DAY 128
DSA

Hashmap and Heap Continued
comparable & comparator
Date → 2 Feb, 2022
Day → Wednesday

Date
Page No.

Comparable and Comparator

COMPARABLE

Priority Queue में हम केवल Integer type का data store कर सकते है by default because ley default में priority Queue Integers की priority compare कर सकते है।

In the last class, we created our own Priority Queue, we inserted the elements into it from an array. The elements we inserted were of Integer datatype.

अब अगर हमें class के Object store करनी है Priority Queue में

lets say → we have a class Student

```
static class Student
{
    String name;
    int ht;
    int wt;
}
```

इस class में 3 data members या variables होंगे।

This is the class student

अगर हमें Student class के object को store करना है PQ में then we need to implement comparator in the student class

हमें Student class के objects को पहले comparable बनाना होगा तभी PQ इन्हें compare कर पाए

PQ को अभी पता नहीं चलेगा save करते वक्त कि element (objects) को किस base पर upheapify / downheapify करना है।

```
static class Student
{
    String name;
    int ht;
    int wt;

    Student (String name, int ht, int wt)
    {
        this.name = name;
        this.ht = ht;
        this.wt = wt;
    }

    public String toString()
    {
        return this.name + "-> ht [" + this.ht
                + "], wt [" + this.wt
                + "]";
    }
}

public static void main (String[] args)
{
    Student s = new Student ("A", 179, 80);

    System.out.println (s);
}
```

This is the student class.

println() function अपने अंदर toString() function को call करता है, और S class का toString() function को preference देता है।

Output → A → ht [179], wt [80]

अब हमारे पास 5 students का data है।
हमें students को height के basis पर sort करना होगा
और height के basis पर sorted students के data
को print करना होगा।

(1) हम Priority Queue में Student के Objects को डाल सकते हैं
और फिर remove करेंगे तो हमें sorted order में elements
मिलेंगे

But Priority Queue में add(), remove() functions
use करते upheapify(), downheapify() functions को

upheapify, Downheapify Comparision पर depend करते हैं

Integer

अगर Integer है
हमारा Data तो
>, < (greater, less)
के signs से हम parent और
child के data में comparision कर सकते
हैं upheapify(), downheapify()
functions में

Objects of a Class

अगर हम किसी Class
के Objects को PQ में
डालना चाहते हैं तो उस
Class में Comparable
implement करना होगा

(2) Comparable

↳ Comparable एक Interface है।

→ Interface basically assists comparision

a) Data का comparision करने में मदद मिलती है Interface से।

b) Interface एक contract of functions होता है।

c) जो class Interface (comparator) implement करती है, मतलब वो class function की body provide करती है।

(3)

→ Priority Queue को ऐसा data चाहिए जो कि comparable हो (Priority Queue class को comparision चाहिए)

→ अगर कोई class (Student) comparable implement करती है तो उसे compare करने वाले function की body provide करनी पड़ती है Student class को Compare करने वाला function implement करके देना पड़ेगा।

Student class को comparision देना पड़ेगा।

} Plugable

तो ये Plugable

अगर Priority Queue Class को comparision चाहिए और student class को comparision देना पड़ेगा।

Comparable and Comparator are Interface

Interface is just like a class. The difference between an Interface and class is that the Interface will not have the body of the function inside it. It will have only the signature of function. (signature means function name, parameter & return type).

An Interface can also be called as a contract for a class. This is because when a class has to implement an interface, it has to provide all the functions (bodies) with the same signature as that in the interface to it. It is just like, to use an interface, class honors its contract.

Now, if we have the data in student class

↳ Name → String datatype
Height → int datatype
Weight → int datatype
Marks → int datatype

अगर हमे सभी Student के data को print करना है

→ Sorted on basis of Height.
→ Sorted on basis of Weight.
→ Sorted on basis of Marks
→ Sorted on basis of Name.

} We can sort the data on the basis of any of these

SORTING THE DATA ON BASIS OF STUDENT HEIGHT, ON BASIS OF Weight, ON BASIS OF Marks,

Date / /
Page No.

(1) If we are using comparable interface

↳ comparable एक Interface होता है
(we don't need to pass 2 objects in comparable)
We pass only 1 object (other student)
in case of comparable
एक ही object pass होता है
this is the first object

Comparable is implemented in same class of objects (Student)

```
int compareTo (Student o)
{

}
```

Comparable has only 1 function called. compareTo.

This is the function to compare 2 values than why did we pass only 1 parameter

Actually there are 2 parameter,
one object is this → जिस object पर से function call हुआ है।
other object is passed as parameter

(2) To compare on basis of weight/height/marks we will use the Comparator interface

(1) ↳ Comparable interface को हम किसी और class में implement करते हैं।
We can't implement Comparator Interface in Student class itself

(2) We must notice that Comparable has only 1 parameter because the first object is the "this" object. Since, the comparator is in the different class and not in the same class., it will have both the parameters as it cannot be called by an object of student class.

If we do not pass any Comparator to the priority queue, then it will work according to the Comparable Interface.

The comparator Interface has a function called compare (Student 1, Student 2)

2 Parameters are passed here because comparator interface is implemented on different class

③ If we want to compare on basis of name, we can compare the Students on basis of priority of the name

↳ Name is of String datatype

String class
ने already comparable implement कर रखा है

Comparable में 1 String ही as a parameter pass करती है।
↳ other String

दोनो String (this, other) के words के character में ASCII difference check किया जाता है Comparable में

a b c d < a b e d

→ ये होगा compareTo function

```
public int compareTo(Student other)
{
    return this.name.compareTo(other.name);
}
```

name के base पर comparision होगा अब

Comparable is the Interface और compareTo() name के base पर दोनो objects को compare कर रहा है।

COMPARABLE AND COMPARATOR IMPLEMENTED <code>

Here we have used in built Priority Queue

Date ____ Page No. ____

```java
import java.util.*;
public class Main
{
    static class Student implements Comparable<Student>
    {
        String name;
        int height;
        int weight;
        int marks;

        Student(String name, int height, int weight, int marks)
        {
            this.name = name;
            this.height = height;
            this.weight = weight;
            this.marks = marks;
        }

        public int compareTo(Student other)
        {
            return this.name.compareTo(other.name);
        }

        public String toString()
        { return (this.name + "->[" + this.height + "],[" + this.weight + "],[" + this.marks + "]");
        }
```

Comparable Interface is implemented on the class of object (Students are the object).

ये Student class का constructor है।

Comparable function अगर किसी class पर implement करना है तो compareTo function का signature लिखना है।

Comparable Interface में इस function का signature होता है।

name के base पर priority दी गई है; String class में compareTo function already implemented है।

println() function toString() function को call करता है।

```
static class StudentHeightComparator implements
    Comparator <Student>
{   public int compare (Student S1, Student S2)
    {   return S1.height - S2.height ;
    }
}
```

Date

किसी class पर उस Comparator implement किया है

compare function का signature लिखना पड़ेगा उस class में

We are comparing on basis of height of Student object in StudentHeightComparator class.

```
static class StudentWeightComparator implements
Comparator <Student>
{   public int compare (Student S1, Student S2)
    {   return S1.weight - S2.weight ;
    }
}
```

किसी class पर आधार comparator apply किया है तो compare function का signature लिखना पड़ेगा.

comparing on basis of weight

```
static class StudentMarksComparator implements
Comparator <Student>
{   public int compare (Student S1, Student S2)
    {   return S1.marks - S2.marks;
    }
}
```

comparing on basis of marks.

→ Comparator दोनों दूसरी class में apply होते हैं; Comparator के compare() function में दोनों objects को pass किया जाता है.

```
public static void main (String[] args)
{
    Student [] students = new Student[5];
```
इसने students नाम का array of Student datatype बनाया

```
    students [0] = new Student ("Amit", 180, 75, 90);
    students [1] = new Student ("Sumeet", 150, 85, 33);
    students [2] = new Student ("Neha", 185, 72, 99);
    students [3] = new Student ("Kunal", 165, 65, 75);
    students [4] = new Student ("Aryan", 177, 55, 88);
```
Name Height Weight Marks

```
    PriorityQueue <Student> pq = new PriorityQueue<>();
```
↳ इस PriorityQueue में name के basis पर priority लगती।
No comparator is passed, it means comparable is the interface by default.

```
    PriorityQueue <Student> pqHeight = new PriorityQueue <>
        (new StudentHeightComparator());
```
इस PQ के object को Student Height comparator का object भी बनाया है।

StudentHeightComparator pass किया है as the Comparator here in the pqHeight

```
    PriorityQueue <Student> pqWeight = new PriorityQueue
        <>(new StudentWeightComparator());
```
This is passed as comparator तो अब bydefault interface(comparable) के interface के according priority नहीं दी जाएगी।

```
    PriorityQueue <Student> pqMarks = new PriorityQueue<>
        (new StudentMarksComparator());
```
Marks वाला Comparator को pass किया गया है।

```
for (Student student : students)
{ pqName.add(student);
  pqHeight.add(student);
  pqWeight.add(student);
  pqMarks.add(student);
}
```

सभी pq में student objects को add किया गया for each loop लगाकर

```
System.out.println("Sorting on basis of Name");
while(pqName.size()>0)
{ Student student = pqName.peek();
  pqName.remove();
  System.out.println(student);
}
```

इस sorting को name के basis पर किया गया है। Here Comparable was used as the Interface

Amit → [180], [75], [90]
Aryan → [177], [55], [80]
Kunal → [165], [65], [75]
Neha → [185], [72], [99]
Sumeet → [150], [85], [33]

System.out.println("Sorting on basis of height \n");

```
while (pqHeight.size()>0)
{
Student student= pqHeight.peek();
pqHeight.remove();
System.out.println(student);
}
```

Sorting on basis of Height

Sumeet → [150], [74], [334]
Kunal → [165], [79], [77]
Aryan → [177], [66], [88]
Amit → [180], [75], [566]
Neha → [185], [77], [888]

```
System.out.println("Sorting on basis of Height \n");
while (pqWeight.size()>0)
{
Student student= pqWeight.peek();
pqWeight.remove();
System.out.println(student);
}
```

Sorting on basis of Weight

Aryan → [177], [66] [88]
Sumeet → [150], [74], [334]
Amit → [180], [75], [566]
Neha → [185], [77], [888]
Kunal → [165], [79], [77]

```
System.out.println("Sorting on basis of Marks\n");
while (pqMarks.size()>0)
{
Student student = pqMarks.peek();
pqMarks.remove();
System.out.println(student);
}
```

Sorting on basis of Marks

Kunal → [165], [79], [77.]
Bryan → [177], [66], [88.]
Sumeet → [150], [74], [334]
Amit → [180], [75], [566]
Neha → [185], [77], [888]

Generic Priority Queue Using MyPriority Queue

```
import java.util.*;
public class MyGenericPriorityQueue
{
  public static class MyPriorityQueue<T>
  {
    ArrayList<T> data;
    Comparator cmptr;
  }
```

Here T can be any datatype or a class of datatypes

T can be any comparator here.

```
  public MyPriorityQueue()
  {
    data = new Arraylist<>();
    cmptr = null;
  }
```

→ This is the constructor of MyPriority Queue class when there is no comparator passed

```
  public MyPriorityQueue(Comparator cmptr)
  {
    data = new Arraylist<>();
    this.cmptr = cmptr;
  }
}
```

This is the constructor of MyPriority Queue class if the comparator is passed.

```
public void add (T val)
{ data.add (val);
  upheapify (data.size()-1);
}
```

The value we want to add can be of any datatype T.

Date / /
Page No.

```
void upheapify (int i)
{
  if (i==0)
  {
    return;
  }
  int pi=(i-1)/2;
  if (isSmaller (i,pi)==true)
  {
    swap (pi,i)
    upheapify (pi);
  }
}
```

The index i is passed here

यहाँ isSmaller() function call हुआ है, और i index वाला element अपने parent index से smaller है तो isSmaller() function true return करेगा, otherwise false.

If the condition is satisfied तो swap करो और recursive call लगाओ.

```
public T peek(){
  if (data.size()==0)
  {
    System.out.println("Underflow");
    return null;
  }
  return data.get(0);
}
```

→ The peek() function will return the value at first index of Arraylist

```
public T remove()                    // This function will return the value of T datatype
{ if (data.size()==0)
  { System.out.println("Underflow");
    return null;
  }
swap(0, data.size()-1)               // swap function is called here.

T val = data.remove(data.size()-1)
downHeapify(0);
return val;

}

void downheapify(int i)              // This is the downheapify function
{ int mini =0;                       // minimum index
  int lci = 2*i+1;                   // left child index.
  if (lci < data.size() && isSmaller(lci,mini)==true)
     { mini = lci;
     }
  int rci= 2*i+2;                    // right child index.
  if (rci<data.size() && isSmaller(rci,mini==true)
    { mini=rci;
    }
  if (mini!=i)
  {swap(i, mini);
   downheapify(mini);
  }
}
```

```
public int size()
{ return vdata.size(); }
```

Size of the ArrayList will be returned using the size() function.

```
void swap(int i, int j)
{
  T ith = vdata.get(i);
  T jth = vdata.get(j);
  vdata.set(i, jth);
  vdata.set(j, ith);
}
```

The isSmaller() function will check weather the value at ith index is smaller than the value at jth index.

```
boolean isSmaller(int i, int j)
{
  T ith = vdata.get(i);
  T jth = vdata.get(j);

  if (comptr != null)
  { if (comptr.compare(ith, jth) < 0)
    { return true; }
    else { return false; }
  }
  else { Comparable vith = (Comparable) ith;
         Comparable vjth = (Comparable) jth;
    if (vith.compareTo(vjth) < 0)
    { return true; }
    else { return false; }
  }
}
```

If there is comparator passed to the pq, then we will use the compare function of vgeeks.Comparator Interface

If there is no comparator passed, then we will use the compare To function to compare both the objects

comparable ith

We need to do explicit type casting to convert to Comparable

```
Static class Student implement Comparable<Student>
{ String name;
  int height;
  int weight;
  int marks;

  Student ( String name, int height, int weight,
            int marks)
  {
    this.name = name;
    this.height = height;
    this.weight = weight;
    this.marks = marks;
  }

  public int compareTo (Student other)
  { return this.name.compareTo (other.name);
  }

  public String toString()
  { return ( this.name + " -> [" +
             this.height + "] , [" +
             this.weight + "] , [" +
             this.marks + " ] "
           );

  }
}
```

↳ This is the Student class

This class implements Comparable.

This is the constructor of student class.

This is the signature of compareTo function

The println() function in System.out.println calls the toString() function behind the scene.

```
static class StudentHeightComparator implements
    Comparator<Student>
{
    public int compare(Student Student1,
                       Student Student2)
    {
        return Student1.height - Student2.height;
    }
}

static class StudentWeightComparator implements
    Comparator<Student>
{
    public int compare(Student Student1,
                       Student Student2)
    {
        return Student1.weight - Student2.weight;
    }
}

static class StudentMarksComparator implements
    Comparator<Student>
{
    public int compare(Student Student1,
                       Student Student2)
    {
        return Student1.marks - Student2.marks;
    }
}
```

→ This StudentHeightComparator class implements the comparator interface

→ This compared function sort the elements on basis of height

→ This class implement comparator interface in which comparision on basis of weight of Student

Comparision on basis of weight

Comparision on basis of marks

```
public static void main(String[] args)
{
    Student[] students = new Student[5];
    students[0] = new Student("Amit", 180, 75, 566);
    students[1] = new Student("Sumeet", 150, 74, 334);
    students[2] = new Student("Neha", 185, 77, 888);
    students[3] = new Student("Kunal", 165, 79, 77);
    students[4] = new Student("Aryan", 177, 66, 88);

    MyPriorityQueue<Student> pqName =
        new MyPriorityQueue<>();

    MyPriorityQueue<Student> pqReverseName =
        new MyPriorityQueue<>(Collections.reverseOrder());

    MyPriorityQueue<Student> pqHeight = new
        MyPriorityQueue(new StudentHeightComparator());
```

Date / / Page No.

This is the array of Student object

These are the 5 objects of Student class

priority queue name

Now the comparision of student will be done on basis of their name.

This PQ has no comparator, so the comparable interface will be the by default interface

priority queue reverse name

Collection.reverseOrder returns a comparator of the comparable of priority of reverse order.

So, the comparision will be done on basis of height

This comparator is passed as interface

```
MyPriorityQueue<Student> pqReverseHeight = new
        MyPriorityQueue<>(Collections.reverseOrder
                (new StudentHeightComparator()));
```

Comparision will be done on basis of Height and then priority will be reversed.

StudentHeightComparator is passed as the comparator here in the Collection.reverseOrder()

```
MyPriorityQueue<Student> pqHeight = new
        MyPriorityQueue<>(new StudentHeightComparator());
```

Comparision will be done on basis of weight.

```
MyPriorityQueue<Student> pqMarks = new
        MyPriorityQueue<>(new StudentMarksComparator());
```

Comparision will be done on basis of Marks

```
for(Student student : students)
{
    pqName.add(student);
    pqReverseName.add(student);
    pqHeight.add(student);
    pqWeight.add(student);
    pqMarks.add(student);

    pqReverseHeight.add(student);
}
```

students array pr for each loop ~~with~~ ~~for~~ & then we will add each of the elements in all the PQ.

→ System.out.println("Sorting on basis of Name\n");

while (pqName.size()>0)

→ This PQ has the elements sorted on basis of names

{

Student student = pqName.peek();

pqName.remove();

System.out.println(student);

}

↳ Output

Sorting on basis of Name.

Amit → [180], [75], [566]

Aryan → [177], [66], [88]

Kunal → [165], [79], [77].

Neha → [185], [77], [888]

Sumeet → [150], [74], [334]

→ System.out.println("Sorting on basis of Reverse Name \n");

while (pqReverseName.size()>0)

→ This PQ has elements sorted on basis of names but in Reverse order

{

Student student = pqReverseName.peek();

pqReverseName.remove();

System.out.println(student);

}

Output

Sorting on basis of ReverseName

Sumeet → [150], [74], [334]

Neha → [185], [77], [888]

Kunal → [165], [79], [77]

Aryan → [177], [66], [88]

Amit → [180], [75], [566]

```
System.out.println("Sorting on basis of Weight \n");
while(pqWeight.size()>0)
{
  Student student = pqWeight.peek();
  pqWeight.remove();
  System.out.println(student);
}
```

→ In this PQ, elements are sorted on the bases of their weight

Output ↓

Sorting on basis of Weight.
Bryan → [77], [66], [88]
Sumeet → [150], [74], [334]
Amit → [180], [75], [556]
Neha → [185], [77], [888]
Kunal → [165], [79], [77]

```
System.out.println("Sorting on bases of Marks \n");
while(pqMarks.size()>0)
{
  Student student = pqMarks.peek();
  pqMarks.remove();
  System.out.println(student);
}
```

Output ↓

Sorting on basis of Marks.
Kunal → [165], [79], [77]
Bryan → [177], [66], [89]
Sumeet → [150], [74], [334]
Amit → [180], [75], [566]
Neha → [185], [77], [888]

```
System.out.println("Sorting on basis of Height \n");
while(pqHeight.size()>0)
{
  Student student = pqHeight.peek();
  pqHeight.remove();
  System.out.println(student);
}
```

→ In this pq, the elements are sorted on basis of their height.

Output

Sorting on basis on Height.

Sumeet → [150], [74], [334]
Kunal → [165], [79], [77]
Aryan → [177], [66], [88]
Amit → [180], [75], [566]
Neha → [185], [77], [888]

```
System.out.println("Sorting on basis of a Reverse Height Comparator \n");
while(pqReverseHeight.size()>0)
{
  Student student = pqReverseHeight.peek();
  pqReverseHeight.remove();
  System.out.println(student);
}
```

The elements are sorted on basis of height but in reverse order.

Output

Sorting on basis on Reverse Height

Neha → [185], [75], [888]
Amit → [180], [75], [566]
Aryan → [177], [66], [88]
Kunal → [165], [79], [77]
Sumeet → [150], [74], [334]