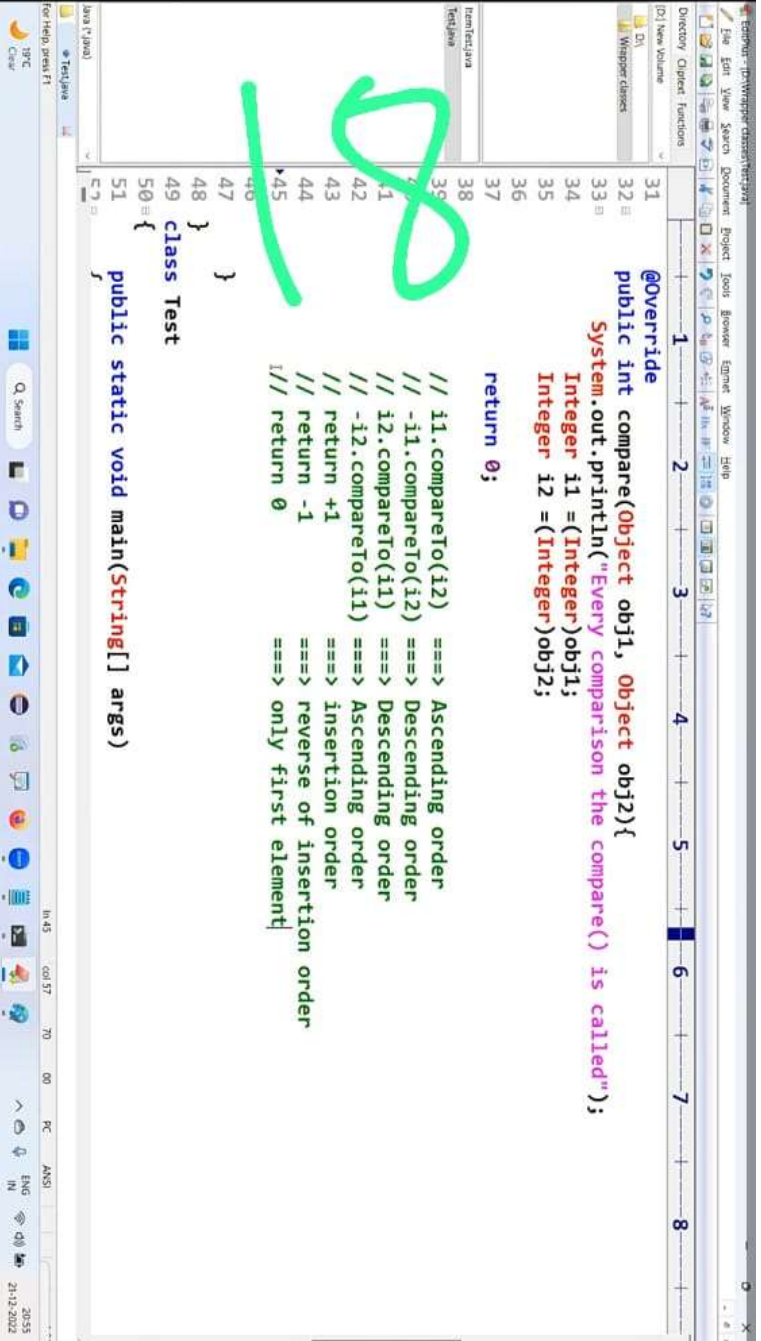


2042
21-12-2022



```
28 */
29 class MyComparator implements Comparator
30 {
31     @Override
32     public int compare(Object obj1, Object obj2){
33         System.out.println("Every comparison the compare() is called");
34         Integer i1 =(Integer)obj1;
35         Integer i2 =(Integer)obj2;
36
37         return -i2.compareTo(i1);//Ascending order
38
39         // i1.compareTo(i2) ==> Ascending order
40         // -i1.compareTo(i2) ==> Descending order
41         // i2.compareTo(i2) ==> Descending order
42         // -i2.compareTo(i1) ==> Ascending order
43
44     }
45 }
46 class Test
47 {
48     public static void main(String[] args)
```

23



```
21_12_2022_comparator_vs_comparator_class_note - Notepad
File Edit View
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("flaming");
        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:

```
Editor - [D:\Wrapper classes\Test.java]
File Edit View Search Document Project Tools Browser Emmet Window Help
Directory: C:\Program Files\Java\jdk-11.0.10\bin
D:\ New Volume
Wrapper classes
Test.java
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57

return -s1.compareTo(s2);

}

class Test
{
    public static void main(String[] args)
    {
        TreeSet ts = new TreeSet(new MyComparator());
        ts.add("sachin");
        ts.add("ponting");
        ts.add("gayle");
        ts.add("deveilliers");
        ts.add("jaysuriya");
        System.out.println(ts); //Ascending order
    }
}
```

20

The screenshot shows an IDE with the following code:

```

36         return -s1.compareTo(s2);
37     }
38 }
39 class Test
40 {
41     public static void main(String[] args)
42     {
43         TreeSet ts = new TreeSet(new MyComparator());
44         ts.add(new StringBuffer("sachin"));
45         ts.add(new StringBuffer("ponting"));
46         ts.add(new StringBuffer("gayle"));
47         ts.add(new StringBuffer("devealiars"));
48         ts.add(new StringBuffer("jaysuriya"));
49         System.out.println(ts); //Ascending order
50     }
51 }
52 }
53 }
54 }
55 }
56 }
57 }

```

A large green handwritten number '2' is drawn over the bottom right portion of the code.

codebar - 1D\Wrapper\class\MyComparator.java

File Edit View Search Document Project Tools Browser Logout Window Help

Directory Object Functions

D:\New Volume

Wrapper\classes

ItemTest.java

Test.java

28 class MyComparator implements Comparator

29 {

30

31 @Override

32 public int compare(Object obj1, Object obj2){

33 String s1 =(String)obj1;

34 String s2 = obj2.toString();

35

36 //reverse of natural order

37 return -s1.compareTo(s2);

38 }

39 }

40 class Test

41 {

42 public static void main(String[] args)

43 {

44 TreeSet ts =new TreeSet(new MyComparator());

45 ts.add(new StringBuffer("sachin"));

46 ts.add(new StringBuffer("ponting"));

47 ts.add(new StringBuffer("gayle"));

48 ts.add(new StringBuffer("deveillers"));

49 ts.add(new StringBuffer("sachin"));

50 }

1 2 3 4 5 6 7 8

25

19°C

100%

Search

ln 44

col 51

59

28

PC

ANSI

ENG

IN

21-12-2022


```
28 class MyComparator implements Comparator
29 {
30     @Override
31     public int compare(Object obj1, Object obj2){
32         String s1 = obj1.toString();
33         String s2 = obj2.toString();
34
35         //natural sorting order
36         return s1.compareTo(s2);
37     }
38 }
39 class Test
40 {
41     public static void main(String[] args)
42     {
43
44         TreeSet ts = new TreeSet(new MyComparator());
45         ts.add(new StringBuffer("sachin"));
46         ts.add(new StringBuffer("ponting"));
47         ts.add(new StringBuffer("gayle"));
48         ts.add(new StringBuffer("deveillers"));
49         ts.add(new StringBuffer("sachin"));
50     }
}
```

23

```
41         return +1;
42     else
43         return s1.compareTo(s2);
44     }
45 }
46 class Test
47 {
48     public static void main(String[] args)
49     {
50
51         TreeSet t = new TreeSet(new MyComparator());
52         t.add("A");
53         t.add(new StringBuffer("ABC"));
54         t.add(new StringBuffer("AA"));
55         t.add("XX");
56         t.add("ABCE");
57         t.add("A");
58         System.out.println(t);
59     }
60 }
61 }
```

```

29 = {
30     @Override
31     public int compare(Object obj1, Object obj2){
32         String s1 = obj1.toString();
33         String s2 = obj2.toString();
34
35         int i1 = s1.length();
36         int i2 = s2.length();
37
38         if (i1<i2)
39             return -1;
40         else if (i1>i2)
41             return +1;
42         else
43             return s1.compareTo(s2);
44     }
45 }
46 class Test
47 {
48     public static void main(String[] args)
49     {

```

25

```
21_12_2022_comparable_vs_comparator_class_note - Notepad
File Edit View

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 use TreeSet(), then the condition is

- a. Object should be homogenous.
- b. Object should be comparable(class should implement Comparable()).

if we are use TreeSet(Comparator c) then what is the condition?

- a. Object need not be homogenous.
- b. Object need not implement Comparable.

96

if we are use TreeSet(Comparator c) then what is the condition?

- Object need not be homogenous.
- Object need not implement Comparable.

When to go for Comparable interface and When to go Comparator interface?

Ans. Predefined Comparable classes like String, Wrapper class ==> Default natural sorting is already available

if we are not happy with natural sorting order, we want customization then we need to go for "Comparator()".

For Predefined Non-Comparable class like StringBuffer => Comparator() is used for both natural sorting order and customized sorting order.

For userdefined class like Employee, Student ==> Developer if he comes up with own logic of sorting, then he should implement comparable()

27

For Predefined Non-Comparable class like StringBuffer => Comparator() is used for both natural sorting order and customized sorting order.

For userdefined class like Employee, Student =====> Developer if he comes up with own logic of sorting, then he should implement Comparable() and give it as a ready made logic.

```
class Employee implements Comparable
{
    int id;
    String name;
    int age;

    public int compareTo(Object obj){
        //sorting is done based on "id"
        ....
    }
}
```

*21_12_2022_comparable vs comparator_class_notes - Notepad
File Edit View
use "Comparator".

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 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 If he is satisfied with Default Natural Sorting Order then he can Use Directly Our Class.

29

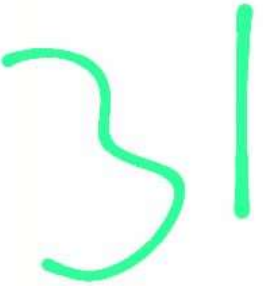

```

47
48
49
50 class Test
51 {
52     public static void main(String[] args)
53     {
54
55         TreeSet ts = new TreeSet();
56         ts.add(new Employee("sachin",10));
57         ts.add(new Employee("ponting",14));
58         ts.add(new Employee("gayle",99));
59         ts.add(new Employee("deveillers",17));
60
61     }
62 }
63
64
65
66
67
68
69

```

30

```
47         return name + " ==> " + id;
48     }
49 }
50 class Test
51 {
52     public static void main(String[] args)
53     {
54
55         TreeSet ts = new TreeSet();
56         ts.add(new Employee("sachin",10));
57         ts.add(new Employee("ponting",14));
58         ts.add(new Employee("gayle",99));
59         ts.add(new Employee("deveaiers",17));
60         System.out.println(ts);
61     }
62 }
63 }
64
65
66
67
68
```



```
35
36 class Employee implements Comparable<Employee>
37 {
38     String name;
39     int eid;
40
41     Employee(String name, int eid){
42         this.name = name;
43         this.eid = eid;
44     }
45
46     public String toString(){
47         return name + " ==> " + eid;
48     }
49 }
50 class Test
51 {
52     public static void main(String[] args)
53     {
54         TreeSet ts = new TreeSet();
55         ts.add(new Employee("Arshad", 101));
56     }
57 }
```

32

```
62 class Test
63 {
64     public static void main(String[] args)
65     {
66
67         TreeSet ts = new TreeSet();
68         Employee e1 = new Employee("sachin", 10);
69         Employee e2 = new Employee("ponting", 14);
70         Employee e3 = new Employee("gayle", 99);
71         Employee e4 = new Employee("deveilliers", 17);
72         ts.add(e1);
73         ts.add(e2);
74         ts.add(e3);
75         ts.add(e4);
76
77         System.out.println(ts);
78     }
79 }
80
81
82
```

33

```
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71

@Override
public int compareTo(Object obj1){
    //DNSO -> based on id, sort the objects

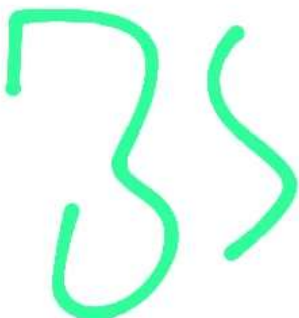
    int id1 = this.eid; //id1 -> FirstEmployee object id
    Employee emp = (Employee) obj1;
    int id2 = emp.eid; // id2 -> SecondEmployee Object id

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

class Test {
    public static void main(String[] args) {
        TreeSet ts = new TreeSet();
        Employee e1 = new Employee("sachin", 10);
        Employee e2 = new Employee("manoj", 15);
    }
}
```

4

```
26
27 */
28 class MyComparator implements Comparator
29 {
30     @Override
31     public int compare(Object obj1, Object obj2){
32         //Customization -> Sort based on name
33         Employee e1 =(Employee)obj1;
34         Employee e2 =(Employee)obj2;
35
36         String i1 = e1.name;
37         String i2 = e2.name;
38
39         return i1.compareTo(i2);
40     }
41 }
42
43 class Employee implements Comparable
44 {
45     String name;
46     int eid;
47 }
```



```
Employee e2 = new Employee("ponting",14);
Employee e3 = new Employee("gayle",99);
Employee e4 = new Employee("deveillers",17);
ts.add(e1); //e1.compareTo(e2);
ts.add(e2);
ts.add(e3);
ts.add(e4);
System.out.println(ts);

System.in.read();

System.out.println();

//Customized Sorting -> based on name
TreeSet ts1 = new TreeSet(new MyComparator());
ts1.add(e1);
ts1.add(e2); //compare(e1,e2)
ts1.add(e3);
ts1.add(e4);
System.out.println(ts1);
```

36

Command Prompt

X + v

- 0 X

D:\Wrapper classes>javac Test.java

Note: Test.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

D:\Wrapper classes>java Test

[sachin ==> 10, ponting ==> 14, devaliers ==> 17, gayle ==> 99]

[devaliers ==> 17, gayle ==> 99, ponting ==> 14, sachin ==> 10]

D:\Wrapper classes>

37

ITC
C++

Search

Windows taskbar icons including File Explorer, Edge, and various application shortcuts.

System tray area showing language (ENG), network, and date/time (21-12-2022, 22:16).

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 RuleBasedCollator.

1

38

21_12_2022,comparable vs comparator, class, java - Notepad

File Edit View

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 compareTo() and equals().

The Only Implemented Classes of Comparator are Collator and RuleBasedCollator.

inbuilt functional interfaces, StreamAPI, JodhAPI, Collection using lambda expression

io(filehandling), serialization and deserialization, packages, reflectionapi, gc, innerclass

39

Ln 39, Col 89

17°C
Clear

Search



100%

Windows (CTRL)

UTF-8

ENG
IN
21-12-2022 22:21

Question

Consider below code:

```
public class Test {  
    static Double d1; // d1 = null  
    static int x = d1.intValue(); // null.intValue() ---> NullPointerException  
  
    public static void main(String[] args) {  
        System.out.println("HELLO");  
    }  
}
```

On execution, does Test class print "HELLO" on to the console?

- A. Yes HELLO is printed on the console
- B. NO Hello is not printed on the console

Answer: B

Q5

File Edit View

Question

Consider below code:

```
public class Test {  
    static Double d1; // static variable ==> d1 = null  
    int x = d1.intValue(); // instance variable ==> only upon creating an object  
  
    public static void main(String[] args) {  
        System.out.println("HELLO"); // HELLO  
    }  
}
```

On execution, does Test class print "HELLO" on to the console?

A. Yes HELLO is printed on the console

B. NO Hello is not printed on the console

Answer: A

A

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

```
    Error obj = new Error();
```

```
    boolean flag1 = obj instanceof RuntimeException; //Line n1
```

```
    boolean flag2 = obj instanceof Exception; //Line n2
```

```
    boolean flag3 = obj instanceof Error; //Line n3
```

```
    boolean flag4 = obj instanceof Throwable; //Line n4
```

```
    System.out.println(flag1 + " " + flag2 + " " + flag3 + " " + flag4);
```

```
}
```

```
}
```

A. Compilation Error

B. false:false:true:true

C. false:true:true:true

D. true:true:true:true

E. false:true:true:false

String s = "sachin";

System.out.println(s instanceof String); //true

System.out.println(s instanceof StringBuffer); //CE

System.out.println(s instanceof Runnable); //false

System.out.println(null instanceof StringBuffer); //false

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

```
    Error obj = new Error();
```

```
    boolean flag1 = obj instanceof RuntimeException; //Line n1
```

```
    boolean flag2 = obj instanceof Exception; //Line n2
```

```
    boolean flag3 = obj instanceof Error; //Line n3
```

```
    boolean flag4 = obj instanceof Throwable; //Line n4
```

```
    System.out.println(flag1 + " " + flag2 + " " + flag3 + " " + flag4);
```

```
}
```

- }
A. Compilation Error
B. false:false:true:true
C. false:true:true:true
D. true:true:true:true
E. false:true:true:false

Note: Error and RuntimeException no relation in hierarchy as parent and child
Error and Exception no relation in hierarchy as parent and child

Answer: A

```
String s = "sachin",  
System.out.println(s instanceof String); //true
```

Ln 41, Col 60



100%

Windows (CTRL)

| UTF-8

ENG IN 21-12-2022 22:39


```
D:\Wrapper\classes>javap java.lang.RuntimeException
Compiled from "RuntimeException.java"
public class java.lang.RuntimeException extends java.lang.Exception {
    static final long serialVersionUID;
    public java.lang.RuntimeException();
    public java.lang.RuntimeException(java.lang.String);
    public java.lang.RuntimeException(java.lang.Throwable);
    protected java.lang.RuntimeException(java.lang.Throwable, boolean, boolean);
}
```

D:\Wrapper\classes>jav

44

```
D:\Wrapper\classes>javap java.lang.RuntimeException
Compiled from "RuntimeException.java"
public class java.lang.RuntimeException extends java.lang.Exception {
    static final long serialVersionUID;
    public java.lang.RuntimeException();
    public java.lang.RuntimeException(java.lang.String);
    public java.lang.RuntimeException(java.lang.String, java.lang.Throwable);
    protected java.lang.RuntimeException(java.lang.String, java.lang.Throwable, boolean, boolean);
}

D:\Wrapper\classes>javap java.lang.Exception
Compiled from "Exception.java"
public class java.lang.Exception extends java.lang.Throwable {
    static final long serialVersionUID;
    public java.lang.Exception();
    public java.lang.Exception(java.lang.String);
    public java.lang.Exception(java.lang.String, java.lang.Throwable);
    public java.lang.Exception(java.lang.Throwable);
    protected java.lang.Exception(java.lang.String, java.lang.Throwable, boolean, boolean);
}

D:\Wrapper\classes>javap java.lang.Throwable
Compiled from "Throwable.java"
public class java.lang.Throwable implements java.io.Serializable {
    static final boolean $assertionsDisabled;
    public java.lang.Throwable();
    public java.lang.Throwable(java.lang.String);
    public java.lang.Throwable(java.lang.String, java.lang.Throwable);
    public java.lang.Throwable(java.lang.Throwable);
    protected java.lang.Throwable(java.lang.String, java.lang.Throwable, boolean, boolean);
}
```

AS

```
public java.lang.Exception(java.lang.Throwable);  
protected java.lang.Exception(java.lang.String, java.lang.Throwable, boolean, boolean);  
}
```

D:\Wrapper\classes>javap java.lang.Throwable

Compiled from "Throwable.java"

```
public class java.lang.Throwable implements java.io.Serializable {  
    static final boolean $assertionsDisabled;  
  
    public java.lang.Throwable();  
    public java.lang.Throwable(java.lang.String, java.lang.Throwable);  
    public java.lang.Throwable(java.lang.String, java.lang.Throwable, boolean, boolean);  
    protected java.lang.String getLocalizedMessage();  
    public java.lang.String getLocalizedMessage();  
    public synchronized java.lang.Throwable getCause();  
    public synchronized java.lang.Throwable initCause(java.lang.Throwable);  
    public java.lang.String toString();  
    public void printStackTrace();  
    public void printStackTrace(java.io.PrintWriter);  
    public void printStackTrace(java.io.PrintWriter);  
    public synchronized java.lang.Throwable fillInStackTrace();  
    public java.lang.StackTraceElement[] getStackTrace();  
    public void setStackTrace(java.lang.StackTraceElement[]);  
    native int getStackTraceDepth();  
    native java.lang.StackTraceElement getStackTraceElement(int);  
    public final synchronized void addSuppressed(java.lang.Throwable);  
    public final synchronized java.lang.Throwable[] getSuppressed();  
    static {};  
}
```

46

D:\Wrapper\classes>



```
top(logger : loggers)
```

```
logger.log();//Line n2
```

```
}
```

What will be the result of compiling and executing Test class?

- A. Line n1 causes compilation error
- B. Line n2 causes compilation error
- C. Exception is thrown at runtime
- D. No output is displayed but program terminates successfully.

```
valid
```

```
====
```

```
class MyThread1 implements Runnable{
```

```
}
```

```
class MyThread2 implements Runnable{
```

```
}
```

```
Runnable[] runnable = new Runnable[2];
```

```
runnable[0] = new MyThread1();
```

```
runnable[1] = new MyThread2();
```

Question>

Given code of Test.java file:

```
interface ILogger {  
    void log();  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        ILogger [] loggers = new ILogger[2]; //Line n1 ==> VM ==> loggers[0] = null; loggers[1] = null;  
        for(ILogger logger : loggers)  
            logger.log(); //Line n2 ==> VM ==> NullPointerException  
    }  
}
```

What will be the result of compiling and executing Test class?

- A. Line n1 causes compilation error
- B. Line n2 causes compilation error
- C. Exception is thrown at runtime
- D. No output is displayed but program terminates successfully.

Answer: C

1