Diagram

Description automatically generated

**Priority Queue**

* provides the functionality of the [heap data structure](https://www.programiz.com/dsa/heap-data-structure).
* It implements the [Queue interface](https://www.programiz.com/java-programming/queue).
* elements are retrieved in sorted order.
* the head of the priority queue will be the smallest element.

the elements of a priority queue may not be sorted. However, elements are always retrieved in sorted order.

**Creating PriorityQueue**

|  |
| --- |
| import the java.util.PriorityQueue package.  Queue<Integer> numbers = new PriorityQueue<>(); |

**Operations on PriorityQueue**

1. **Adding Elements:**

|  |
| --- |
| Queue<Integer> pq = **new** PriorityQueue<>();  **for**(**int** i=0;i<3;i++)  {              pq.add(i);  } |

1. **Removing Elements:**

|  |
| --- |
| pq.remove(2);  System.out.println("After Remove - " + pq);    System.out.println("Poll Method - " + pq.poll());    System.out.println("Final PriorityQueue - " + pq); |

1. **Accessing the elements:**

|  |
| --- |
| String element = pq.peek();   System.out.println("Accessed Element: " + element); |

1. **Iterating the PriorityQueue:**

|  |
| --- |
| Iterator iterator = pq.iterator();    **while** (iterator.hasNext()) {     System.out.print(iterator.next() + " ");   } |

it uses a default natural ordering. In this case, it gave us the data back in the ascending order.

1. **Custom ordering using Comparator:**

Custom ordering is possible with the help of a **comparator.**

|  |
| --- |
| static class CustomIntegerComparator implements Comparator<Integer> {  @Override  public int compare(Integer o1, Integer o2) {  return o1 < o2 ? 1 : -1;  }  } |

to add this comparator to the priority queue.

|  |
| --- |
| **public** **class** Priority\_Queue\_Demo {  **public** **static** **void** main(String[] args)      {  PriorityQueue<Integer> PQ = new PriorityQueue<>(new CustomIntegerComparator());  PQ.add(11);  PQ.add(5);  PQ.add(-1);  PQ.add(12);  PQ.add(6);  System.out.println("Integers stored in reverse order of priority in a Priority Queue\n");  while (!PQ.isEmpty()) {  System.out.println(PQ.poll());  }  } |

output of the above program:

12

11

6

5

-1

1. **Custom ordering using Comparable interface:**

import java.util.Objects;

import java.util.PriorityQueue;

class Employee implements Comparable<Employee> {

private String name;

private double salary;

public Employee(String name, double salary) {

this.name = name;

this.salary = salary;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getSalary() {

return salary;

}

public void setSalary(double salary) {

this.salary = salary;

}

@Override

public boolean equals(Object e) {

if (this == e) return true;

if (e == null || getClass() != e.getClass()) return false;

Employee employee = (Employee) e;

return Double.compare(employee.salary, salary) == 0 &&

Objects.equals(name, employee.name);

}

@Override

public int hashCode() {

return Objects.hash(name, salary);

}

@Override

public String toString() {

return "Employee{" +

"name='" + name + '\'' +

", salary=" + salary +

'}';

}

// Compare two employee objects by their salary

@Override

public int compareTo(Employee employee) {

if(this.getSalary() > employee.getSalary()) {

return 1;

} else if (this.getSalary() < employee.getSalary()) {

return -1;

} else {

return 0;

}

}

}

public class PriorityQueueUserDefinedObjectExample {

public static void main(String[] args) {

// Create a PriorityQueue

PriorityQueue<Employee> employeePriorityQueue = new PriorityQueue<>();

// Add items to the Priority Queue

employeePriorityQueue.add(new Employee("Rajeev", 100000.00));

employeePriorityQueue.add(new Employee("Chris", 145000.00));

employeePriorityQueue.add(new Employee("Andrea", 115000.00));

employeePriorityQueue.add(new Employee("Jack", 167000.00));

while (!employeePriorityQueue.isEmpty()) {

System.out.println(employeePriorityQueue.remove());

}

}

}

**# Output**

Employee{name='Rajeev', salary=100000.0}

Employee{name='Andrea', salary=115000.0}

Employee{name='Chris', salary=145000.0}

Employee{name='Jack', salary=167000.0}

**Java Comparable and Comparator**

Text

Description automatically generated

Java Comparable

example of a Comparable interface that sorts the list elements on the basis of age.

**import** java.util.\*;

**import** java.io.\*;

**class** Student **implements** Comparable<Student>

{

**int** rollno;

String name;

**int** age;

Student(**int** rollno,String name,**int** age)

{

**this**.rollno=rollno;

**this**.name=name;

**this**.age=age;

}

**public** **int** compareTo(Student st)

{

**if**(age==st.age)

**return** 0;

**else** **if**(age>st.age)

**return** 1;

**else**

**return** -1;

}

}

//Creating a test class to sort the elements

**public** **class** TestSort3

{

**public** **static** **void** main(String args[])

{

ArrayList<Student> al=**new** ArrayList<Student>();

al.add(**new** Student(101,"Vijay",23));

al.add(**new** Student(106,"Ajay",27));

al.add(**new** Student(105,"Jai",21));

Collections.sort(al);

**for**(Student st:al)

{

System.out.println(st.rollno+" "+st.name+" "+st.age);

}

}

}

## Java Comparator

**Student.java**

**class** Student

{

**int** rollno;

String name;

**int** age;

Student(**int** rollno,String name,**int** age)

{

**this**.rollno=rollno;

**this**.name=name;

**this**.age=age;

}

}

**AgeComparator.java**

**import** java.util.\*;

**class** AgeComparator **implements** Comparator<Student>

{

**public** **int** compare(Student s1,Student s2)

{

**if**(s1.age==s2.age)

**return** 0;

**else** **if**(s1.age>s2.age)

**return** 1;

**else**

**return** -1;

}

}

**NameComparator.java**

**import** java.util.\*;

**class** NameComparator **implements** Comparator<Student>

{

**public** **int** compare(Student s1,Student s2)

{

**return** s1.name.compareTo(s2.name);

}

}

**TestComparator.java**

**import** java.util.\*;

**import** java.io.\*;

**class** TestComparator{

**public** **static** **void** main(String args[])

{

//Creating a list of students

ArrayList<Student> al=**new** ArrayList<Student>();

al.add(**new** Student(101,"Vijay",23));

al.add(**new** Student(106,"Ajay",27));

al.add(**new** Student(105,"Jai",21));

System.out.println("Sorting by Name");

//Using NameComparator to sort the elements

Collections.sort(al,**new** NameComparator());

//Traversing the elements of list

**for**(Student st: al){

System.out.println(st.rollno+" "+st.name+" "+st.age);

}

System.out.println("sorting by Age");

//Using AgeComparator to sort the elements

Collections.sort(al,**new** AgeComparator());

//Travering the list again

**for**(Student st: al){

System.out.println(st.rollno+" "+st.name+" "+st.age);

}

}

}

Output:

Sorting by Name

106 Ajay 27

105 Jai 21

101 Vijay 23

Sorting by Age

105 Jai 21

101 Vijay 23

106 Ajay 27