

TREE SET

SURYA Gold

Date _____

Page _____

to tree set 1) set interface class implementation class ob.

the set interface

* it was introduced from idk 1.2

* it is present in java.util package

* the object inside tree set has to be comparable
in nature that is only similar or homogeneous type
possible to be inserted into tree set

* the object inside tree set always follows natural
shortening order & association order

* underlying data structure of tree set is binary tree

progl:- package set programs;

obj1

abc

import java.util.Iterator;

clingo

import java.util.TreeSet;

guldu

public class demo

58k

{
 Scanner sc = new Scanner(System.in);
 TreeSet t = new TreeSet();

t.add("dragon");

t.add("guldu");

t.add("abc");

t.add("58k");

Iterator itr = t.iterator();

while (itr.hasNext())

{
 System.out.println(itr.next());

}

) internal working

TreeSet t = new TreeSet();

t.add(90);

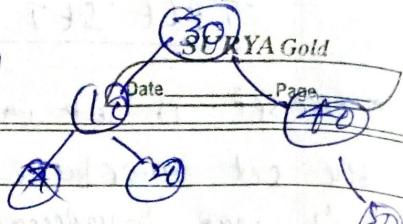
t.add(10); \Rightarrow compareTo(10); -1 \Rightarrow left node

t.add(40); \Rightarrow 4 < compareTo(10); +1 \Rightarrow right node

t.add(20); \Rightarrow 20 < compareTo(10); -1 \Rightarrow LN

20 < compareTo(10); +1 \Rightarrow RN

l. add(50); so compare(50); +1 => 2n
 S. compareTo(40); +1 => 3n



traversing order

compareTo()

left node

$< \Rightarrow -1$

parent node

$> \Rightarrow +1$

right node

$\hat{=} \Rightarrow 0$

b. add() -> method internally calls compare() method.

* in Tree set ^{the add()} method is internally calling compare

TO() 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

Q1 :- package setprogram;

public class Demo

{

Sum (String args)

{

// string s = "100d";

if(s.charAt(0) == 'U') {

int x = 10;

if x. -> no method can be accessed because int is primitive

Integer y = 35;

data + doesn't have any method

S.o.p(y. compareTo(35));

S.o.p(y. compareTo(50));

S.o.p(y. compareTo(120)).
 S.o.p(new Integer(120). compareTo(0));

S.o.p(new Integer(120). compareTo(0));

}

passing board

(1) +2 was lost at time

Compara

* Comparable

package

* Comparable

the name

* Comparable +

to make

* it can a

interface

a tree metas

proj1 package

public class

{

public

public

{

public

{

public

{

public

{

proj2 package

import java.util

public class

{

public

{

student

-11

-11-

Tree se

6

Comparable interface

at 05/09

SURYA Gold

Date _____

Page _____

* Comparable is an interface which is present in surya gold library

package

* Comparable interface has a abstract method with the name compareTo()

* Comparable interface has to be implemented in order to make the objects ~~has~~ comparable type
it can also make use of generic with comparable interface

return type of compareTo is integer.

e.g. some

prg1: package set program

public class student implements Comparable<student>

{

public int age;

public student (int age)

{

this.age = age;

}

public int compareTo (student)

{

return this.age - s.age;

t.add (s2);

t.add (s3);

for (student s: t)

{

s.o.s ("age: " + s.age);

}

2

prg2: package set programs

import sun.util.TreeSet;

public class Demo

{

student s1 = new student(32);

s1 s2 = —————— (23);

— (1) s3 = —————— (22);

TreeSet<student> t = new TreeSet<student>();

t.add(s1);

$s@0x1$
S1 → age: 28

$s@0x2$
S2 → age: 23

$s@0x3$
S3 → age: 22

(S1)

(S2
22)

+ (int compareTo(Student s))

t. add(s))

S2. compareTo(S1)

return stu.age - s.age;

~~s@0x2.age - s@0x1.age~~

23 - 21 = 2

s@0x2.compareTo(s@0x1)

t.add(s))

"s@0x3.compareTo(s@0x1)"

① s@0x3.age - s@0x1.age

22 - 21 = 1

② s@0x3.compareTo(s@0x2)

22 - 23 = -1

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

prg1:- package setprograms;

public class Employee implements Comparable<Employee>

{

public int id;

public String name;

public double salary;

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 + "]";

}

④ override
public int compareTo(Employee e)

{

 return this.id - e.id;

}
 return this.name.compareTo(e.name);
}

else

b

Employee id=15, name=B, salary=98.25

 if [id=10, "A" = 2, u = 123.56]

(S1) pay2 :- package set program:

import java.util.*;

public class SetEmployee

{

 sum (String args)

d

 Employee e1 = new Employee(10, "A", 123.56);

 e2 = "B" [30, "dinesh", 456.40];

 e3 = "C" [20, "guldu", 123.12];

 e4 = "D" [15, "AB", 98.25];

 TreeSet<Employee> t = new TreeSet<Employee>();

 t.add(e1);

 t.add(e2);

 t.add(e3);

 t.add(e4);

 Iterator p = t.iterator();

 while (p.hasNext())

{

 System.out.println(p.next());

e1
e2
e3
e4

t.add(e2); -> e2 compareTo(e1)

"A".compareTo("Z")

t.add(e3) -> e3.compareTo(e1)

 true -> e3

 e -> e1

e1

e1 ->

e2 ->

e3 ->

e4 ->

"C".compareTo("A")

"C".compareTo("Z")

 true -> e3

 e -> e2

e2

e3

e4

Ques:- package setprogram;
public class car implements Comparable<car>

SURYA Gold

Date _____

Page _____

{
double price;

public car (double price)

{ double ftr, ^{price} price = price;

}
@ override

public string tostring

{ return "price "+ price;

Ques:- package setprograms;

import java.util.TreeSet;

public class car implements Comparable<car>

01/01/2017

123, 141

123, 241

123, 341

{
// public double price;

public double price;

material properties → comparable

public car (double price)

variable properties → comparator

{
ftr, price = price;

}
@ package

public int compareTo (car c) {

{
if (we should omitted some class
only)

double p = (double) ftr -

Double p = this . price;

price

{ return ftr . price . compareTo (c . price); int p = (int) ftr .

return p . compareTo (c . price);

}

psuedo (main (args))

{

TreeSet <car> t = new TreeSet<car>();

t . add (new car (123, 341));

t . add (new car (123, 141));

t . add (new car (123, 241));

t . add (new car (101, 781));

for (car c : t)

{
System.out.println (c . tostring());

225

which is

compare

object of

should have

to comparable

. public

* the balance

separately

store

1) create a

file comparable

2) override

comparing logic

has to be

3) create a

pack it a

collection

Ques:- pack

public class

{

public int

public car

{

public .

2

3

Ques:- make

public

{

@over

public

{

2

Comparator interface

→ implements interface

SURYA Gold

Date 8/5/17

which is present in java.util package.

- * compare it use compare on matched object two if object of different types can be compared. but they should have one common property b/w them
- * comparator interface has an abstract method has
 - public int compare(Object o1, Object o2);
- * the business logic & sorting logic as to be written separately

→ store for using comparator interface

- 1) create a class and implement the comparator interface

- 2) override the compare method function

comparing logic are the sorting logic
has to be written with in this method

- 3) create an object of this specific 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 is abundant
- i really downcasted

this method

specify class & method

constructor call

prog1:- package com;

public class car

{

 public int price

 public car (int price)

{

 this.price = price;

}

}

P:10

C1

P:20

C2

P:50

C3

(P:30)

C4

prog2:- package com; import java.util.Comparator;

public sort car class by price comparator implement Comparator

{

 @Override

 if (Object1 < Object2) {

 return -1;

 public int compare(car c1, car c2)

{

 return c1.price - c2.price;

}

 String s = (c1 + " " + c2 + " ");

2>

Ques 3 :- package com;

```
import java.util.*;  
public class sort car
```

f. add(Car);

f. del(Car);

C1.prize <= C2.prize

SURYA Gold

Date _____ Page 20

Page 20

10) - 10)

20 - 10)

30 - 10)

30 - 20)

car c1 = new car(10);

11) c2 = new car(20); compare(car c1, car c2)

12) c3 = new car(50); entered object

13) c4 = new car(30); de-referenced existing

By price comparator p = new BypriceComparator();

Treeset & cars t = new TreeSet<car>(p);

f. add(c1);

f. add(c2);

f. add(c3);

f. add(c4);

for (car c : t)

{ System.out.println(" " + c.name + " " + c.prize); }

s.o.p (f.e.p. price)

{ } } } }

Ques 1 :- package com;

public class student

{

public String name;

public student(String name)

{ this.name = name;

}

@ overrided

public String toString()

return "Name: " + name;

Ques 2 :- package

import java.util.*;

public class

1. public void

2. public int

3. return

{ }

4. public

5. return

{ }

6. public

7. import

8. public

9. public

10. public

11. public

12. public

13. public

14. public

15. public

16. public

17. public

18. public

19. public

20. public

21. public

22. public

23. public

24. public

25. public

26. public

27. public

28. public

29. public

30. public

31. public

32. public

33. public

34. public

35. public

36. public

37. public

38. public

39. public

40. public

41. public

42. public

43. public

44. public

45. public

46. public

47. public

48. public

49. public

50. public

51. public

52. public

53. public

54. public

55. public

56. public

57. public

58. public

59. public

60. public

61. public

62. public

63. public

64. public

65. public

66. public

67. public

68. public

69. public

70. public

71. public

72. public

73. public

74. public

75. public

76. public

77. public

78. public

79. public

80. public

81. public

82. public

83. public

84. public

85. public

86. public

87. public

88. public

89. public

90. public

91. public

92. public

93. public

94. public

95. public

96. public

97. public

98. public

99. public

100. public

101. public

102. public

103. public

104. public

105. public

106. public

107. public

108. public

109. public

110. public

111. public

112. public

113. public

114. public

115. public

116. public

117. public

118. public

119. public

120. public

121. public

122. public

123. public

124. public

125. public

126. public

127. public

128. public

129. public

130. public

131. public

132. public

133. public

134. public

135. public

136. public

137. public

138. public

139. public

140. public

141. public

142. public

143. public

144. public

145. public

146. public

147. public

148. public

149. public

150. public

151. public

152. public

153. public

154. public

155. public

156. public

157. public

158. public

159. public

160. public

161. public

162. public

163. public

164. public

165. public

166. public

167. public

168. public

169. public

170. public

171. public

172. public

173. public

174. public

175. public

176. public

177. public

178. public

179. public

180. public

181. public

182. public

183. public

184. public

185. public

186. public

187. public

188. public

189. public

190. public

191. public

192. public

193. public

194. public

195. public

196. public

197. public

198. public

199. public

200. public

201. public

202. public

203. public

204. public

205. public

206. public

207. public

208. public

209. public

210. public

211. public

212. public

213. public

214. public

215. public

216. public

217. public

218. public

219. public

220. public

221. public

222. public

223. public

224. public

225. public

226. public

227. public

228. public

229. public

230. public

231. public

232. public

233. public

234. public

235. public

236. public

237. public

238. public

<

3rd iteration \rightarrow S3

X \rightarrow S@3

Y \rightarrow S@1

"B" - comparable ("C")

66 - 67 \Rightarrow -1

SURYA Gold 67

Date

Page

65

③

prog1:- package com;

public class employee

{
int id;
}

String name;

Difference b/w

comparable

i) comparable is present in

java.util package

ii) comparable has an abstract method by the same compareTo()

iii) mutual comparison ①

mutual sorting

iv) business logic and sorting logic

logic is written in same

class

comparator

i) comparator is present in

java.util package

ii) comparator has an abstract method by the same compare

iii) unmutual comparison ②

unmutual sorting

iv) business logic & sorting logic

is written in different

class

②

prog2:- package com;

public class employee

{

int id;

String name;

double salary;

Employee (int id; String name, double salary)

{

fun1. id = id;

fun1. name = name;

fun1. salary = salary;

}

② override

public String
{
return "empl";
}

}

prog2:- package
import java.util.

public class com
{

override

public int compare
{

return 0;

}

prog3:- package
import java.util.

public class com
{

② override

public int compare
{

return 0;

}

prog4:- package
import java.util.

public class com
{

② override

public int compare
{

Double

Double

return

}

Double

return

}

public String toString()

```
{  
    return "Employee [id=" + id + ", name=" + name + ", salary=" + salary + "]";  
}
```

(B) 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);
```

```
}
```

```
}
```

Program - Package com;
Import java.util.Iterator;
Import java.util.TreeSet;

SURYA Gold

Date _____ Page _____

public class main class

{

public static (String args)

{

Employee e1 = new Employee (101, "Dhruv", 1.13);

Employee e2 = new Employee (102, "Dhruv", 1.15);

Employee e3 = new Employee (103, "Soham", 1.12);

Employee e4 = new Employee (104, "Abhishek", 1.11);

Employee e5 = new Employee (105, "Eckhard", 1.09);

// Sort By Name name = new TreeSet<Employee>;

// Sort By Salary salary = new TreeSet<Employee>();

Sort By Id id = new TreeSet<Employee>();

sortByName.name = new TreeSet<Employee>();

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());

}

}

MAP

* Maps in
hierarchy

* Maps

of keys

* where
can be

* key & va

* In other

key
value

* Map

* Map

or another

1) put()

2) get()

3) remove

4) contains

5) containsKey

6) isEmpty

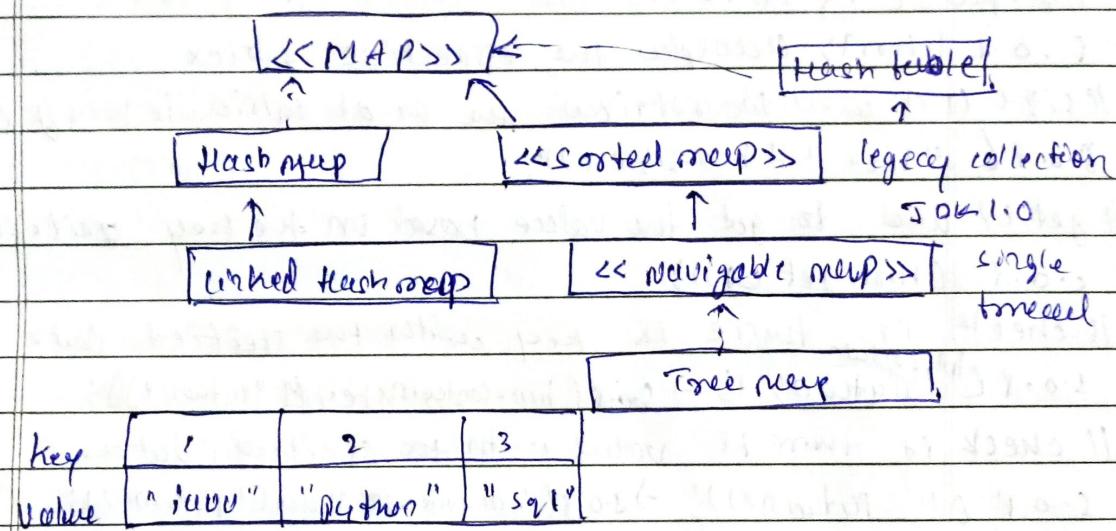
7) clear

8) size

9) keySet

MAP'S

- * maps is an interface present in the collection hierarchy
- * maps are used to represent 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 other words maps is a collection of entry



- * Map is an interface present in `java.util` package
- * Map was introduced from Jdk 1.2
- * method whaler map interface
 - 1) put() method
 - 2) get() method
 - 3) remove method
 - 4) containsKey method
 - 5) containsValue method
 - 6) isEmpty method
 - 7) clear method
 - 8) size method
 - 9) keySet method

Prgrm:- package mapprograms;
import java.util.*;

or } Java > 50, & = 90 to 125 & = returning

SURYA Gold

site 3

Date _____ Page _____

90.89

public class Demo

false

true

false

true

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, 90.89);

s.o.p(hm); // display the respective entries

// size() is used to retrieve the no of entries i.e k + v pairs

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("contains " + hm.containsKey("Python")); → s.o.p(hm.containsKey("Python"));

// check if there is value with the specified data.

s.o.p(" " + hm.containsValue("Python")); → s.o.p(hm.containsValue("Python"));

// check if there is data with the empty or not

s.o.p(hm.isEmpty());

// clear all the data

hm.clear();

s.o.p(hm.isEmpty());

}

Prgrm:- package mapprograms;

import java.util.*; import java.util.Set; import java.util.Iterator;

import java.util.hashmap; import java.util.util.CollableHashMap

public class Demo

{

return (String[]) args)

{

enhancedhashmap<integer, string> hm = new Enhancedhashmap<

integer, string>();

l. put(10, "j

l. put(120, "g

l. put(30, "p

l. put(40, "s

s.o.p("i

set <integer

Iterator<i

while (i.

put he

s.o.p("

i

l. remove

s.o.p("

l

prgrm:-

package

import

import

import

public

{

pic von l sru

l

TreeMap

b. put(10

e. put(2

e. put(2

set <int

Iterator

while (

int l

String

l

```
l.put(10, "java");
l.put(20, "dbc");
```

10 = java, 20 = dbc, 30 = python, 40 = sql
10 : java
20 : dbc
30 : python
SURYA Gold
Date _____ Page _____

```
l.get(30, "python");
l.get(40, "sql");
```

40 : sql
{ 10 = java, 20 = dbc, 30 = python, 40 = sql }

```
s.o.p(11);
set < Integer > s = l.keySet();
```

```
Iterator < Integer > i = s.iterator();
while (i.hasNext())
```

```
{
    int key = i.next(); // directly printing
    s.o.p(key + ":" + l.get(key));
}
```

by keys
1. remove(20); removes element based on the key
s.o.p();

iterated
2
1

site
1. progr:- package myprograms;

```
import java.util.Iterator;
```

```
import java.util.set;
```

```
import java.util.TreeMap;
```

```
public class TreeMapDemo
```

```
{
```

```
public void main(String args)
```

```
{
```

TreeMap < Integer, String > t = new TreeMap< Integer, String >();

```
t.put(10, "java");
```

```
t.put(20, "dbc");
```

```
t.put(25, "sql");
```

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)

o/p 10 quidu

25 jnider

100 Drgca

SURYA Gold

Date _____ Page _____

{

S.O.P ("obj" : "ff.get(obj)));

10 : quidu

20 : jnider

100 : Drgca

}
1>

| underlined | Hash map | Linked Hashmap | Tree map |
|--------------------|------------|--------------------------------|-------------|
| Order structure | Hash Table | Connected 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 (3D) | 1.2 | 1.4 | 1.2 |

prg1:- package newprograms;

public class Employee implements Comparable<Employee>

{

int id;

public Employee (int id)

{

this.id = id;

}

@Override

public String toString()

{

return " " + id;

}

@Override

public int compareTo(Employee e)

{

Integer i = this.id;

return this.id - e.id; return i.compareTo(e.id)

}

}

Prgr2:- public

import java

public

{

public

{

Employee

Employee

Employee

Tree map

t. put /

t. get

t. put

set /

Itera

length

3

com

ord

c.

3

box

1

3

1

3

1

3

1

3

alp SURYA Gold
Date: 20-07-2020 Page: 1

```
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, "Spiracle");
        Set<Employee> s = t.keySet();
        Iterator<Employee> p = s.iterator();
        while (p.hasNext()) {
            Employee key = p.next();
            String value = t.get(key);
            System.out.println(key + " : " + value);
        }
    }
}
```