**Laboratory 8:**

**TEST PROGRAMS:**

**TEST PROGRAM 1:**

**package** laboratory8;

**import** java.util.Queue;

**import** java.util.LinkedList;

**public** **class** linklist {

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

// creating queue using the linked list class

Queue<Integer> num = **new** LinkedList<>();

// offer elememts the queue

num.offer(1);

num.offer(2);

num.offer(3);

System.***out***.println("Queue: " + num);

// access elements of the queue

**int** accessednum = num.peek();

System.***out***.println("Accessed element is: " + accessednum);

// removing elements from the queue

**int** removenum = num.poll();

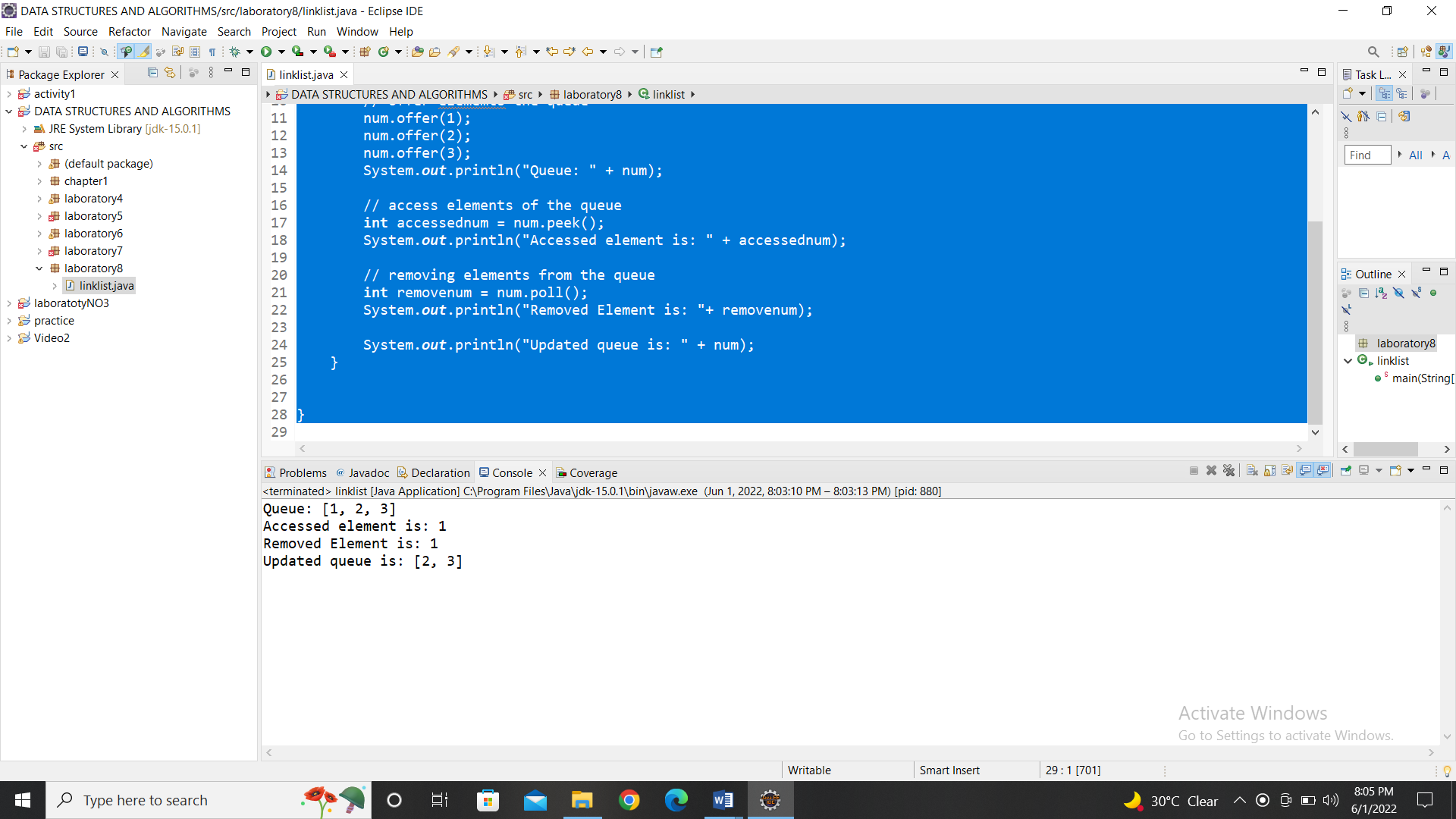
System.***out***.println("Removed Element is: "+ removenum);

System.***out***.println("Updated queue is: " + num);

}

}

OUTPUT:



**TEST PROGRAM 2:**

**package** laboratory8;

//a class to represent a queue

**class** que {

**private** **int**[] arr; // array to store queue elements

**private** **int** front; // front points to the front elements of the queue

**private** **int** rear; // rear points to the last elements odf the queue

**private** **int** capacity; // maximum capacity of the queue

**private** **int** count; // curent size of the queue

// constructor to initialize the queue

que(**int** size)

{

arr = **new** **int**[size];

capacity = size;

front = 0;

rear = -1;

count = 0;

}

// Utility function to return the size of the queue

**public** **int** size()

{

**return** count;

}

// Utility function to check if the queue is empty or not

**public** **boolean** isEmpty()

{

**return** (size() == 0);

}

// Utility function to check if the queue is full or not

**public** **boolean** isFull()

{

**return** (size() == capacity);

}

**public** **int** dequeue() {

**if** (isEmpty())

{

System.***out***.println("Underflow/n Program Terminated");

System.*exit*(-1);

}

**int** x = arr[front];

System.***out***.println("Removing " + x);

front = (front + 1) % capacity;

count--;

**return** x;

}

// Utility function to add an item to the queue

**public** **void** enqueue(**int** item)

{

**if** (isFull())

{

System.***out***.println("Overflow\nProgram Terminated");

System.*exit*(-1);

}

System.***out***.println("Inserting " + item);

rear = (rear + 1) % capacity;

arr[rear] = item;

count++;

}

// utilily function to return the front element of the queue

**public** **int** peek()

{

**if** (isEmpty())

{

System.***out***.println("UnderFlow\nProgam terminated");

System.*exit*(-1);

}

**return** arr[front];

}

}

**public** **class** queue {

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

// creating a queue of capacity 5

que q1 = **new** que(5);

q1.enqueue(1);

q1.enqueue(2);

q1.enqueue(3);

System.***out***.println("The front element is " + q1.peek());

q1.dequeue();

System.***out***.println("The front element is " + q1.peek());

System.***out***.println("The size of the queue is: " + q1.size());

q1.dequeue();

q1.dequeue();

**if** (q1.isEmpty()) {

System.***out***.println("The queue is empty:");

}

**else** {

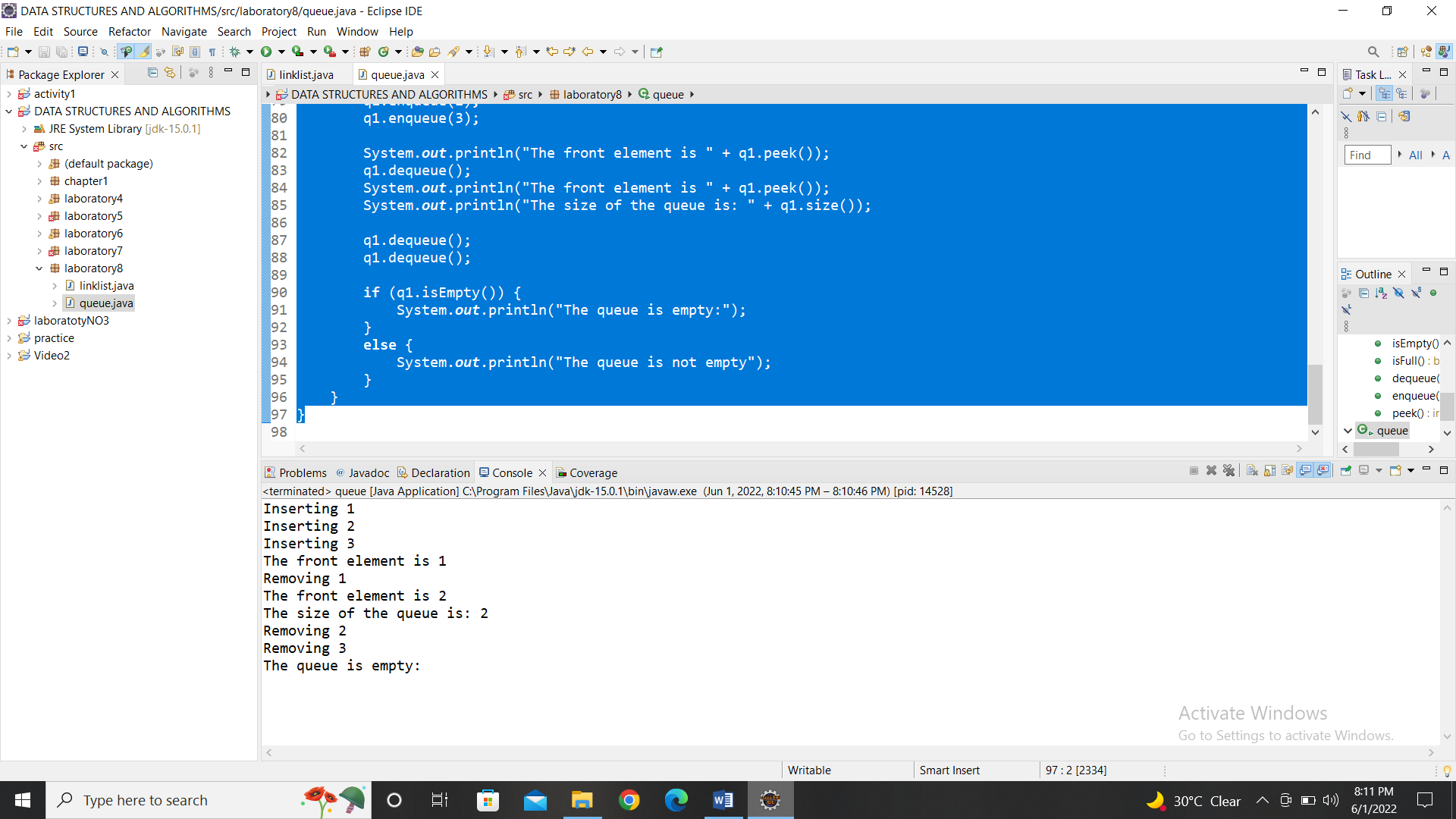
System.***out***.println("The queue is not empty");

}

}

}

Output:



**TEST PROGRAM 3:**

**package** laboratory8;

**import** java.util.Queue;

**import** java.util.PriorityQueue;

**public** **class** pqueue {

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

//creating queue using the priority queue class

Queue<Integer> num = **new** PriorityQueue<>();

//offering elements to the queue

num.offer(5);

num.offer(1);

num.offer(1);

System.***out***.println("Queue: " + num);

//Access elements of the queue

**int** accnum = num.peek();

System.***out***.println("Accessed Element:" + accnum);

//Removing elements from the queue

**int** delnum = num.poll();

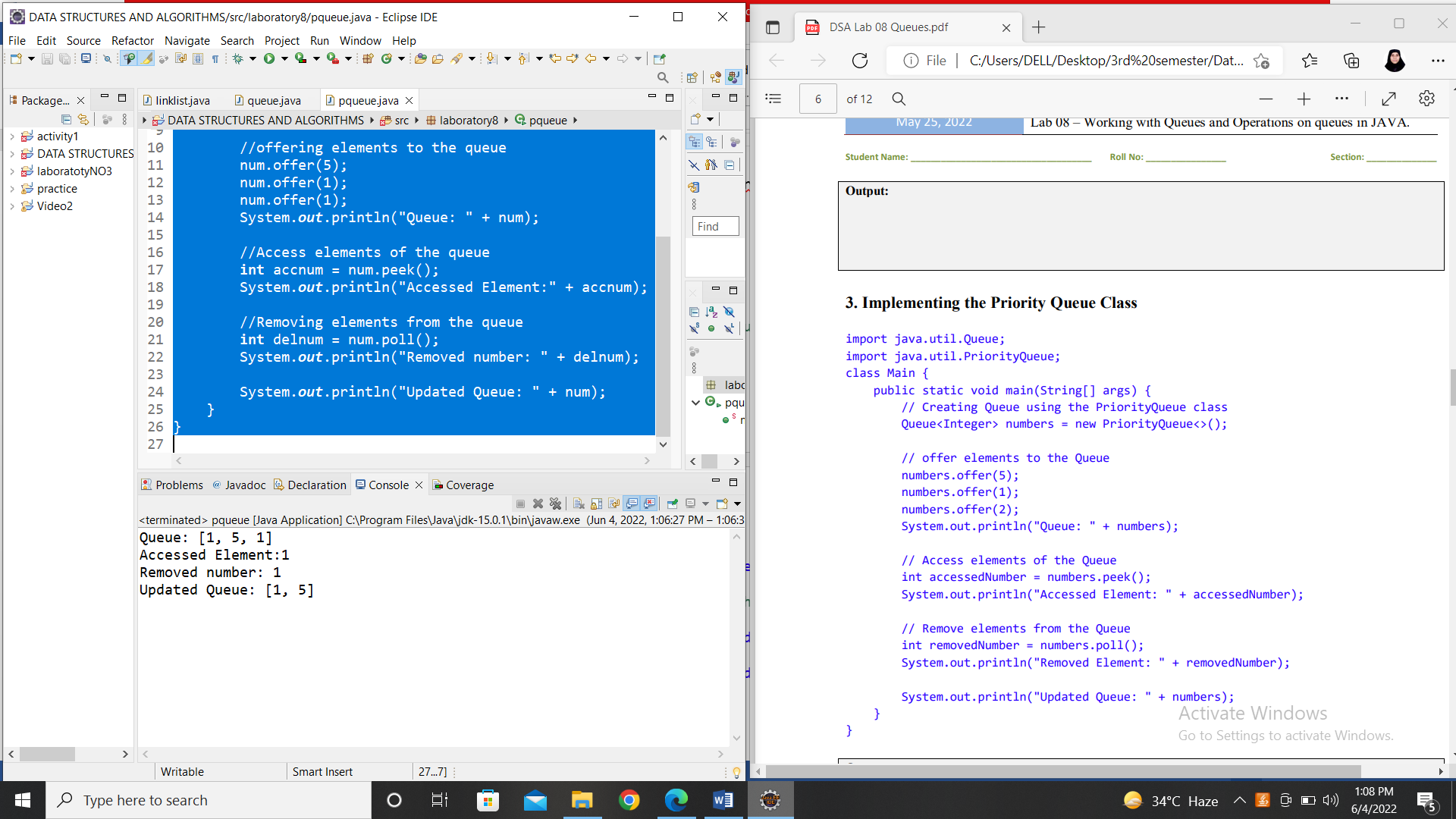
System.***out***.println("Removed number: " + delnum);

System.***out***.println("Updated Queue: " + num);

}

}

Output:



**TEST PROGRAM 4:**

**package** laboratory8;

**import** java.util.PriorityQueue;

**public** **class** pq {

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

PriorityQueue<Character> chr = **new** PriorityQueue<>();

chr.add('a');

chr.add('b');

chr.add('y');

chr.add('z');

System.***out***.println("Priority Queue is: "+ chr);

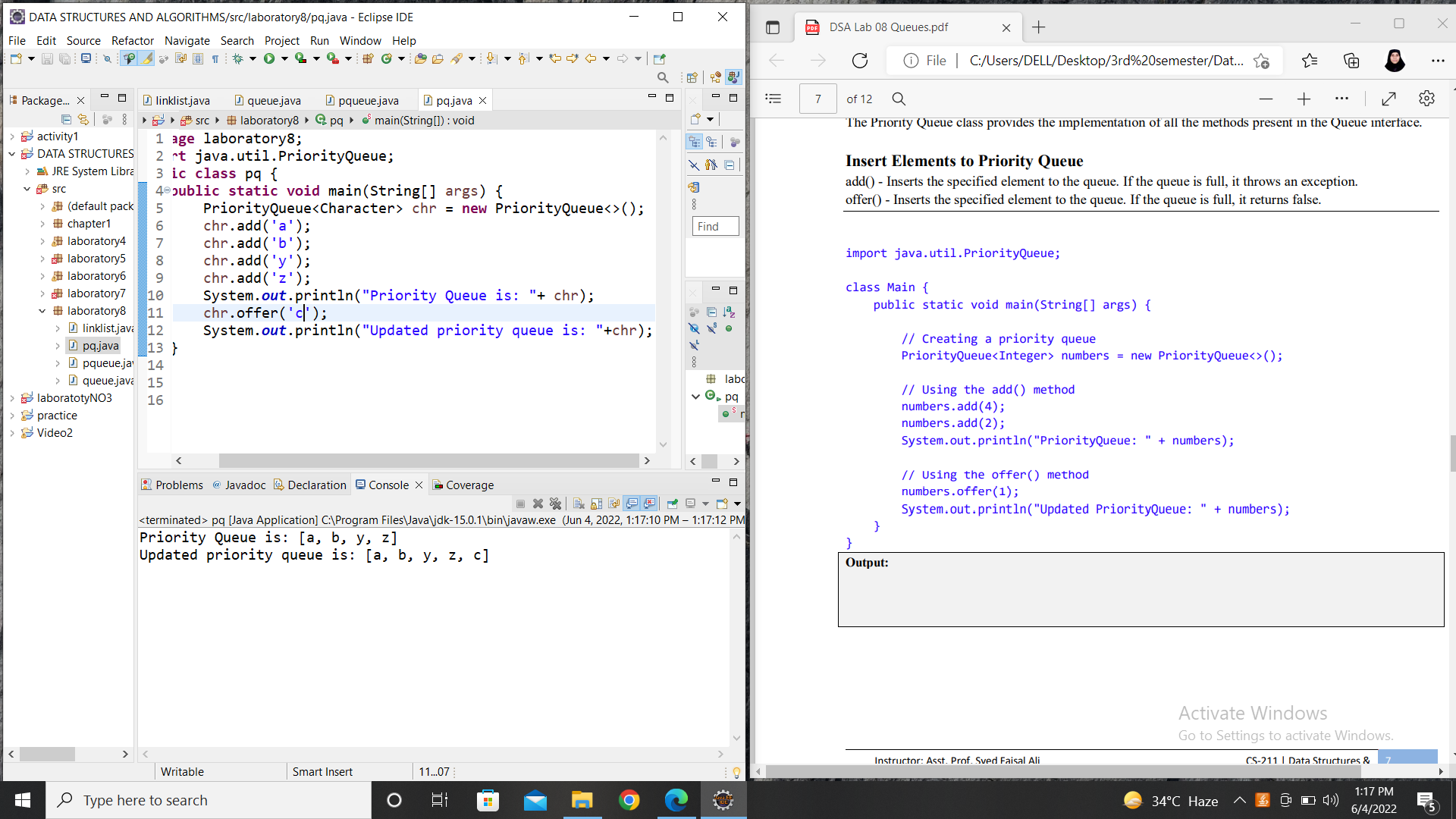
chr.offer('c');

System.***out***.println("Updated priority queue is: "+chr);

}

}

Output:



**TEST PROGRAM 5:**

**package** laboratory8;

**import** java.util.PriorityQueue;

**import** java.util.Iterator;

**public** **class** pqmethods {

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

// Creating a priority queue

PriorityQueue<Integer> numbers = **new** PriorityQueue<>();

numbers.add(4);

numbers.add(2);

numbers.add(1);

numbers.add(5);

numbers.add(6);

numbers.add(7);

numbers.add(12);

System.***out***.println("PriorityQueue: " + numbers);

// Using the peek() method

**int** number = numbers.peek();

System.***out***.println("Accessed Element: " + number);

// Using the remove() method

**boolean** result = numbers.remove(2);

System.***out***.println("Is the element 2 removed? " + result);

// Using the poll() method

**int** n= numbers.poll();

System.***out***.println("Removed Element Using poll(): " + n);

// using iterator method

Iterator<Integer> iterate = numbers.iterator();

**while**(iterate.hasNext()) {

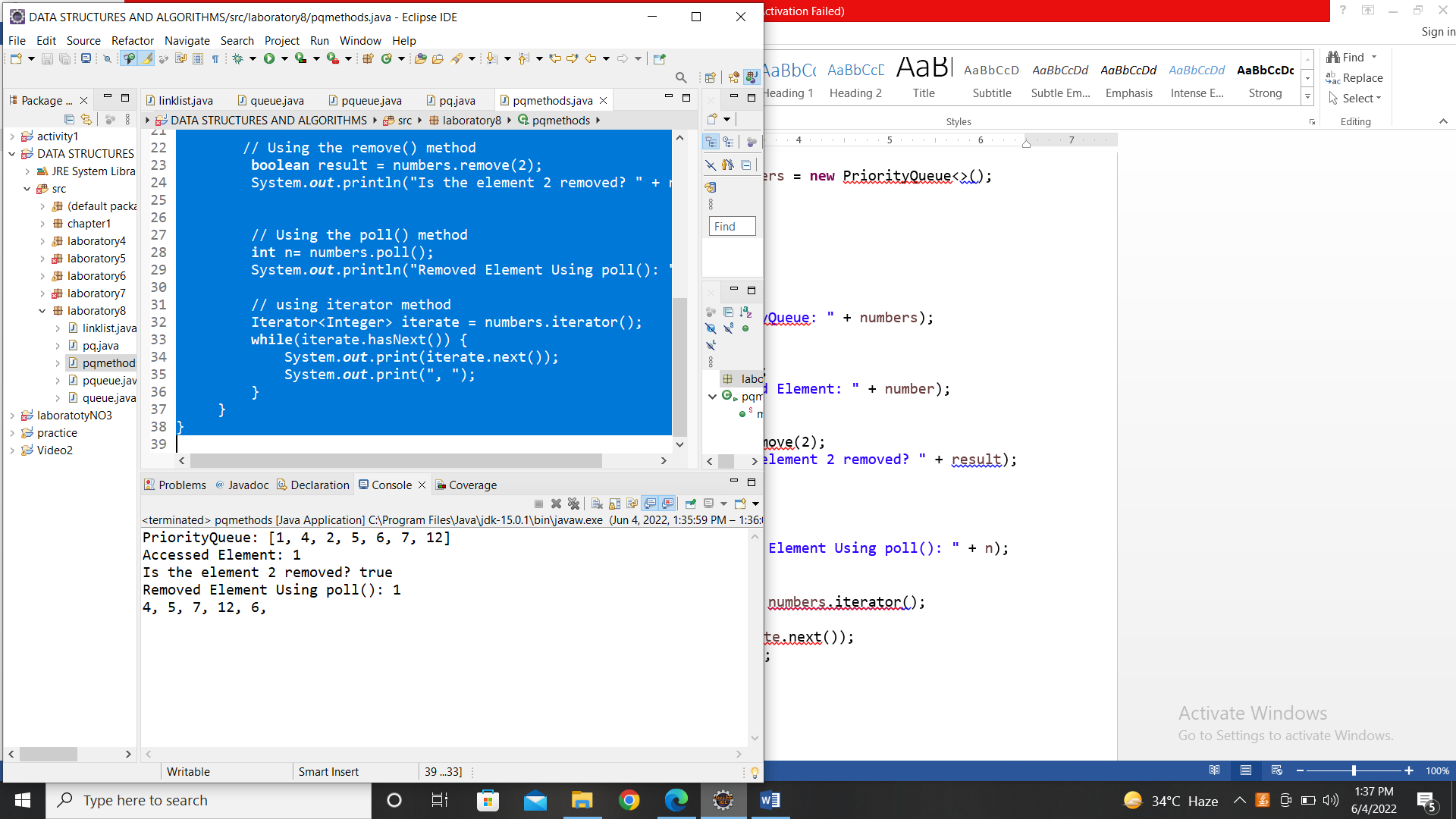
System.***out***.print(iterate.next());

System.***out***.print(", ");

}

} }

Output:



**TEST PROGRAM 6:**

**package** laboratory8;

**import** java.util.PriorityQueue;

**import** java.util.Comparator;

**import** java.util.Iterator;

**class** CustomComparator **implements** Comparator<Integer> {

@Override

**public** **int** compare(Integer num1,Integer num2) {

**int** value = num1.compareTo(num2);

//elements are sorted in reverse order

**if** (value <0) {

**return** -1;

}

**else** **if** (value >0) {

**return** 1;

}

**else** {

**return** 0;

}

}

}

**public** **class** pqcompare {

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

// Creating a priority queue

PriorityQueue<Integer> numbers = **new** PriorityQueue<>(**new** CustomComparator());

numbers.add(4);

numbers.add(2);

numbers.add(1);

numbers.add(3);

System.***out***.println("PriorityQueue: " + numbers);

// using iterator method

System.***out***.println("Priority Queue after using Iterator is:");

Iterator<Integer> iterate = numbers.iterator();

**while**(iterate.hasNext()) {

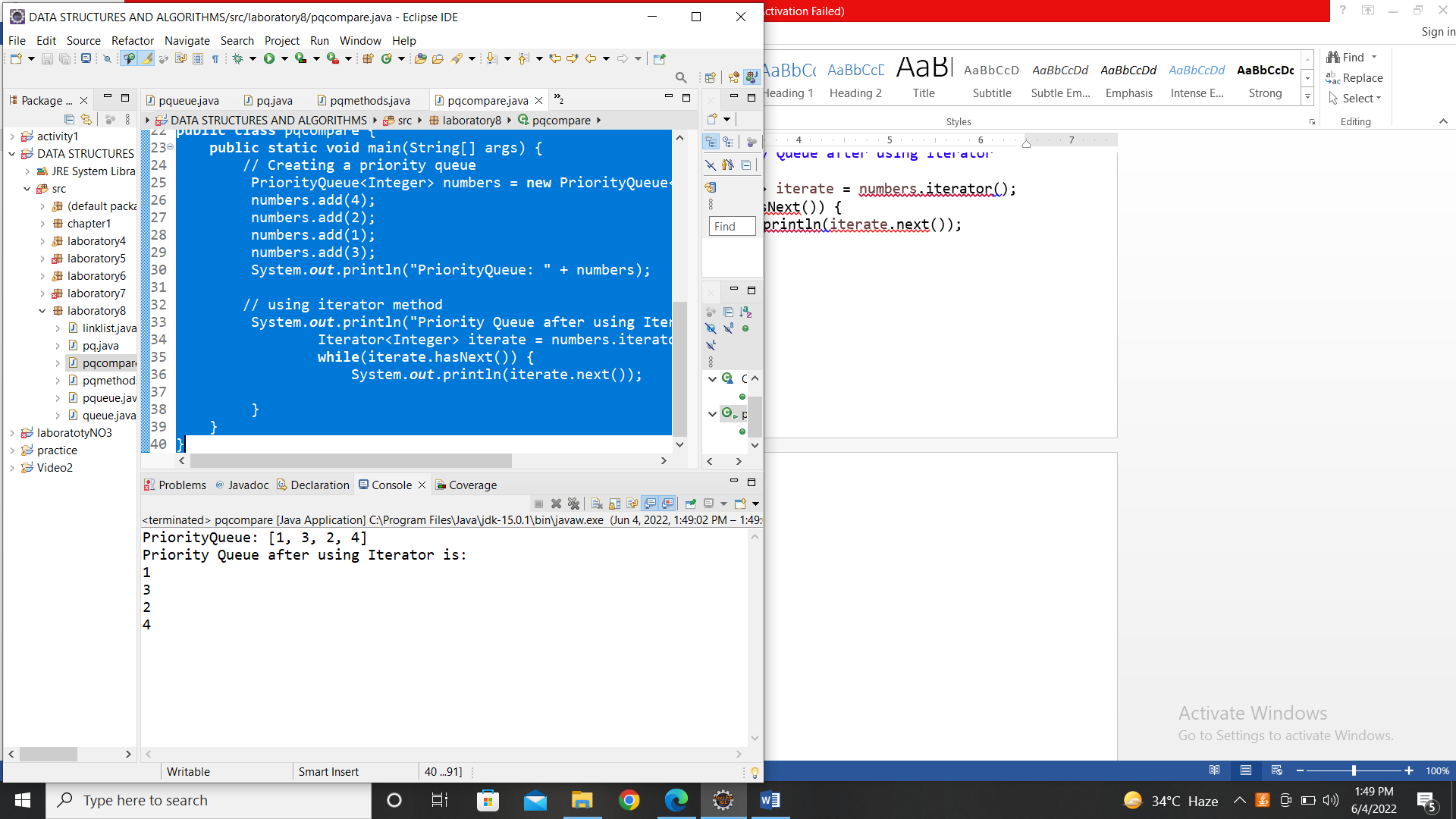
System.***out***.println(iterate.next());

}

}

}

Output:



**PROGRAMMING EXERCISE:**

**QUESTION 1:**

Application of Queue in Data Structure

* Managing requests on a single shared resource such as CPU scheduling and disk scheduling
* Handling website traffic
* Routers and switches in networking
* Maintaining the playlist in media players
* Applied to add song at the end or to play from the front.
* In real life scenario, Call Center phone systems use Queues to hold people calling them in order, until a service representative is free.
* Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrive i.e First come first served.

**QUESTION 2:**

**package** laboratory8;

//Java program to demonstrate

//FCFS Disk Scheduling algorithm

**class** GFG

{

**static** **int** *size* = 8;

**static** **void** FCFS(**int** arr[], **int** head)

{

**int** seek\_count = 0;

**int** distance, cur\_track;

**for** (**int** i = 0; i < *size*; i++)

{

cur\_track = arr[i];

// calculate absolute distance

distance = Math.*abs*(cur\_track - head);

// increase the total count

seek\_count += distance;

// accessed track is now new head

head = cur\_track;

}

System.***out***.println("Total number of " + "seek operations = " + seek\_count);

// Seek sequence would be the same

// as request array sequence

System.***out***.println("Seek Sequence is");

**for** (**int** i = 0; i < *size*; i++)

{

System.***out***.println(arr[i]);

}

}

//Driver code

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

{

// request array

**int** arr[] = { 176, 79, 34, 60,92, 11, 41, 114 };

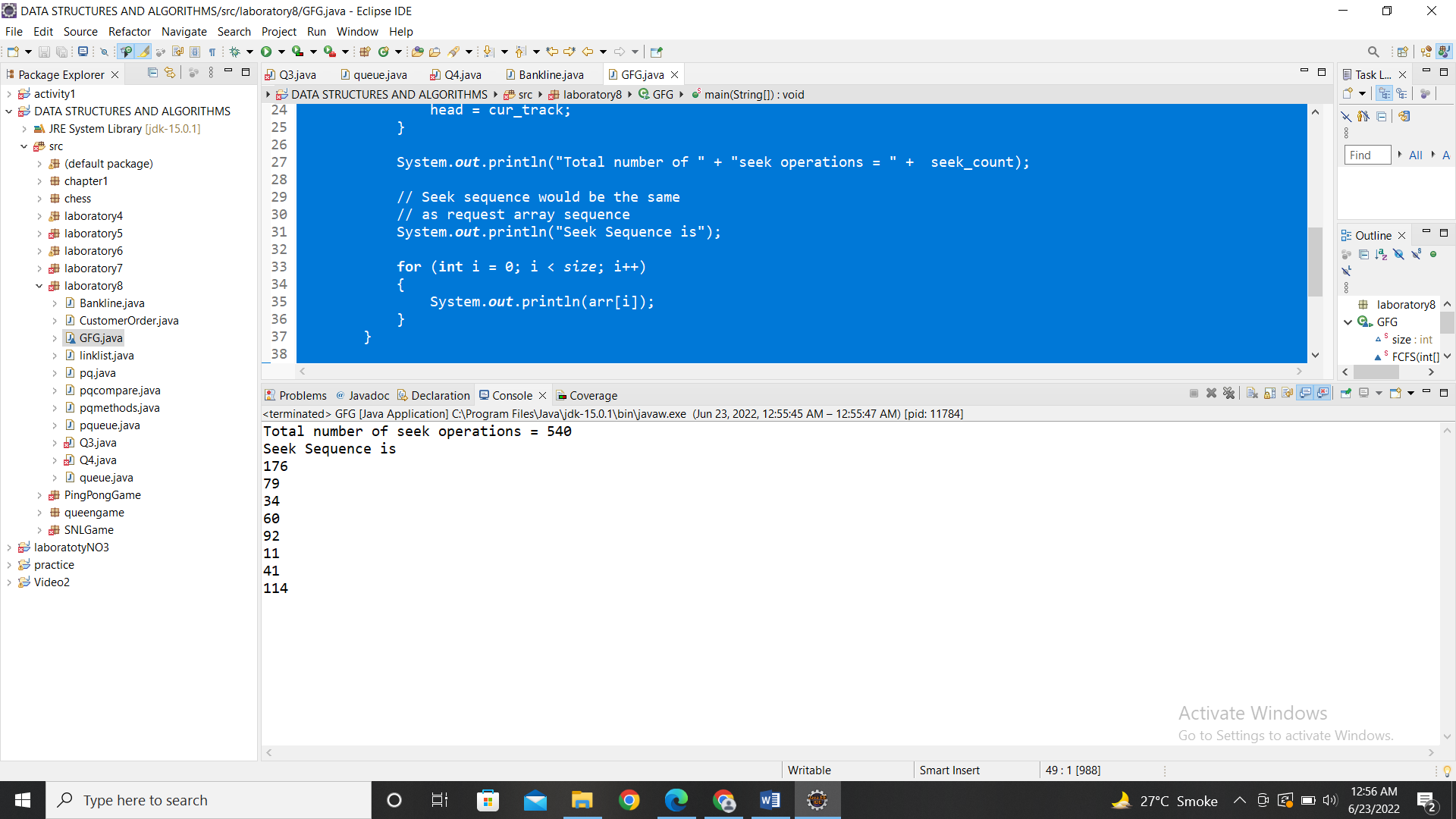
**int** head = 20;

*FCFS*(arr, head);

}

}

Output:



**QUESTION 3:**

**package** laboratory8;

**import** java.util.Comparator;

**import** java.util.PriorityQueue;

**import** java.util.Queue;

**public** **class** Bankline **implements** Comparable<Bankline> {

**private** **int** Number;

**private** **int** PersonAge;

**private** String PersonName;

**public** Bankline(**int** Number,**int** PersonAge,String PersonName) {

**this**.Number = Number;

**this**.PersonAge = PersonAge;

**this**.PersonName = PersonName;

}

@Override

**public** **int** compareTo(Bankline b) {

**return** b.Number > **this**.Number ? 1 :-1;

}

@Override

**public** String toString() {

**return** "Number:" + **this**.Number + ", PersonAge:" + **this**.PersonAge + ", PersonName:" + PersonName;

}

**public** **int** getPersonAge() {

**return** PersonAge;

}

**static** **class** PersonAgeComparator **implements** Comparator<Bankline> {

@Override

**public** **int** compare(Bankline b1, Bankline b2) {

**return** b1.getPersonAge()< b2.getPersonAge() ? 1: -1;

}

}

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

// making the objects of persons who are in the bank line

Bankline b1 = **new** Bankline(1,35,"Asadullah");

Bankline b2 = **new** Bankline(2,50,"Saifullah");

Bankline b3 = **new** Bankline(3,21,"Abdullah");

Bankline b4 = **new** Bankline(4,19,"Saad");

Bankline b5 = **new** Bankline(5,80,"Uzair");

// initializing the priority queue

Queue<Bankline> line = **new** PriorityQueue<> (**new** PersonAgeComparator());

// adding all the objects in the priority queue

line.add(b1);

line.add(b2);

line.add(b3);

line.add(b4);

line.add(b5);

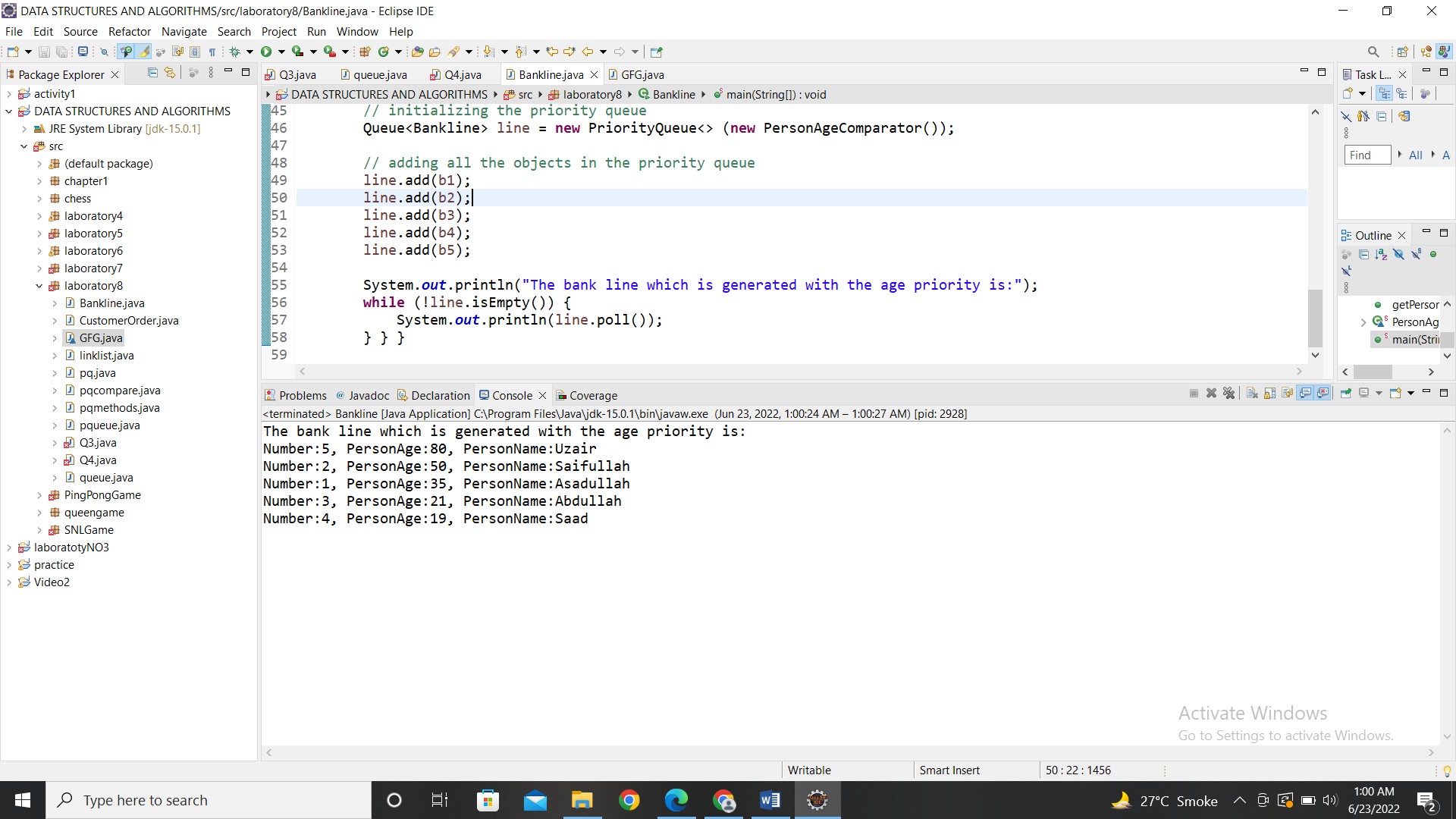
System.***out***.println("The bank line which is generated with the age priority is:");

**while** (!line.isEmpty()) {

System.***out***.println(line.poll());

} } }

Output:



**QUESTION 5:**

ALGORITHM:

**STEP 1:** Start

**STEP 2:** Initialize the queue of type Integers

**STEP 3:** Add elements into the queue by add method.

**STEP 4:** Then use the built in clear method to clear the whole queue

**STEP 5:** Stop

CODE:

**package** laboratory8;

**import** java.util.LinkedList;

**import** java.util.PriorityQueue;

**import** java.util.Queue;

**public** **class** Q5 {

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

Queue<Integer> q1 = **new** LinkedList<Integer>();

//Add elements to the Queue

q1.add(20);

q1.add(10);

q1.add(70);

q1.add(50);

q1.add(90);

System.***out***.println(q1);

q1.clear();

System.***out***.println("Queue after clearing all the elements");

System.***out***.println(q1);

Queue<Integer> q2 = **new** PriorityQueue<Integer>();

q2.add(20);

q2.add(10);

q2.add(70);

q2.add(50);

q2.add(60);

System.***out***.println(q2);

q2.clear();

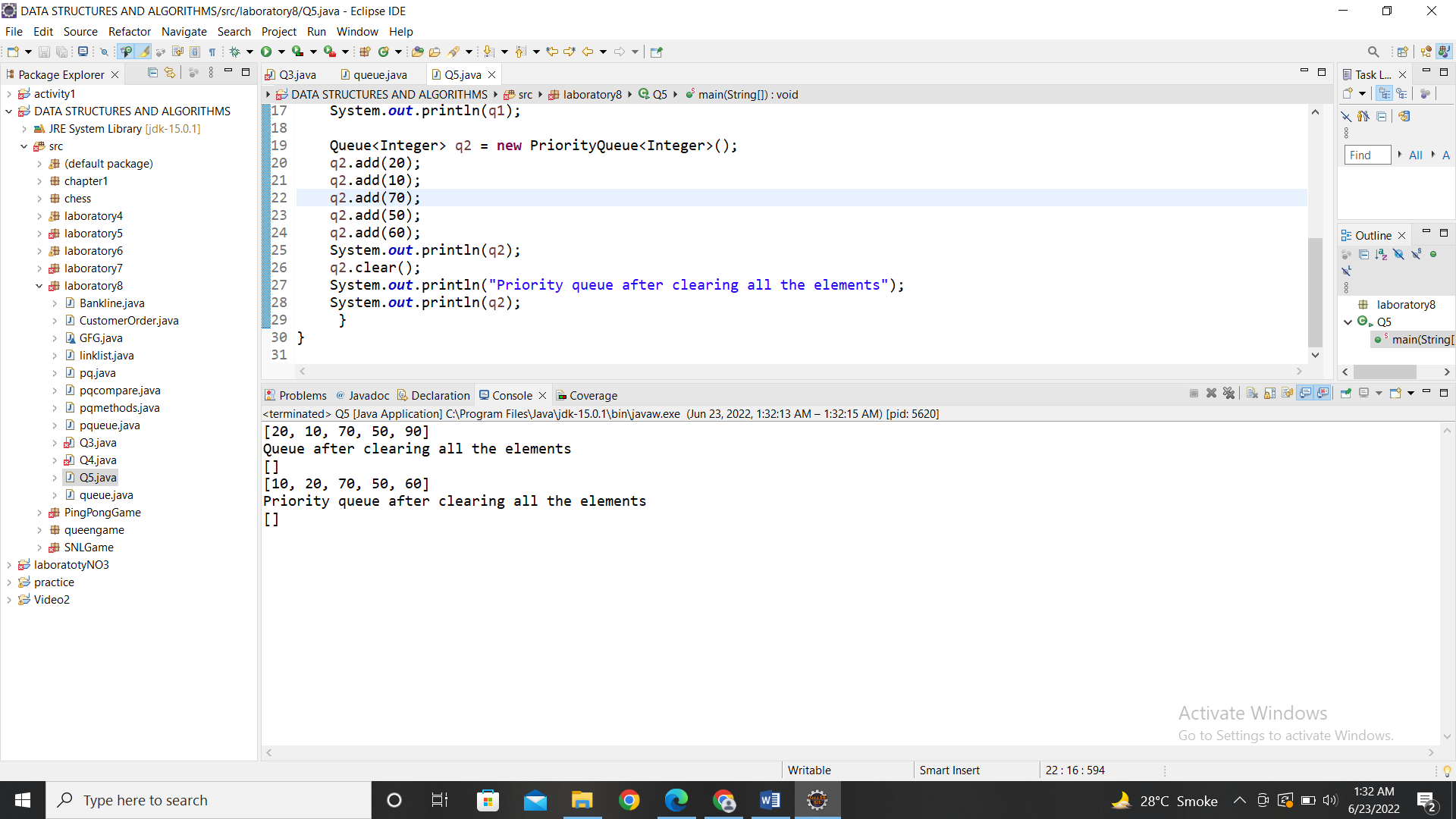
System.***out***.println("Priority queue after clearing all the elements");

System.***out***.println(q2);

}

}

OUTPUT:



**QUESTION 6:**

Operations performed on queues and priority queues are:

Enqueue operation is use to add the eleemnts into the queue.

Dequeue operation is used to remove elements from the queue.

isEmpty operation checks if the queue is empty or not.

isFull operation checks if the queue is full or not

peek operation is used to give the first element of the queue.

Poll operation is used to remove and return the first element of the queue.

Clear operation is used to clear all the elements from the queue.