TREE SET

SURYA Gold

Date Page

* tree set is ~~sub interface~~ class implementation class of the set interface
* it was introduced from jdk 1.2
* it is present java.util package
* the object inside tree set has to be comparable in nature that is only similar (or) homogeneous type object possible to be inserted into treeset
* the object inside tree set always follows natural sorting order this ascending order
* underlined data structure of treeset is binary tree

```
prog:- package set programs;
import java.util.Iterator;
import java.util.TreeSet;
public class demo
{
psvm (String [] args)
{
Treeset t= new TreeSet();
t.add ("dinagu");
t.add ("guldu");
t.add ("abc");
t.add ("srk");
Iterator itr = t.iterator();
while (itr.hasnext())
  {
   s.o.p (itr.next());
  }
}
```

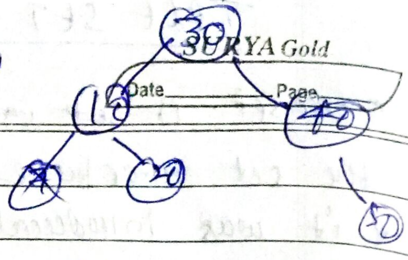
or

```
abc
dinagu
guldu
srk
```

internal complexity

```
Treeset t = new TreeSet();
t.add (30);
t.add (10); => 10.compareTo(30); -1 => left node
t.add (40); => 40.compareTo(30); +1 => right node
t.add (20); => 20.compareTo(30); -1 => LN
               20.compareTo(10); +1 => RN
```

t.add (50); 50.compareTo(30); +1 => RN
50.compareTo(40); +1 => RN



traversing order — compareTo()
left node — < => -1
parent node — > => +1
right node — = => 0

7 10 20 30 40 50

t.add() -> method internally calls compareTo() method.

* in Tree set the add() method it internally calling compareTo() method.

* compareTo () method returns 0(zero) if the value are same, return -1 if the value is less than, return +1 if the value is greater than

prg1:- package set programs;
public class Demo
{
psvm (String [] args)
{
// String s = "java";
// s.o.p (s.indexOf('v'));
// int x = 10;
// x. -> no method can be accessed because int is primitive data & it doesn't have any methods
Integer y = 35;
s.o.p (y. compareTo (35));
s.o.p (y. compareTo(50));
~~s.o.p (y. compareTo (120). co.~~
s.o.p (new Integer (120). compareTo(87));
}
}

Comparable
* comparable i
package
* comparable
the same
* comparable
to make
* it can
interface
* the return
prg1: package
public class
}
public
public
{
this
}
public
{
re
}
}
prg 2: pack
import java
public class
{
psvm (Str
{
Student
-//-
-//-
TreeS
(

Comparable interface

04/05/19

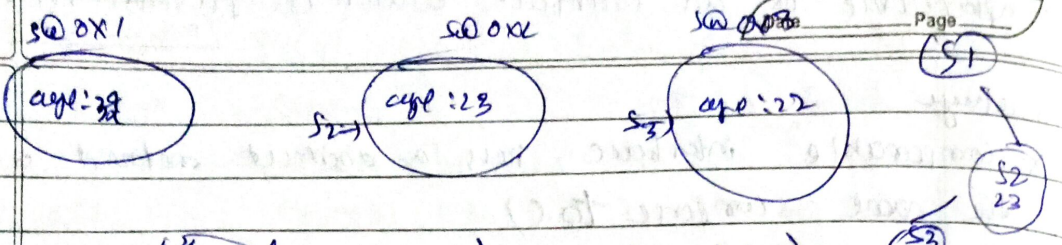
* comparable is an interface which is present in java.lang package
* comparable interface have a abstract method with the name compareTo()
* comparable interfaces have to be implemented in order to make the objects as comparable type
* it can also make use of generics with comparable interface
* return type of compareTo is integer.

prg1:-

```
package set program
public class student implements comparable<student>
{
    public int age;
    public student(int age)
    {
        this.age = age;
    }
    public int compareTo(student s)
    {
        return this.age - s.age;
    }
}
```

prg 2:-

```
package set program
import java.util.TreeSet;
public class demo
{
public static void main(String[] args)
{
    Student s1 = new Student(32);
    // s2 = ---(23);
    // s3 = ---(22);
    TreeSet<student> t = new TreeSet<student>();
    t.add(s1);
    t.add(s2);
    t.add(s3);
    for(student s:t)
    {
        s.o.println("age:"+s.age);
    }
}
}
```




```
int compareTo(Student[5])
{
return this.age - s.age;
s@0x2.age - s@0x1.age
   23 - 21 => 2
① s@0x3.age - s@0x1.age
   22 - 21 => +1
② s@0x3 compareTo(s@0x2)
   22 - 23 = -1
```

s1.compareTo(s1)
s@0x2.compareTo(s@0x1)
t.add(s3);
s@0x3.compareTo(s@0x1);

NOTE :- compareTo method is overridden in all wrapper classes as well as String class.

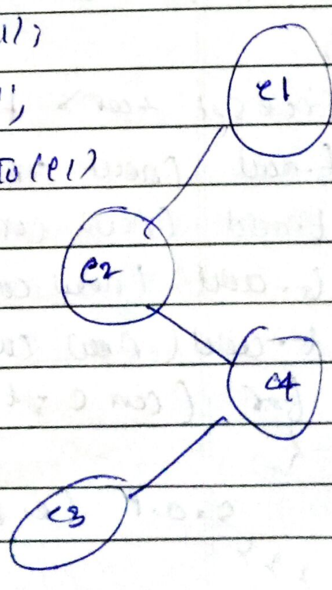
```
prg1:- package set programs;
public class employee implements comparable<Employee>
{
public int id;
public String name;
public double salary;
public Employee(int id, String name, double salary)
{
this.id = id;
this.name = name;
this.salary = salary
}
@override
public string toString()
{
return "Employee [id=" + id + ", name=" + name + ", "
+ "salary=" + salary + "]";
}
```

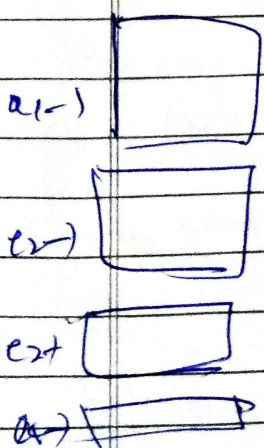
```
@Override
public int compareTo(Employee e)
{
  //return this.id - e.id;
  return this.name.compareTo(e.name);
  }
}
```

o/p
Employee[id=15, name=AB, salary=456.45]
" [id=10, " = Z, " = 123.56]
" [id=30 " = dinga, sal = 456.40]
" [id=20, name=guldu " = 123.12]

```
prg2:- package set programs;
import java.util.*;
public class SortEmployee
{
 psvm (String[] args)
 {
 Employee e1=new Employee(10, "Z", 123.56);
    "     e2 = "    "  (30,"dinga", 456.40);
    "     e3 = "    "  (20,"guldu", 123.12);
    "     e4 = "    "  (15, "AB", 456.45);
 TreeSet<Employee> t= new TreeSet<Employee>();
   t.add(e1);
   t.add(e2);
   t.add(e3);
   t.add(e4);
   Iterator i=t.iterator();
   while (i.hasNext())
    {
      s.o.p(i.next());
   }
  }
}
```

t.add(e2); -> e2.compareTo(e1);
"A".compareTo "Z";
t.add(e3) -> e3.compareTo(e1)
this -> e3
e -> e1
"c".compareTo("Z")
"c".compareTo("A")
this -> e3
e -> e2





```
prg!- package set program!
public class car implements comparable< car >
{
    double price;
    public car ( double price)
    { ~~double~~ this. price = price;
    }
    @ override
    public string to string
    { return 'price' + price;
```

```
prg1:- package set programs;
import java.util.Treeset;
public class car implements comparable<car>
{
// public double price;
public double price;
public car ( double price)
{
  this.price=price;
}
@ override
public int compareTo (car c)
{
  Double p= this.price;
//return this.price.compareTo(c.price);
  return p.compareTo(c.price);
}
psvm ( string [] args)
{
Treeset <car> t= new Treeset<car>();
t.add (new car(123.34));
t.add (new car(123.14));
t.add (new car(123.24));
t.add (new car(101.78));
for (car c : t)
{
   s.o.p (c.price);
}}}
```

O/P
101.78
123.14
123.24
123.34

natural properties → comparable
unnatural properties → comparator

{ we should written same class only

double p = (Double) this.price

int p = (int) this.price

compar…
into…
which is …
* compare…
object of…
should ha…
* comparato…
public…
* the baline…
seperatel…
steps
1] create A…
the compara…
2] override t…
comparing lo…
has to b…
3] create…
pass it a…
collection…
prg1:- pack…
public cla…
{
public int…
public co…
{
this. …
}
}
prg2:- packa…
public sa…
{
@over…
public…
{
re…
2}

Comparator interface

SURYA Gold
Date 8/5

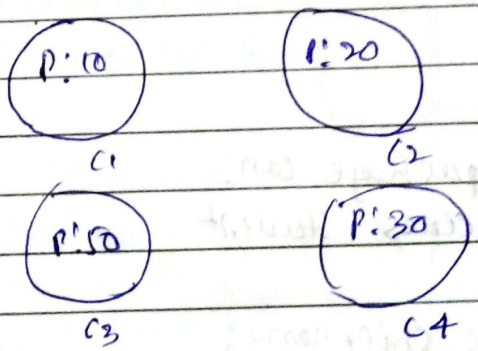
* Comparator is an interface which is present in java.util package.
* compare is use compare an related object that is object of different types can be compared. but they should have one common property btw them
* comparator interface have an abstract method that is public int compare(object o1, object o2);
* the business logic & sorting logic as to be written seperately

steps for using comparator interface

1) create A class and implement the comparator interface
2) over ride the compare method that is comparing logic are the sorting logic has to be written with in this one method
3) create an object of this specific class. class & pass it as parameter to the constructor call of the collection

it don't use generic write object in compare argument it use generic if automatic -ically down casted

pgm1:- package com;
public class car
{
public int price
public car (int price)
{
this.price = price;
}
}



pgm2:- package com; import java.util.comparator;
public class MyPriceComparator implements Comparator<Car>
{
@Override
public int compare(Car c1, Car c2) // second object o1, object o2
{
return c1.price - c2.price;
}
}

// if you want use
sop(c1+" "+c2+"\n");

SURYA Gold Date ____ Page ____

Prg3 :- package com;
import java.util.Treeset;
public class sort car
{
psvm (string []args)
{
car c1 = new car(10);
" c2 = " " (20);
" c3 = " " (50);
" c4 = " " (30);
Bypricecomparator p = new Bypricecomparator();
Treeset <cars> t = new Treeset <car> (p);
t.add(c1);
t.add(c2);
t.add(c3);
t.add(c4);
for (car c : t)
{
s.o.p (c.price)
}
}
}

t.add(c1);
t.add(c2);
c1.price - c2.price

[10] - [10]
20 - 10
30 - 10
30 - 20

(10) → (20) → (30) → (50)

compare (car c1, car c2)
entered object — already existing object

Prg1 :- package com;
public class Student
{
public string name;
public Student(string name)
{
this.name = name;
}
@ overrride
public string tostring()
{
return "name : " + name;
}
}

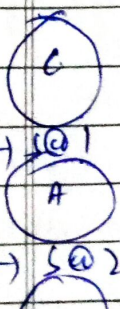
Prg2:- package com;
import java.util.Comparator;
public class sortbynamelogic implements comparator<Student>

{
@Override
public int compare(Student s1, Student s2)
{
return s1.name.compareTo(s2.name);
}
}

prg3:- package com;
import java.util.Treeset
public class sortstudents
{
psvm(---)
{
Student s1 = new student("c");
" s2 = " "("A");
" s3 = " "("B");
sortbynamelogic n = new sortbynamelogic();
Treeset<Student> t = new Treeset<Student>(n);
t.add(s1);
t.add(s2);
t.add(s3);
for(student stu: t)
{
l.o.p(stu);
}
}

public int compare(student x, student y)
{
return x.name.compareTo(y.name);
}

① Iteration → s2
x → s@2
y → s@1
s@2.name.compareTo(s@1.name)
"B".compareTo("C")
66 - 67 =



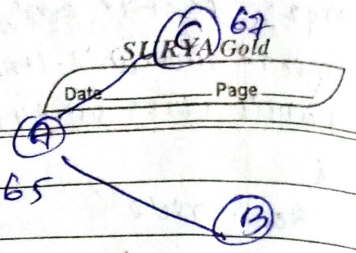
3rd Iterator → 53

x → s@3

y → s@1

x.compareTo("C")

66 − 67 ⇒ −1



prg1:- package com;

public class employee

{

int id;

String name;

Difference b/w	
Comparable	Comparator
1) Comparable is present in java.lang package	1) Comparator is present in java.util package
2) Comparable has an abstract method by the name compareTo()	2) Comparator has an abstract method by the name compare
3) mutual comparision (or) mutual sortening	3) un mutual comparision (or) un mutual sortening
4) Business logic and sortening logic is written in same class	4) Business logic & sortening logic is written in different class

prg1:- package com;

public class employee

{

int id;

String name;

double salary;

Employee(int id; String name, double salary)

{

this.id = id;

this.name = name;

this.salary = salary;

}

@ override

public String

{

return "empl

}

}

prg2:- package

import java.ut

public class Co

{

@override

public int compare

{

return

}

}

prg3:- package

import java.ut

public class S

{

@ override

public int

{

return

}

}

prg4:- packa

import java

public class

{

@ overrid

public int

{

Double

Double

return

}

}

(B)

rator
present in
uage
an object
compare
rision (or)
ating
sorting logic
different

```
public String toString()
{
return "Employee [id=" + id + ", name=" + name + ", " + "salary=" + salary + "]";
}
}
```

prg2:- package com;
import java.util.Comparator;
public class SortByID implements Comparator<Employee>
{
@Override
public int compare(Employee o1, Employee o2)
{
return o1.id - o2.id
}
}

prg3:- package com;
import java.util.Comparator;
public class SortByName implements Comparator<Employee>
{
@Override
public int compare(Employee o1, Employee o2)
{
return o1.name.compareTo(o2.name);
}
}

prg4:- package com;
import java.util.Comparator;
public class SortBySalary implements Comparator<Employee>
{
@Override
public int compare(Employee e1, Employee e2)
{
Double x = e1.salary;
Double y = e2.salary;
return x.compareTo(y);
}
}

```
Pgm5:- package com;
import java.util.Iterator;
import java.util.TreeSet;
public class main class
{
public static(String [] args)
{
Employee e1 = new Employee (101, "Dhoni", 1.23);
Employee e2 = new Employee (100, "Dingi", 1.45);
Employee e3 = new Employee (103, "Sobu", 1.12);
Employee e4 = new Employee (102, "Albek", 1.11);
Employee e5 = new Employee (901, "Euclid", 1.99);
// Sort By name name = new Sort By name();
// Sort By Salary sal = new Sort By Salary();
Sort By Id id = new Sort By Id();
Sort By Name name = new Sort Name();
TreeSet<Employee> s = new TreeSet<Employee> (id);
s.add(e1);
s.add(e2);
s.add(e3);
s.add(e4);
s.add(e5);
Iterator itr = s.iterator();
while (itr.hasNext())
{
s.o.p (itr.next());
}
}
}
```

SURYA Gold
Date Page

MAP

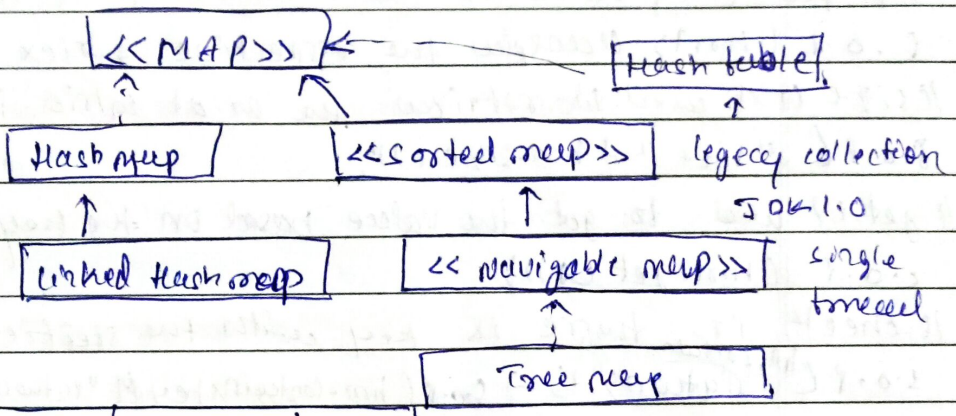
* maps is
hirachy
* maps
of he
* where
can be
* key & v
* in oth

Key
Value

* map
* map
* method
1) put()
2) get
3) re
4) Conte
5) Conta
6) isEm
7) clear
8) Si
9) Key

MAP'S

* maps is an interface present in the collection hirachy
* maps are used to represents the data in terms of key and value pair
* where keys can't be duplicated and values can be duplicated.
* key & value are together called as entry
* In otherwords maps is a collection of entry



Key	1	2	3
Value	"java"	"Python"	"sql"

* map is a interface present in java.util package
* map was introduced from JDK 1.2
* ~~method under map interface~~

1) put() method
2) get() method
3) remove method
4) contains key method
5) contains value method.
6) isEmpty method
7) clear method
8) size method
9) key set method

Prgl:- package mapprograms;
import java.util.HashMap;

public class Demo
{
public static void main(String[] args)
{
HashMap hm = new HashMap();
// put() is used to insert the key and value pair
hm.put("Java", 50);
hm.put(25.78, "Python");
hm.put(1, 90.84);
S.o.p(hm); //display the respective entries
//size() is used to retrieve the no of entries ie key pair
S.o.p("Size" + hm.size());
// get() used to get the value based on the key specified
S.o.p(hm.get(1));
// check if there is key with the specified data
S.o.p("Python")); -> S.o.p(hm.containsKey("Python"));
// check if there is value with the specified data.
S.o.p("Python")); -> S.o.p(hm.containsValue("Python"));
// check if there is value with the empty or not
S.o.p(hm.isEmpty());
// clear all the data
hm.clear();
S.o.p(hm.isEmpty());
}

O/P: {Java=50, 1=90.84, 25.78=Python}
Size 3
90.84
false
true
false
true

prgl:- package mapprograms;
import java.util.Set; import java.util.Iterator;
import java.util.LinkedHashMap;
public class Demo
{
public static void main(String[] args)
{
LinkedHashMap<Integer, String> l = new LinkedHashMap
<Integer, String>();

```
t.put(10, "java");
t.put(20, "jdbc");
t.put(30, "python");
t.put(40, "sql");
s.o.p();
Set<Integer> s = t.keySet();
Iterator<Integer> i = s.iterator();
while(i.hasNext())
{
    int key = i.next();
    s.o.p(key + ":" + t.get(key));
}
t.remove(20); // removes element based on the key
s.o.p();
}
}
```

10 = java, 20 = jdbc, 30 = python, 40 = sql

SURYA Gold Date Page

{10 = java, 30 = python, 40 = sql}

=> directly printing

```
prgl:- package mapprograms;
import java.util.Iterator;
import java.util.Set;
import java.util.TreeMap;
public class TreeMapDemo
{
public static void main(String[] args)
{
TreeMap<Integer, String> t = new TreeMap<Integer, String>();
t.put(100, "ninja");
t.put(20, "baba");
t.put(25, "spider");
Set<Integer> s = t.keySet();
Iterator<Integer> i = s.iterator();
while(i.hasNext())
{
    int key = i.next();
    String value = t.get(key);
    s.o.p(key + " " + value);
}
```

```
S.O.P("-----");
for (int obj : s)
{
    S.O.P(obj+" : "+tt.get(obj));
}
}
```

o/p
10 gulabi
20 jamun
100 Dinga

10 : gulabi
20 : jamun
100 : Dinga

under linked	Hash map	Linked Hashmap	Tree map
Data structure	Hash Table	Linked list & Hash table	Binary Tree
Insertion order	X	yes	X
Sorted order	X	X	yes
package	java.util	java.util	java.util
Null value allowed	yes	yes	NO
version (JDK)	1.2	1.4	1.2

prg1:- package mapprograms;
public class Employee implements Comparable<Employee>
{
int id;
public Employee(int id)
{
this.id = id;
}
@Override
public String toString()
{
return "id:"+id;
}
@Override
public int compareTo(Employeee)
{
Integer i = this.id;
//return this.id - e.id; return i.compareTo(e.id)
}
}

Prg2:- package map programs;

```
import java.util.*;
public class TreeMapDemo
{
public static void main (String[] args)
{
Employee e1 = new Employee (20);
Employee e2 = new Employee (10);
Employee e3 = new Employee (30);
TreeMap<Employee, String> t = new TreeMap<Employee, String>
                              ();
t.put (e1, "wipro");
t.put (e2, "Infosys");
t.put (e3, "Spiders");
Set<Employee> s = t.keySet();
Iterator<Employee> i = s.iterator();
while (i.hasNext())
{
Employee key = i.next();
String value = t.get(key);
S.O.P (key + " " + value);
}
for (Employee obj : s)
{
S.O.P (obj + ":" + t.get (obj));
}
}
}
```

o/p

Id: 10 Infosys
Id: 20 wipro
Id: 30 Spiders

Id = 10 : Infosys
Id = 20 : wipro
Id = 30 : Spiders