Lesson 05 Demo 04

Implementing PriorityQueue

Objective: To demonstrate the use of priority queue in Java for sorting and processing data with a first-in-first-out approach

Tools Required: Eclipse IDE

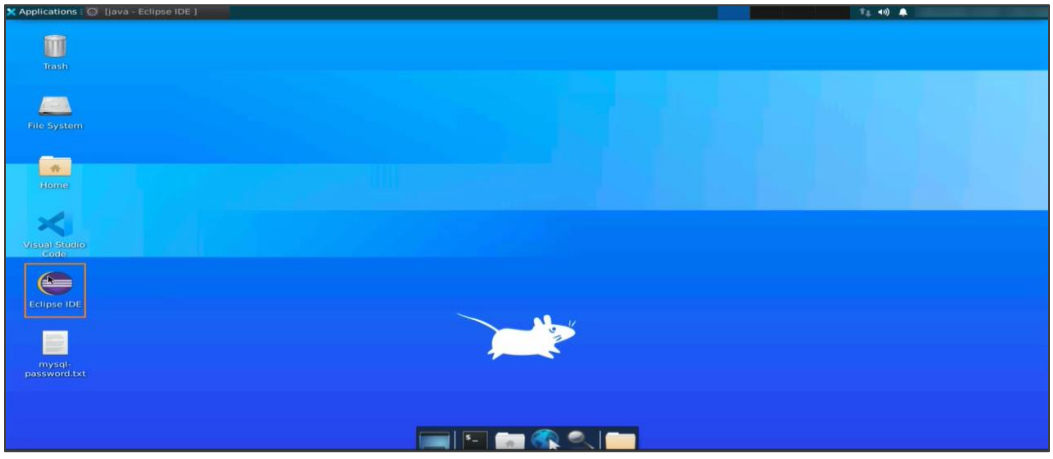
Prerequisites: None

Steps to be followed:

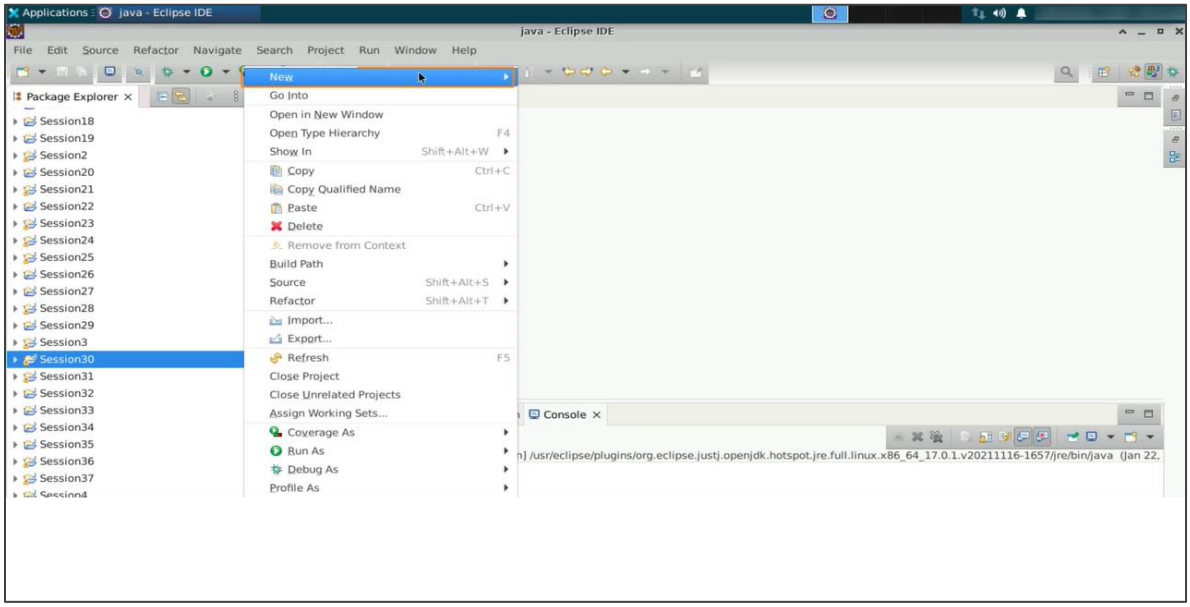
1. Create a new project
2. Add data to the queue using the add method
3. Implement the use of peek and poll methods
4. Implement the use of the variable size using the **queue.size()** method
5. Use the **queue.iterator()** function with example data
6. Implement queue prioritization

Step 1: Create a new project

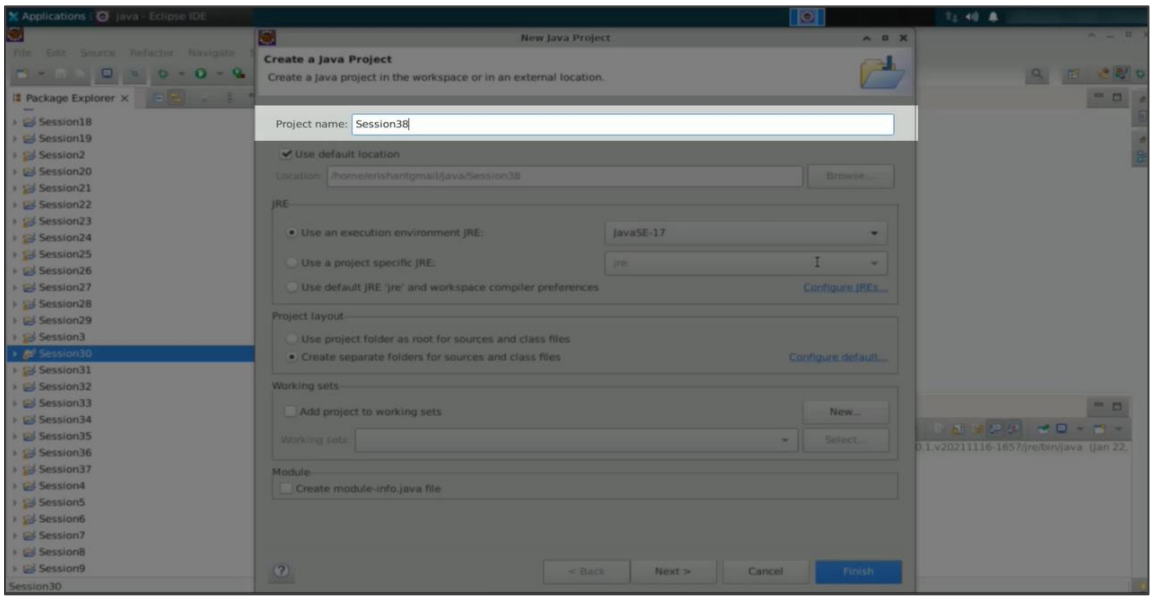
1.1 To create a queue, open the **Eclipse IDE**



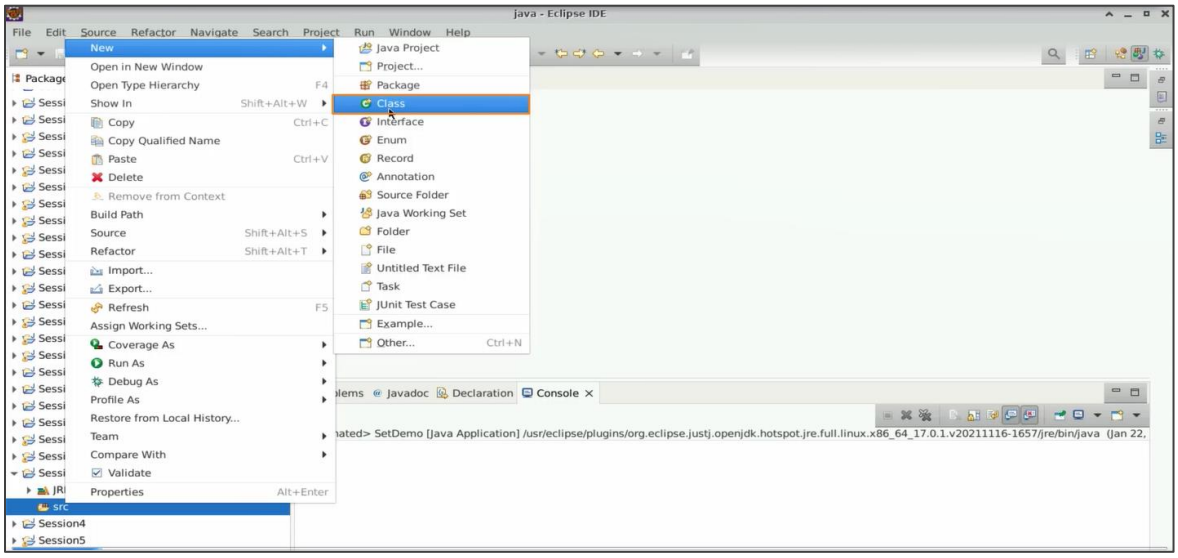
1.2 Create a new Java project

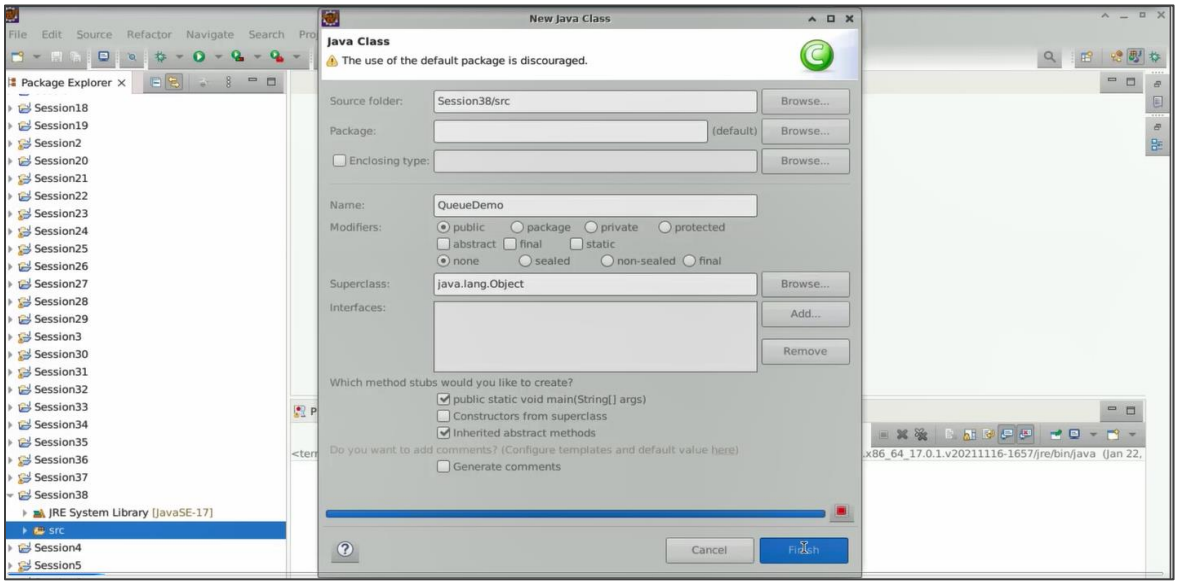


1.3 Name the new project as **Session38**



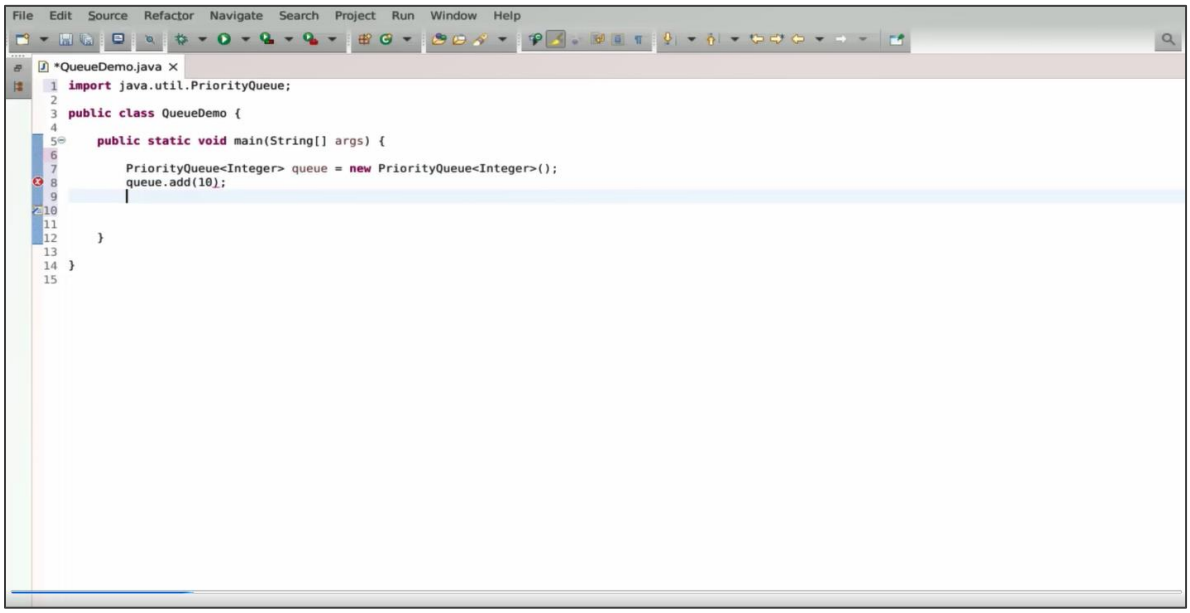
1.4 Right-click on **Session38** and create a new class called **QueueDemo** with the main method



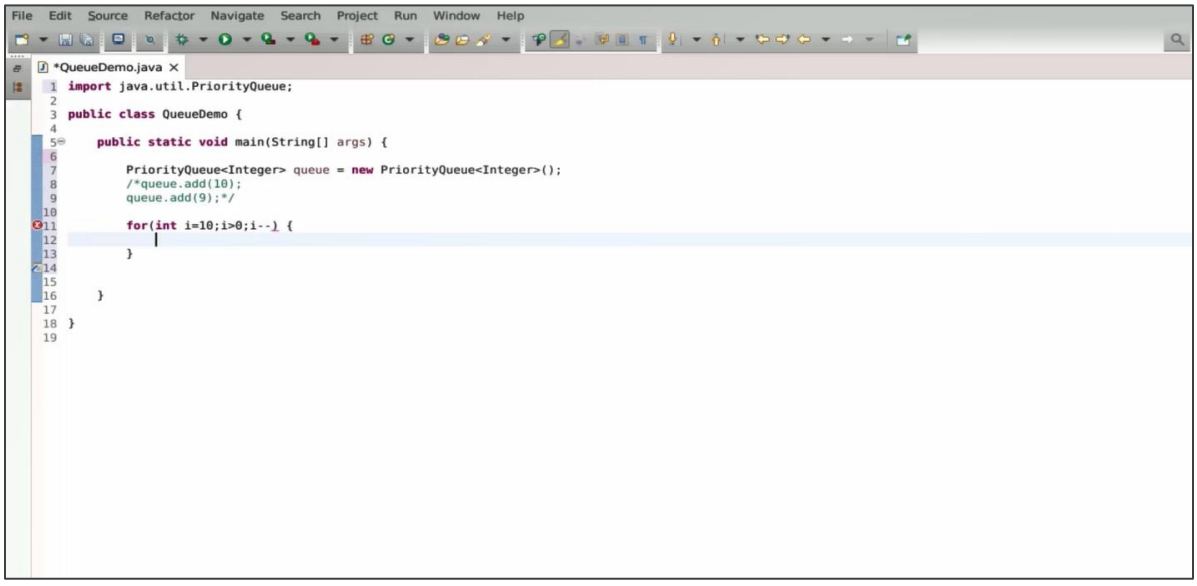


Step 2: Add data to the queue using the add method

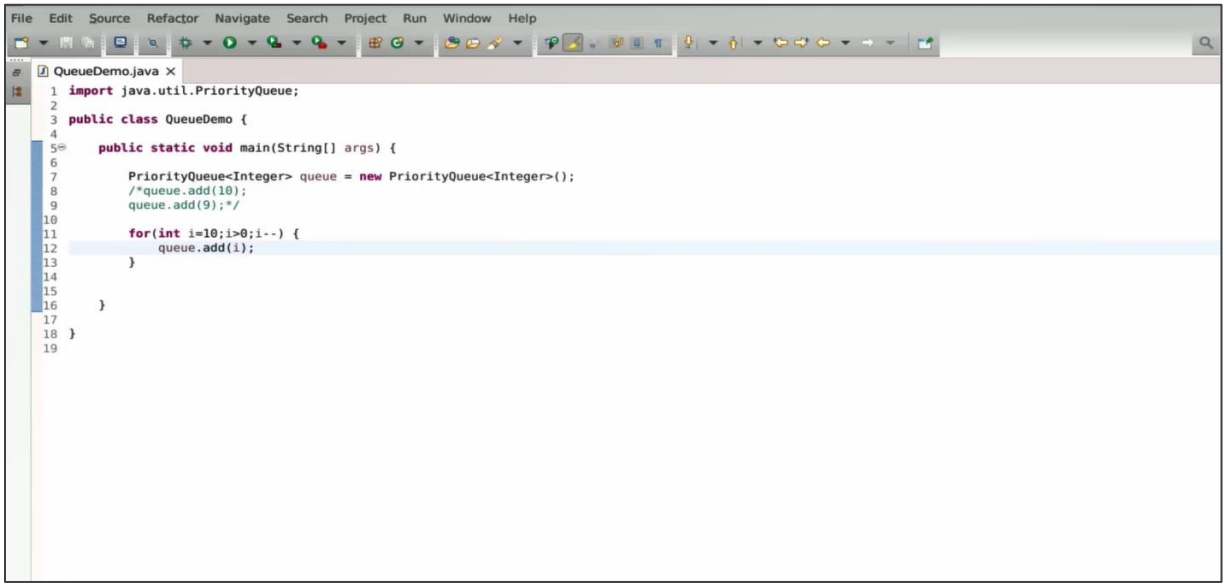
2.1 To add data to the queue, use the add method like lists or sets. Write: **queue.add(10);**



2.2 Add more data to the queue, such as **9**, or alternatively use a loop to add 10 more numbers



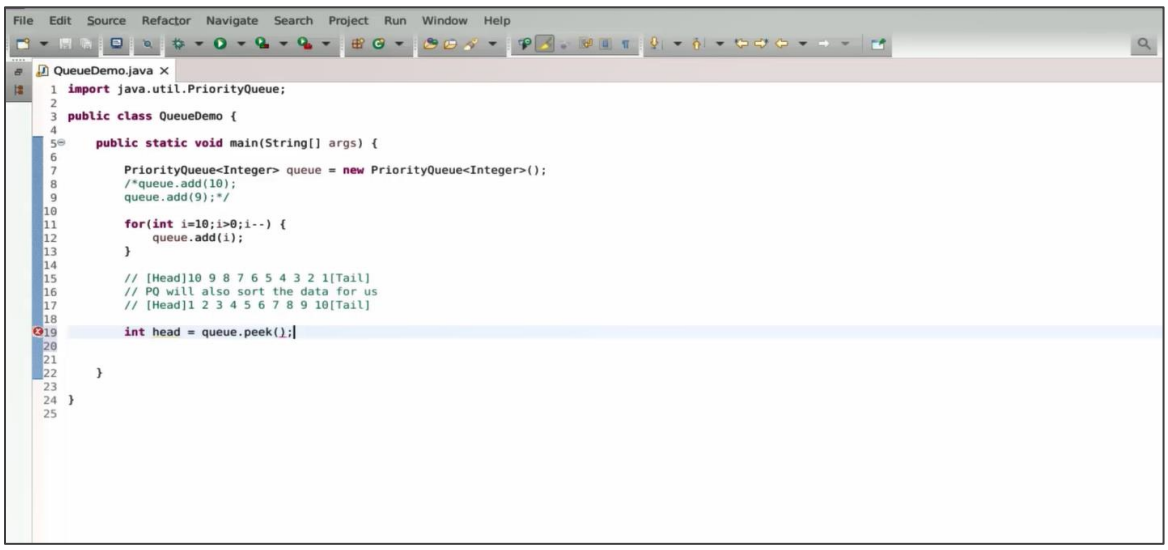
2.3 Use a reverse loop starting from 10 and decrementing **i**. Write: **queue.add(i);**



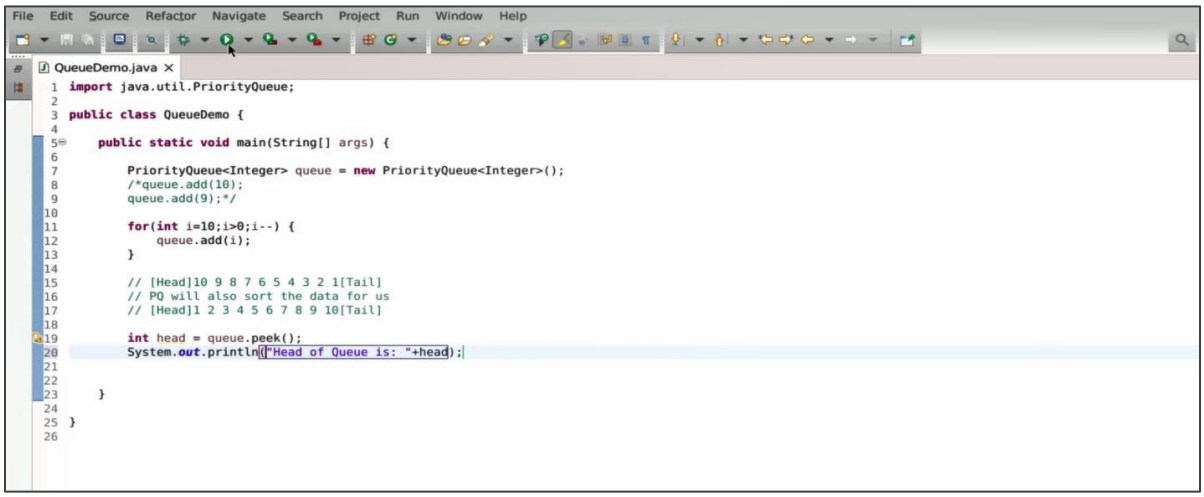
This creates a queue with elements from 10 to 1, where 10 is the head and 1 is the tail. The priority queue automatically sorts the data, so the queue will be arranged in ascending order. This means that the head of the queue will be 1, demonstrating the sorted nature of the priority queue.

Step 3: Implement the use of peek and poll

3.1 Write: **int head = queue.peek();**

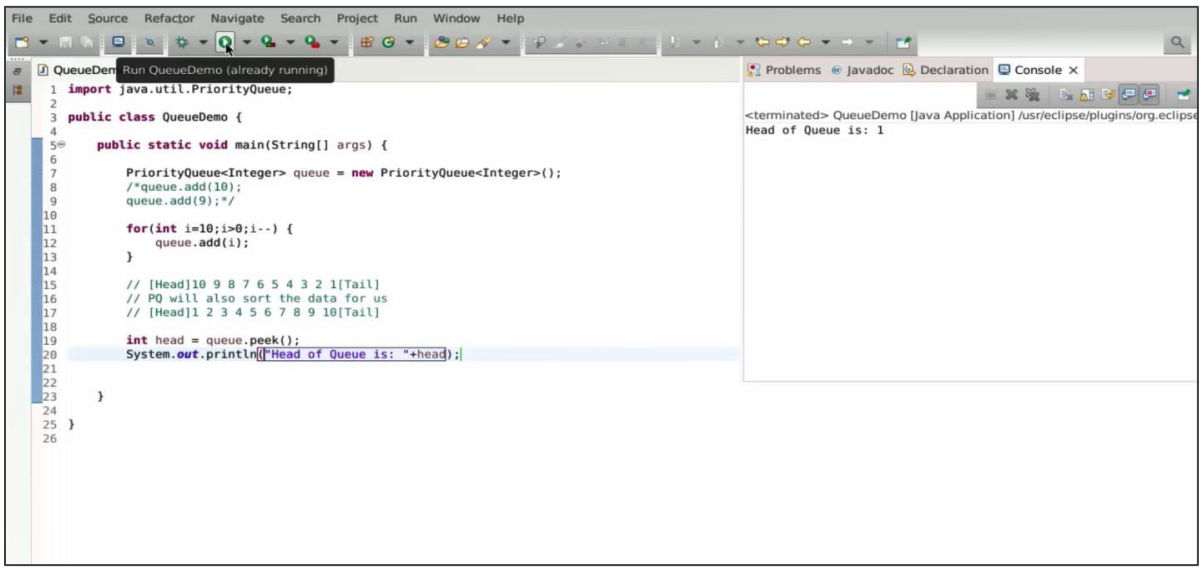


3.2 To retrieve the head of the queue, use the peek method. Write: "**Head of Queue is: "+ head**

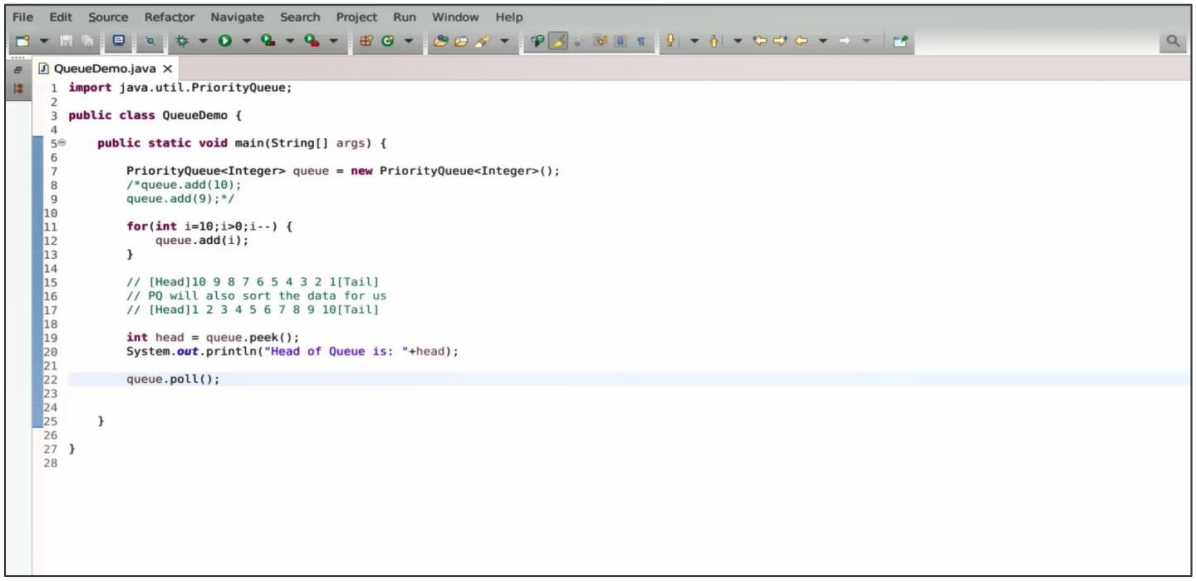


3.3 Output is shown below:

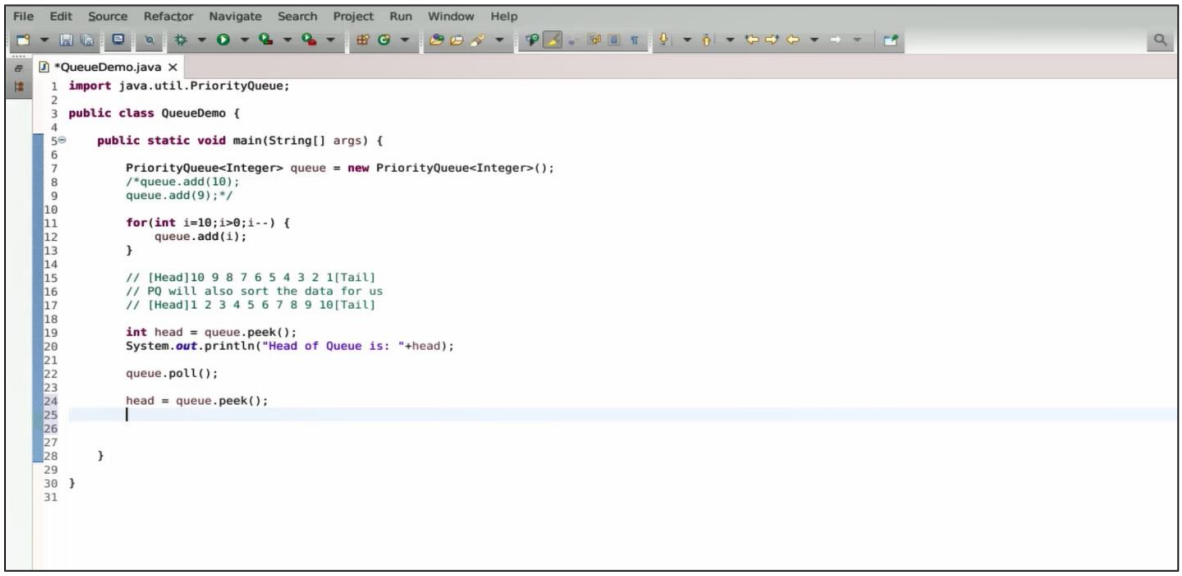
Head of the Queue is: 1



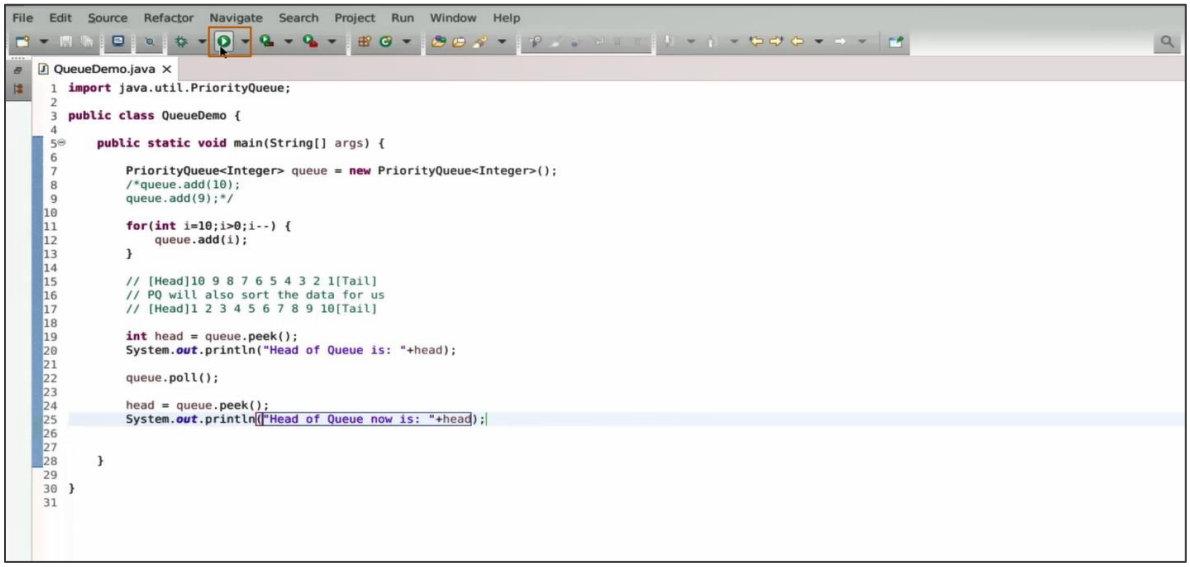
3.4 To remove the head of the queue, write: **queue.poll();**



3.5 Retrieve the new head by writing: **head = queue.peek();**

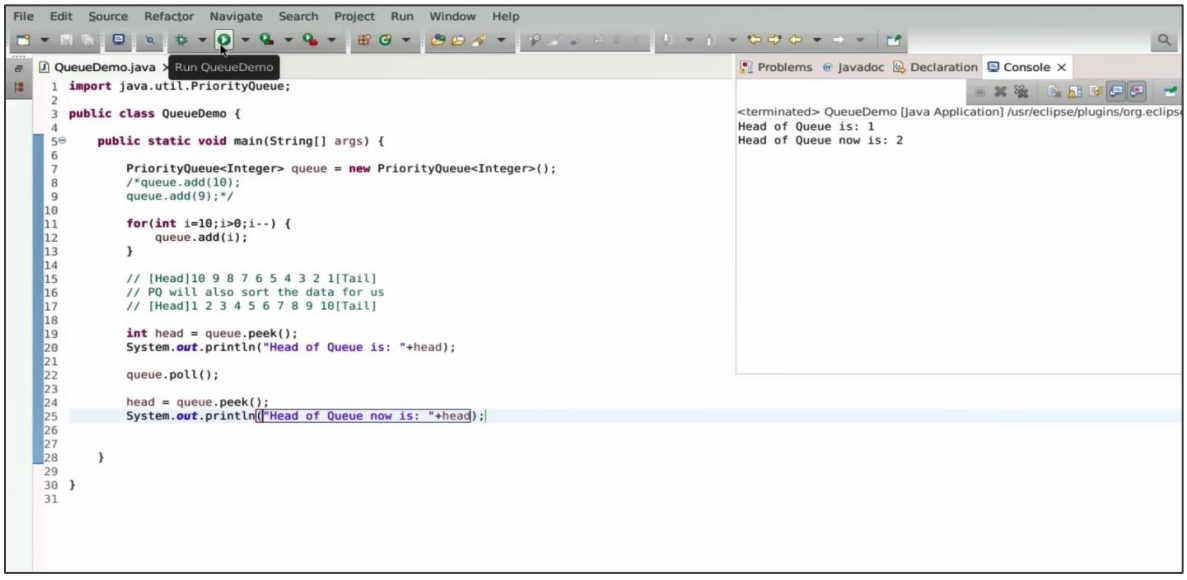


3.6 Display the new head by writing: **"Head of Queue now is: " + head**

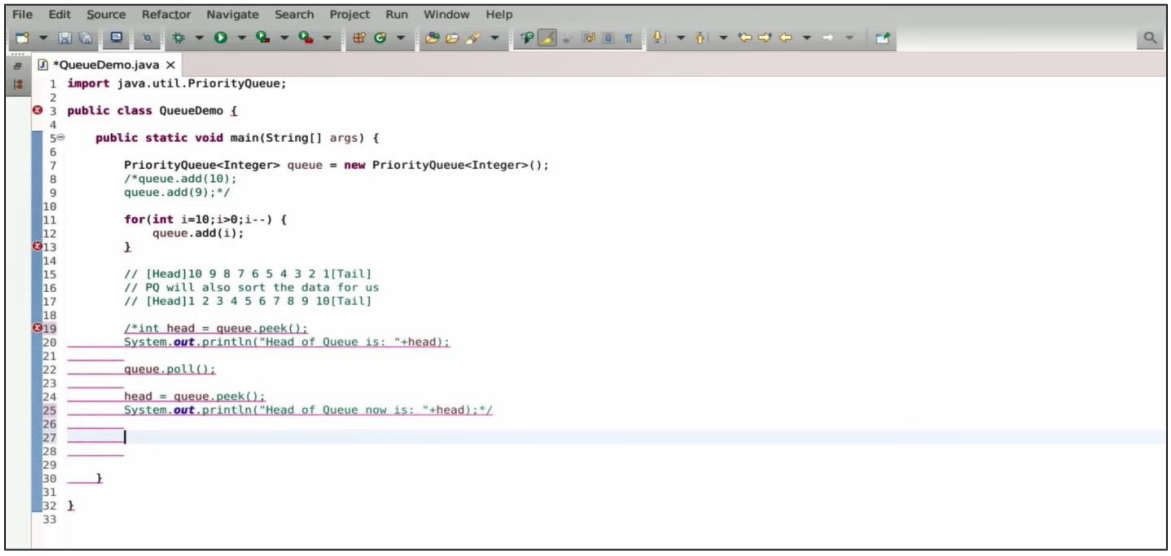


3.7 Output is shown below:

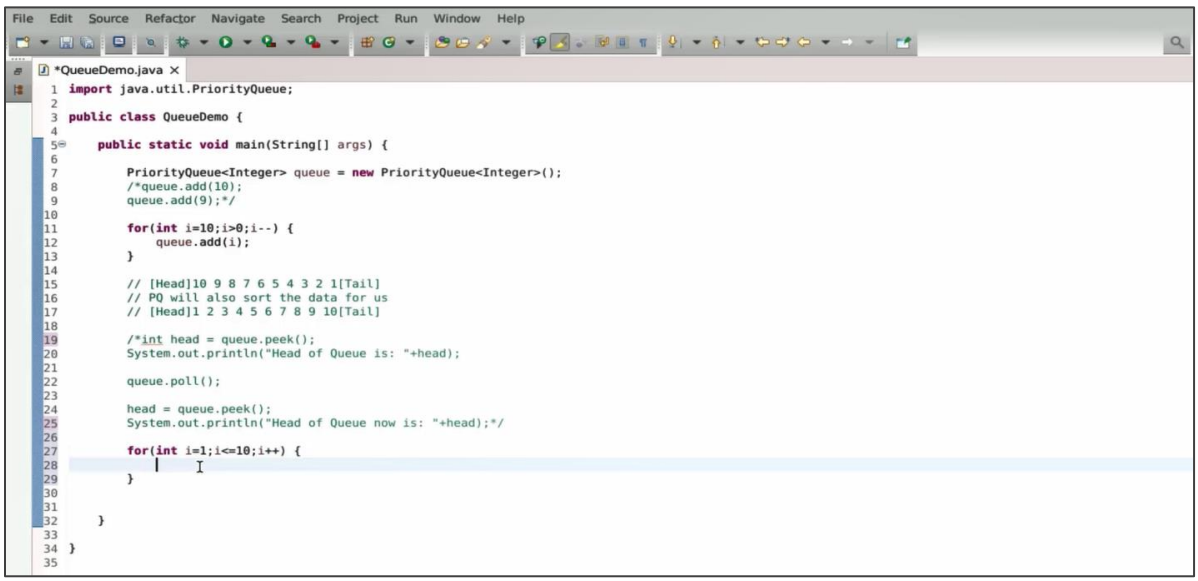
Head of Queue now is: 2



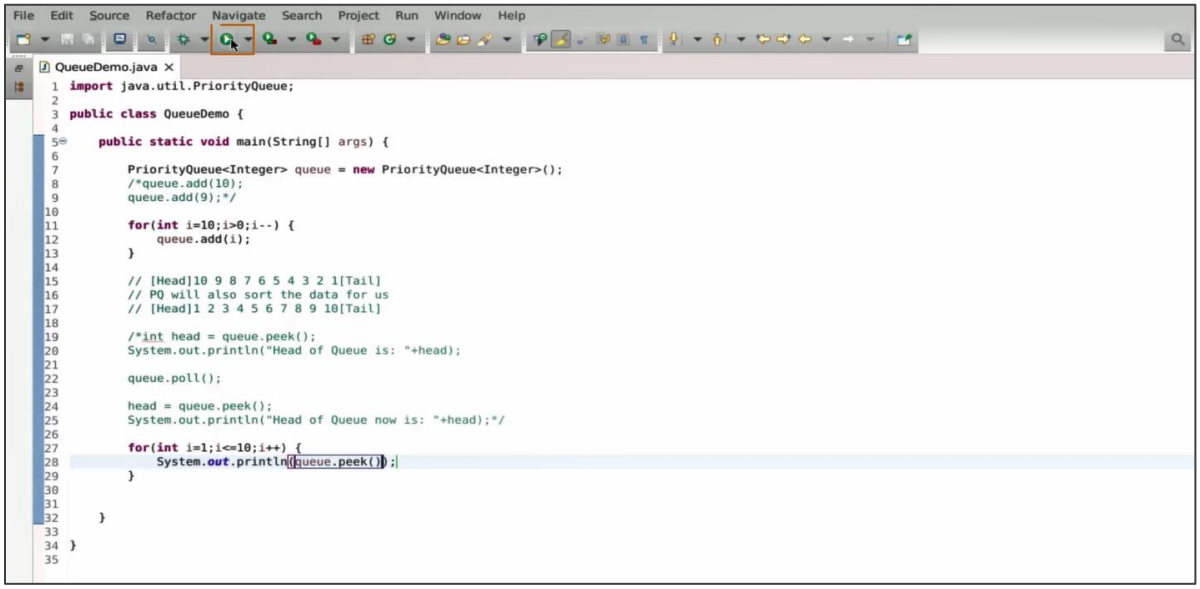
3.8 Comment out the peek and poll code



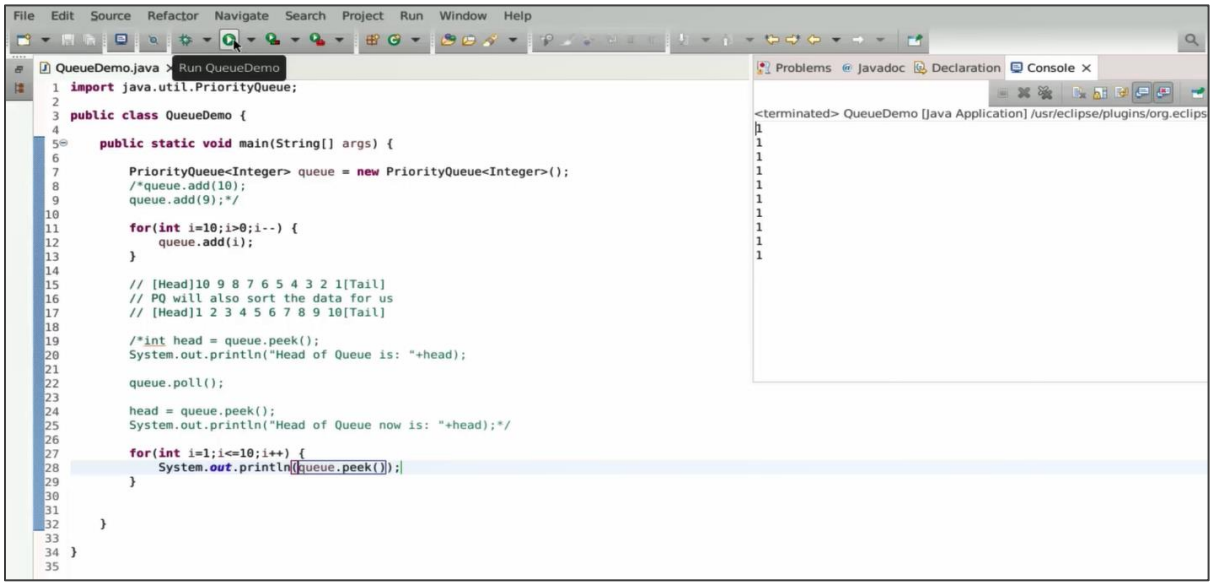
3.9 Write a loop starting with **i = 1, i <= 10,** and **i++** to iterate over the queue



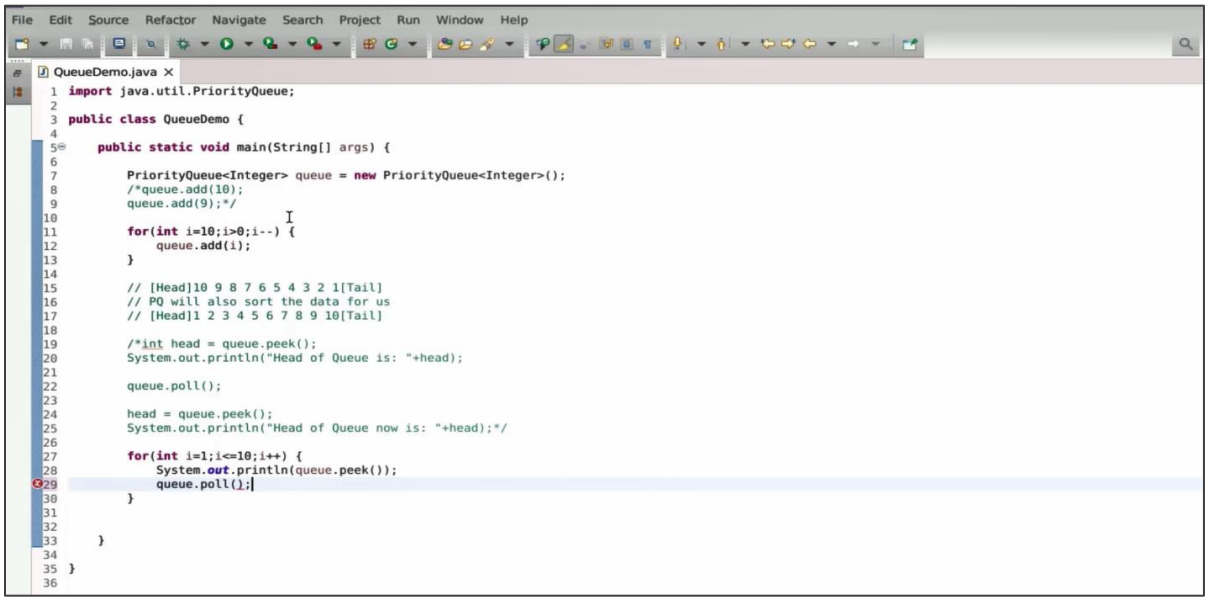
3.10 Print the head of the queue using **queue.peek()**



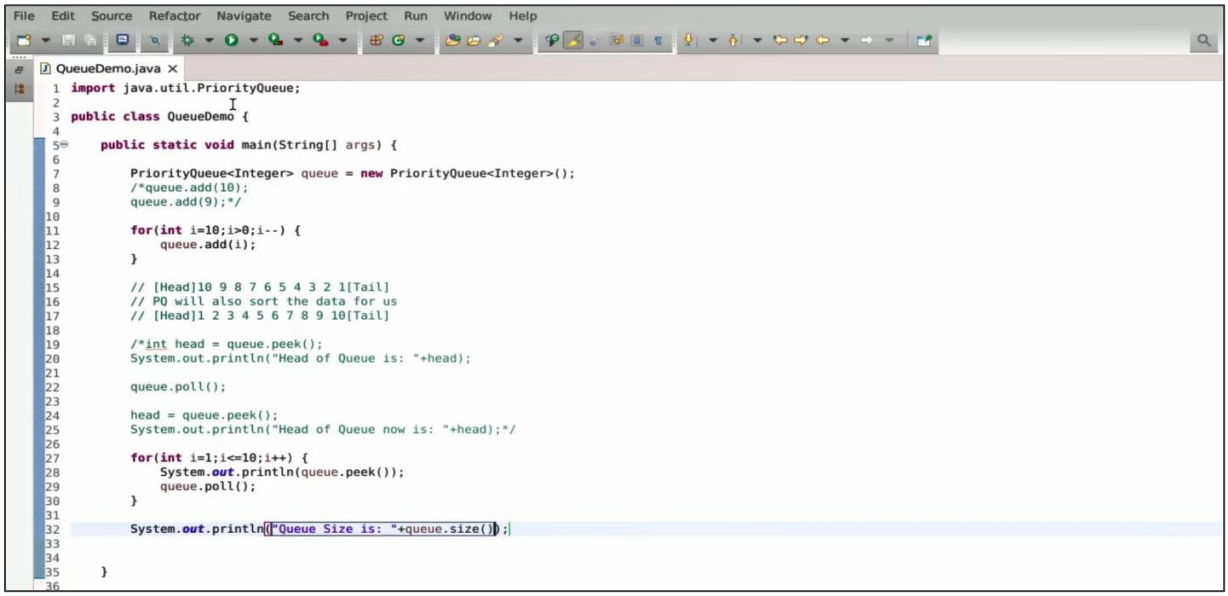
3.11 Output will display the head as 1 ten times



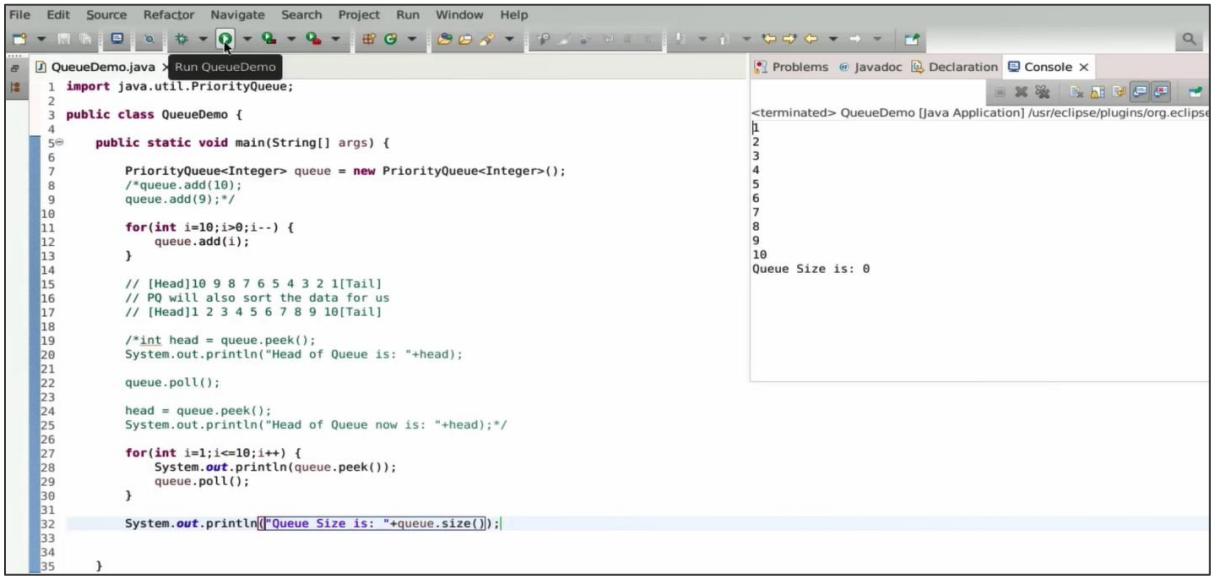
3.12 Instead of using **peek(),** you can use **queue.poll()** inside the loop to remove the head and display all the elements in the queue



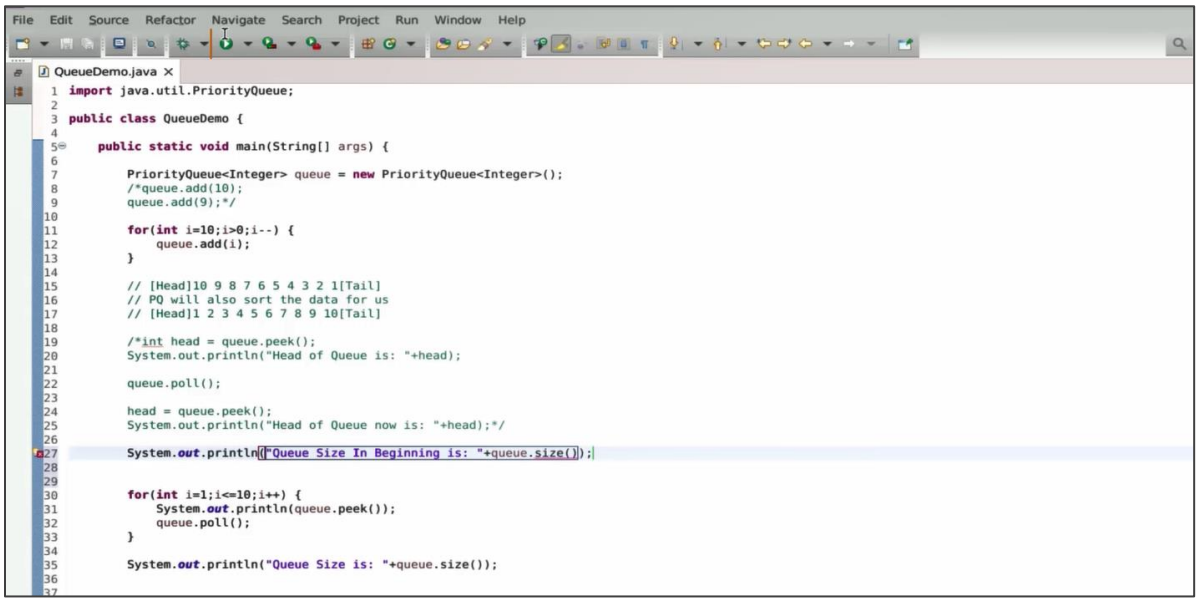
3.13 After the loop, print the size of the queue using **"Queue Size is: " + queue.size()**



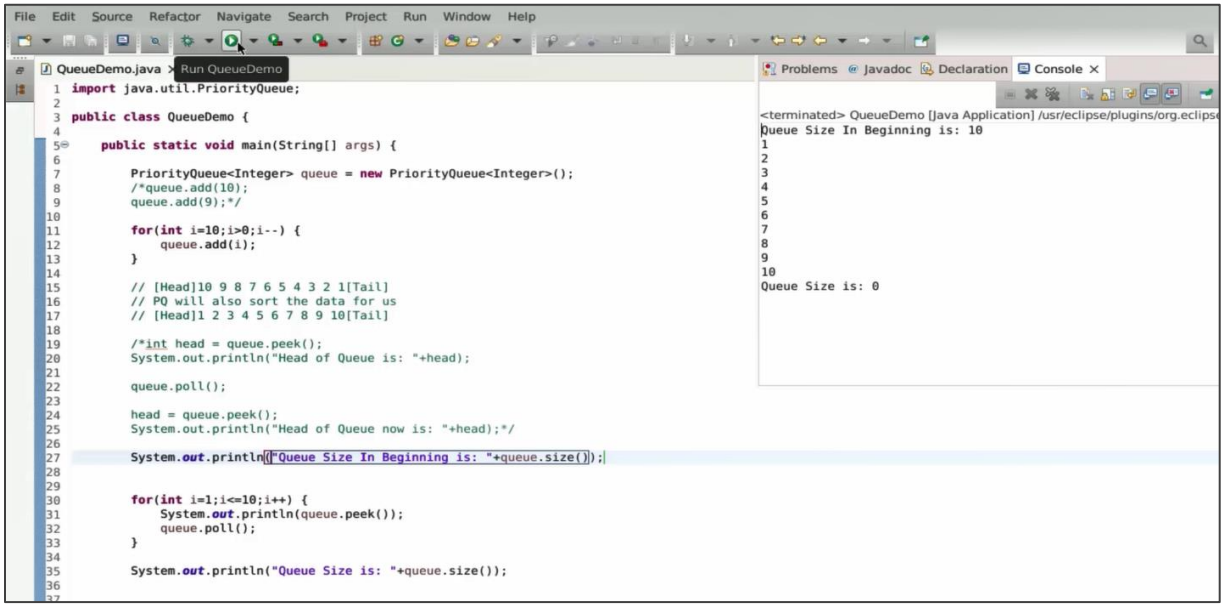
3.14 Running this code will display that the **Queue Size is: 0**



3.15 Before the loop, print the size of the queue using **"Queue Size In the Beginning is: " + queue.size()**

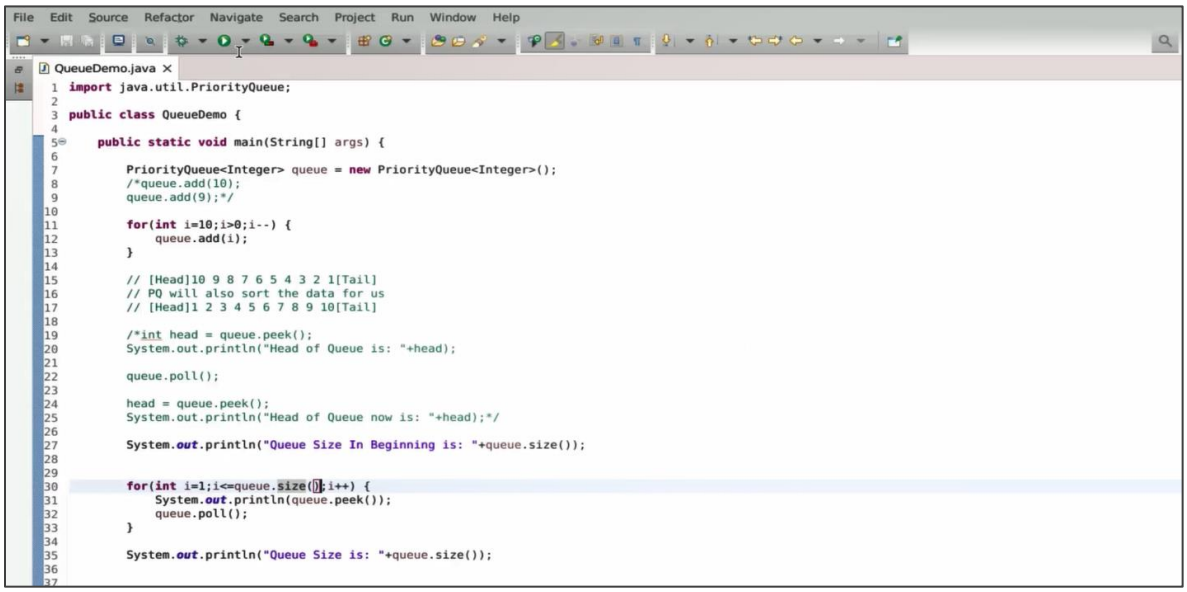


3.16 Running the program will show that the **Queue Size In Beginning is: 10**



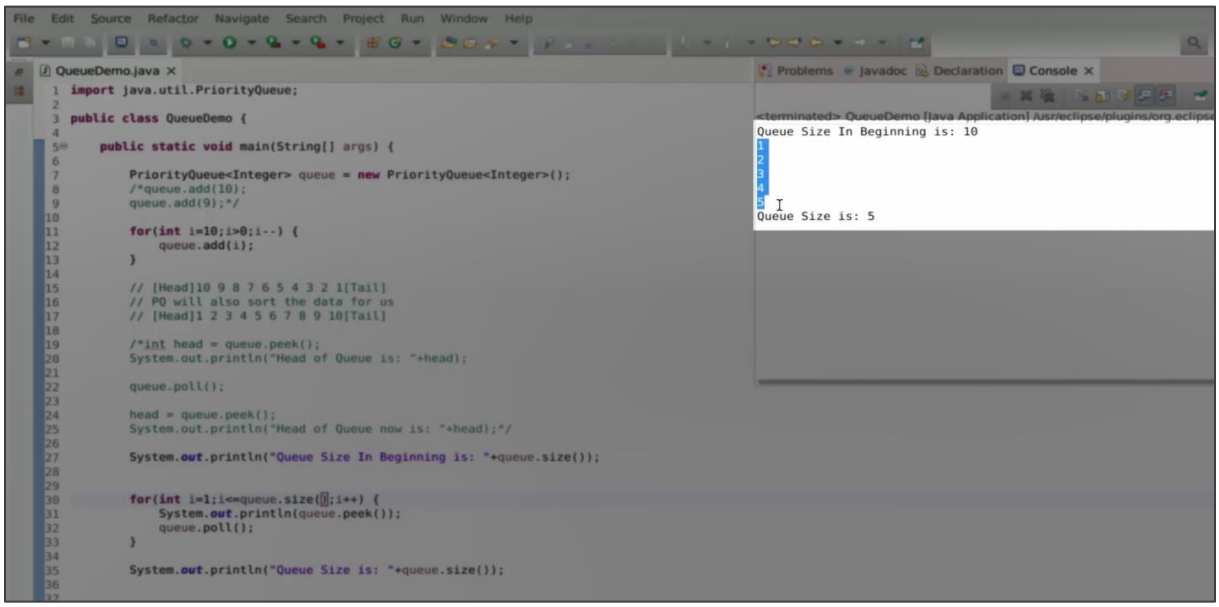

```
import java.util.PriorityQueue;

public class QueueDemo {

    public static void main(String[] args) {

        PriorityQueue<Integer> queue = new PriorityQueue<Integer>();
        /*queue.add(10);
        queue.add(9);*/

        for(int i=10;i>0;i--) {
            queue.add(i);
        }

        // [Head]10 9 8 7 6 5 4 3 2 1[Tail]
        // PQ will also sort the data for us
        // [Head]1 2 3 4 5 6 7 8 9 10[Tail]

        /*int head = queue.peek();
        System.out.println("Head of Queue is: "+head);

        queue.poll();

        head = queue.peek();
        System.out.println("Head of Queue now is: "+head);*/

        System.out.println("Queue Size In Beginning is: "+queue.size());


        for(int i=1;i<=10;i++) {
            System.out.println(queue.peek());
            queue.poll();
        }

        System.out.println("Queue Size is: "+queue.size());
```

3.17 Modify the loop condition from **i <= 10** to **i <= queue.size()**



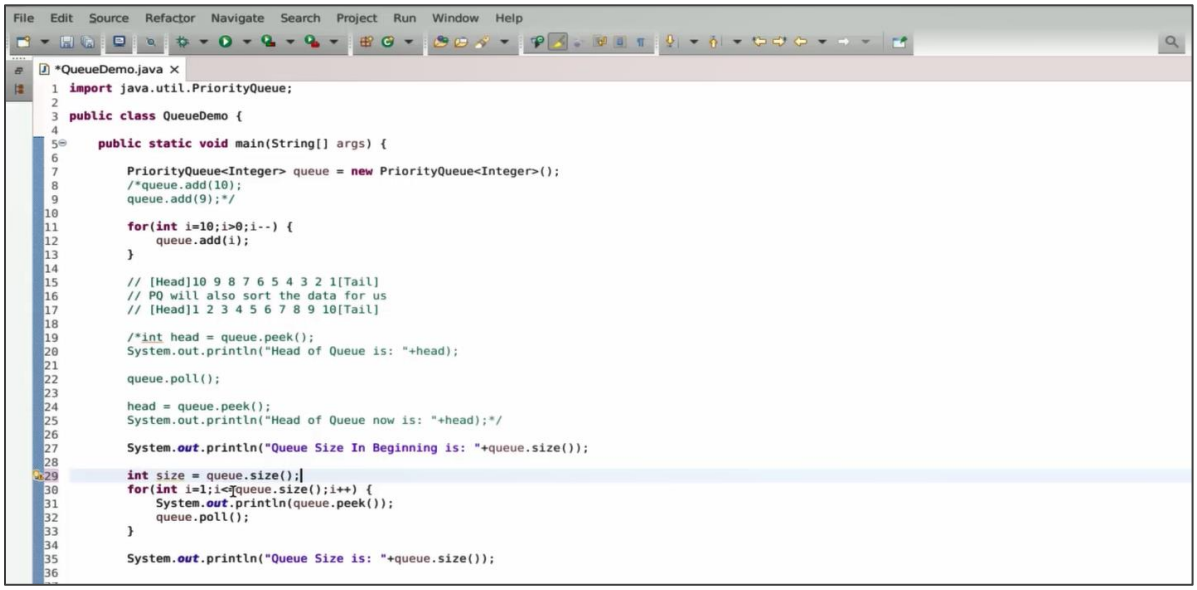

```
import java.util.PriorityQueue;

public class QueueDemo {

    public static void main(String[] args) {

        PriorityQueue<Integer> queue = new PriorityQueue<Integer>();
        /*queue.add(10);
        queue.add(9);*/

        for(int i=10;i>0;i--) {
            queue.add(i);
        }

        // [Head]10 9 8 7 6 5 4 3 2 1[Tail]
        // PQ will also sort the data for us
        // [Head]1 2 3 4 5 6 7 8 9 10[Tail]

        /*int head = queue.peek();
        System.out.println("Head of Queue is: "+head);

        queue.poll();

        head = queue.peek();
        System.out.println("Head of Queue now is: "+head);*/

        System.out.println("Queue Size In Beginning is: "+queue.size());


        for(int i=1;i<=queue.size();i++) {
            System.out.println(queue.peek());
            queue.poll();
        }

        System.out.println("Queue Size is: "+queue.size());
```

3.18 Running this code may give incorrect output. Whenever **poll** is called on the queue inside the loop, the size decreases, which affects the loop condition. Therefore, it is challenging to **use queue.size()** inside the loop for iterative purposes.

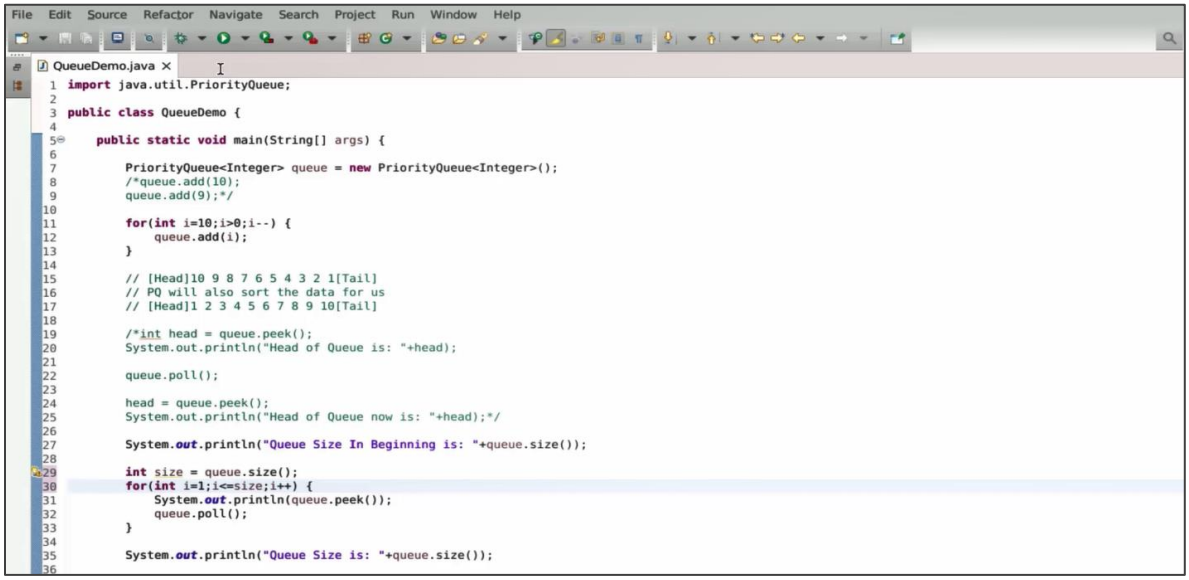


Step 4: Implement the use of variable size using queue.size()

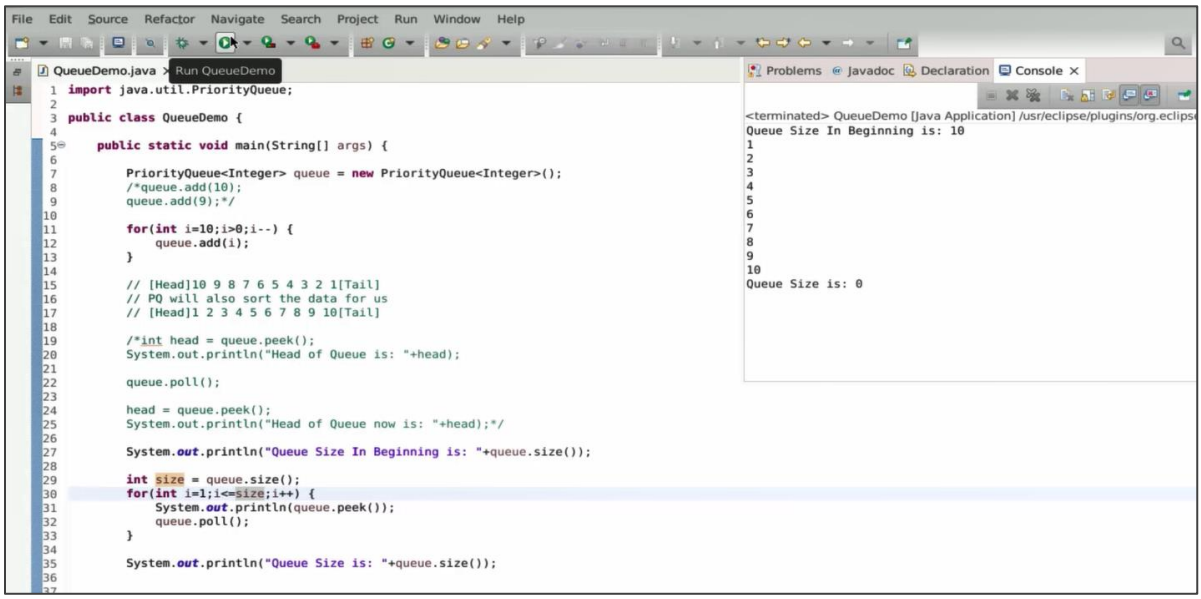
4.1 Declare a separate variable size and assign it **queue.size()**



4.2 Modify the loop condition to **i <= size**

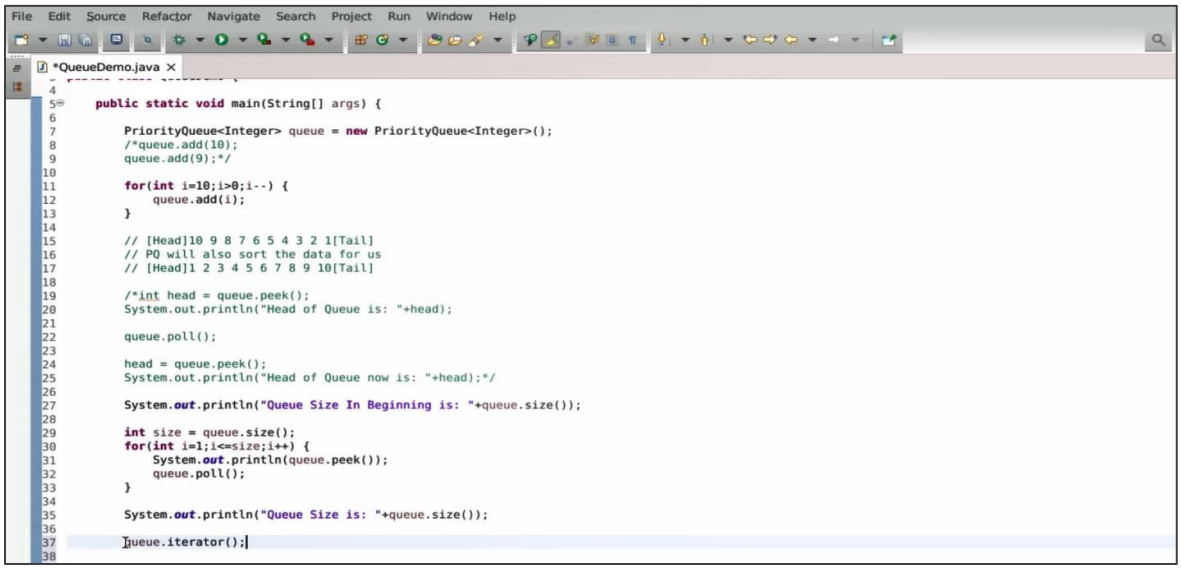


4.3 When processing the queue in an algorithmic approach, if using **poll()** inside the loop, the queue cannot be cleared until the end



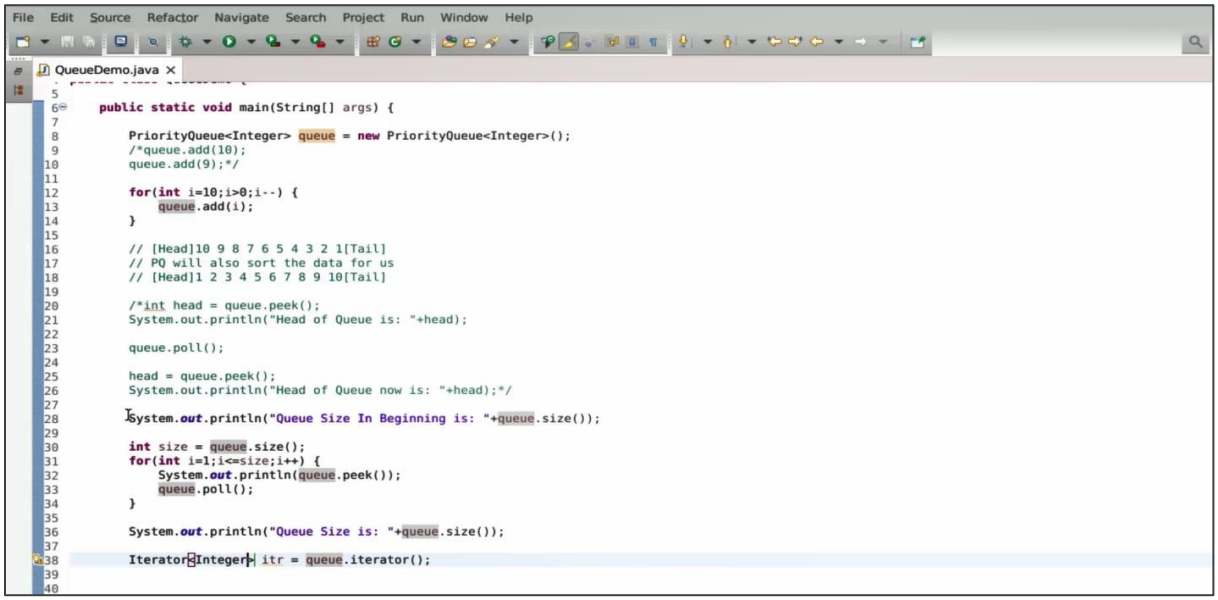
Step 5: Use the queue.iterator() function with example data

5.1 Write: **queue.iterator();**

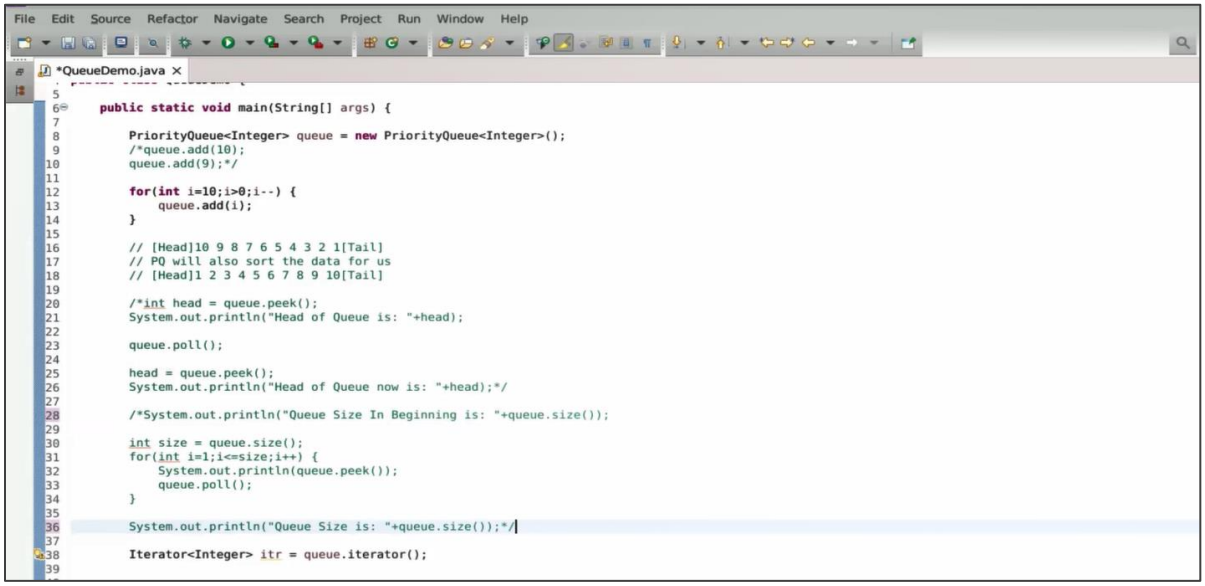


Most collection APIs support iterators.

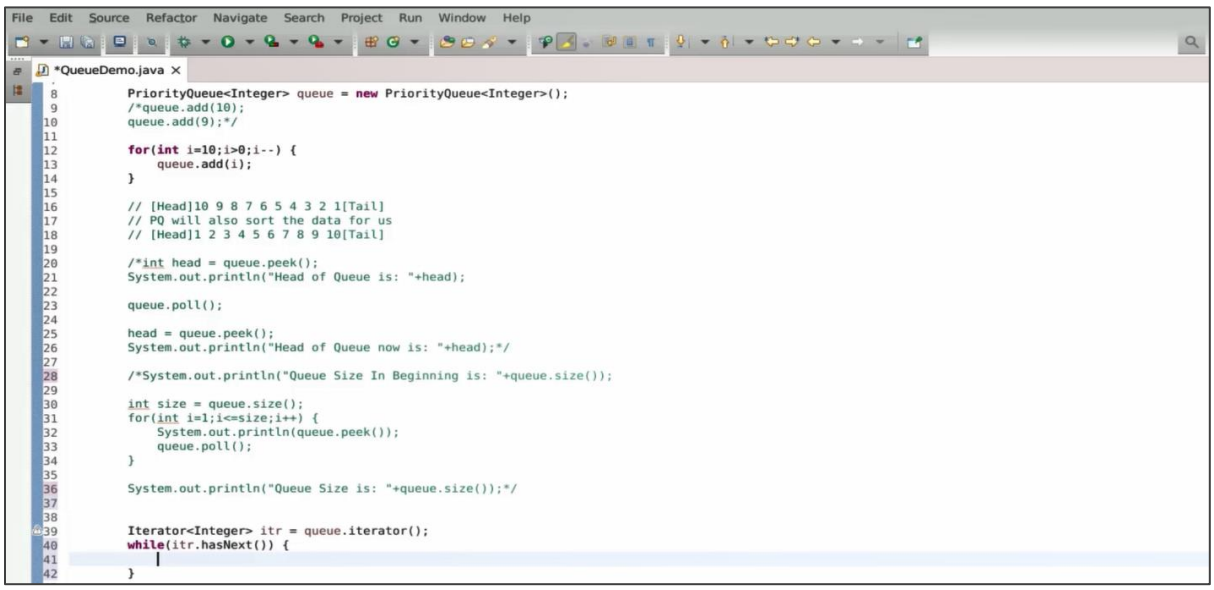
5.2 Declare an iterator with the data type Integer by writing: **Iterator<Integer> itr = queue.iterator();**



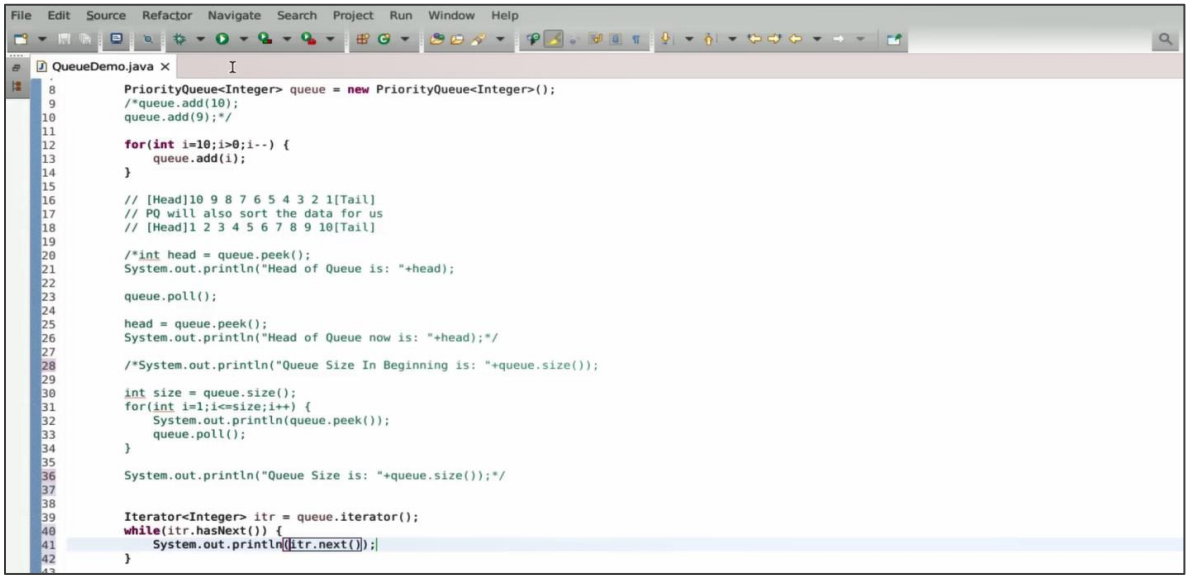
5.3 Comment out the previous code



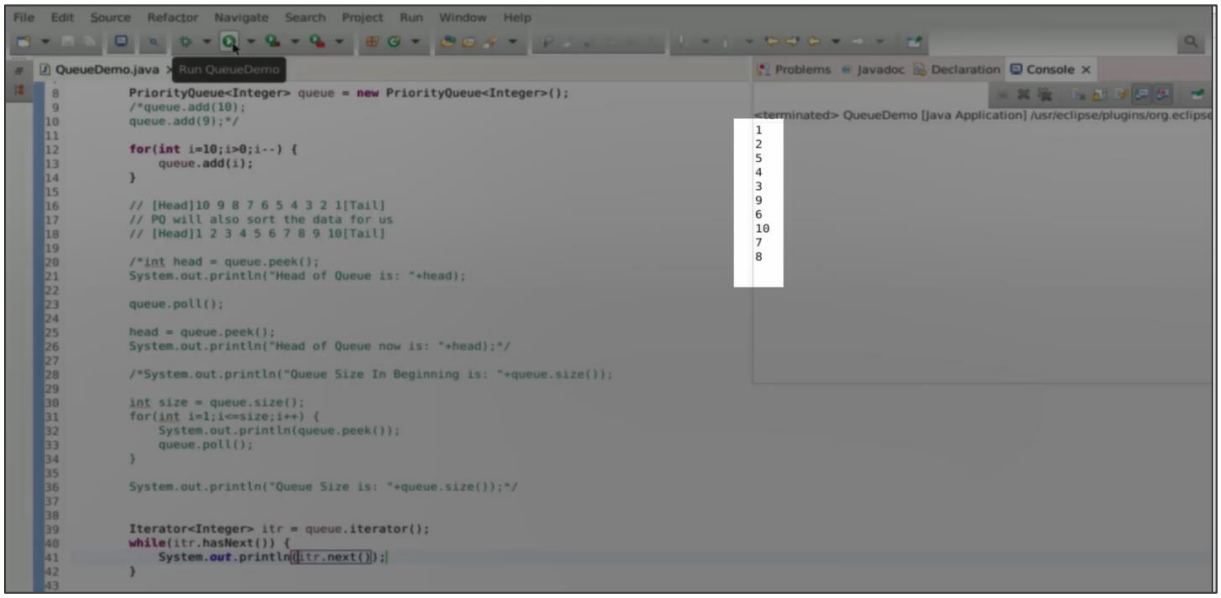
5.4 Iterate over the queue using a while loop with **itr.hasNext()** as the condition



5.5 Print each element using **itr.next()**



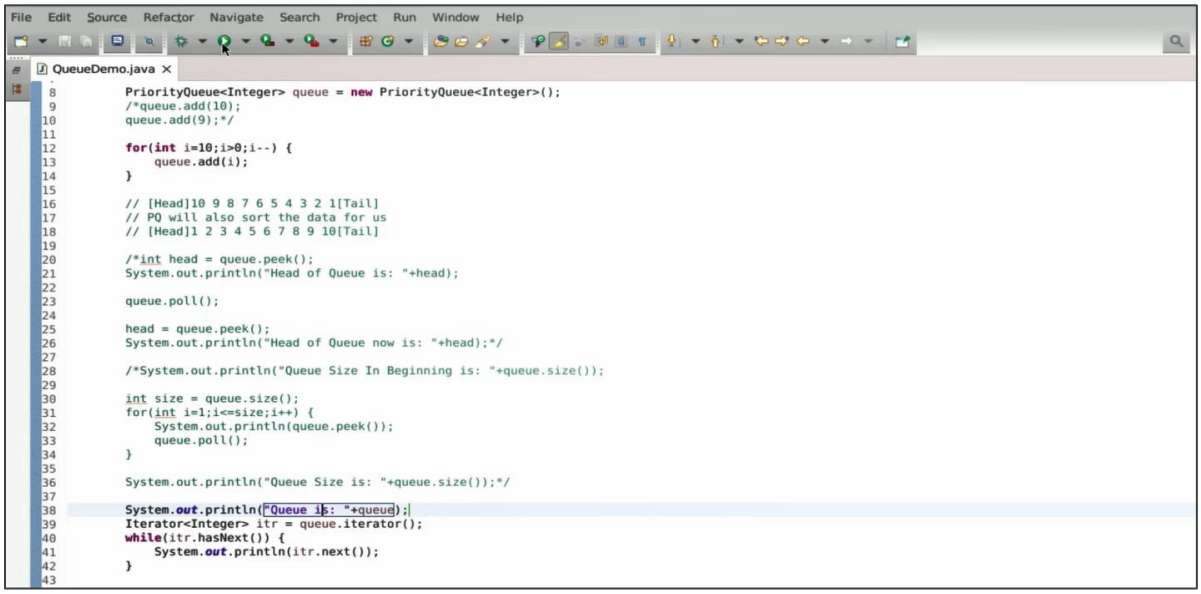
The iteration will display the data in an unordered manner.



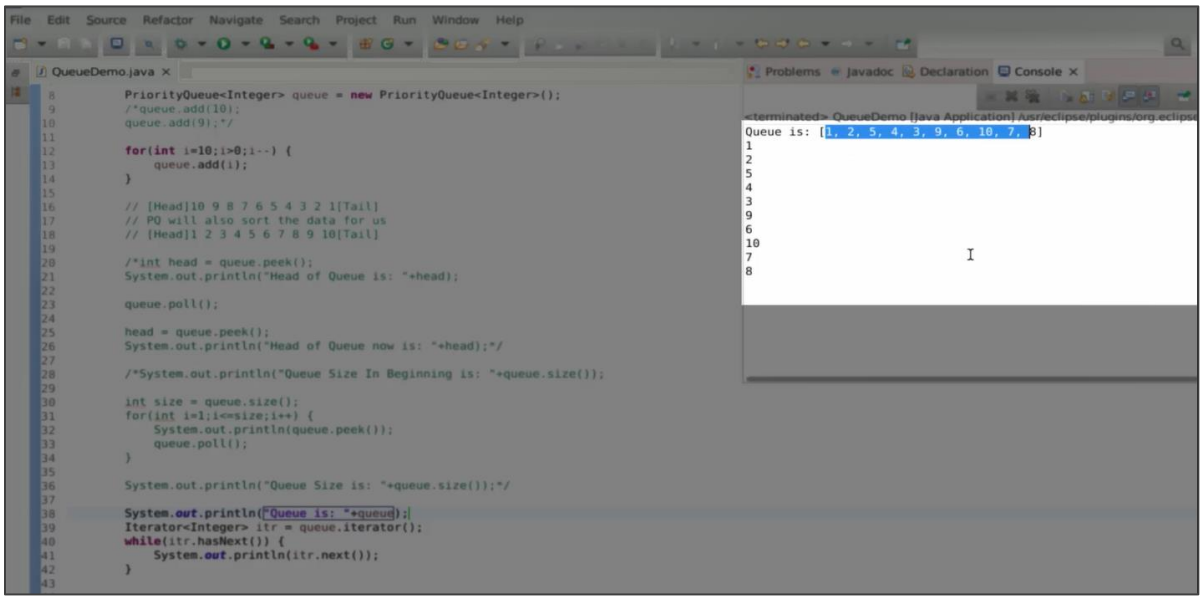
The priority queue automatically sorts the data, allowing you to process the queue using a first-in-first-out approach with sorted data.

Step 6: Implement of queue prioritization

6.1 When trying to print the queue, for example, write: **"Queue is: " + queue**



6.2 The data stored in the queue will be displayed in an unordered arrangement



By following these steps, you'll gain skills to manage and prioritize data efficiently using a PriorityQueue, which automatically sorts elements for processing.