LAB 8

PRACTICE QUESTIONS

Implementing the LinkedList Class;

package PRACTICE;

import java.util.Queue;

import java.util.LinkedList;

public class EX8\_1 {

public static void main(String[] args) {

// Creating Queue using the LinkedList class

Queue<Integer> queue = new LinkedList<>();

// offer elements to the Queue

queue.offer(1);

queue.offer(2);

queue.offer(3);

System.out.println("Queue: " + queue);

// Access elements of the Queue

int accessednumber = queue.peek();

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

// Remove elements from the Queue

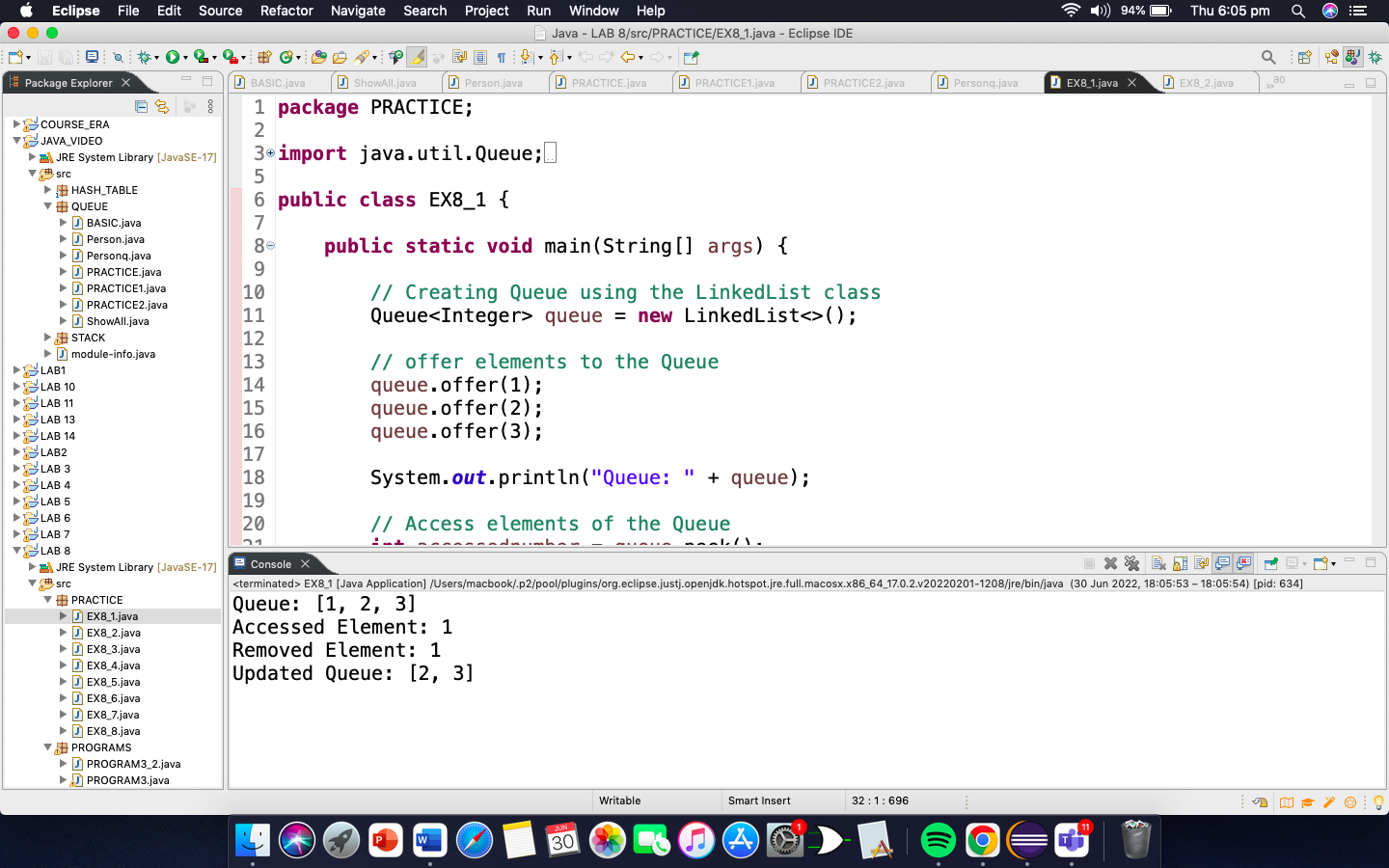
int removedNumber = queue.poll();

System.out.println("Removed Element: " + removedNumber);

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

}

}



Implementing the Queues using Array:

**package** PRACTICE;

**public** **class** EX8\_2 {

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

**private** **int** front; //fronts points to the front element of the queue

**private** **int** rear; //rear points to the last element of the queue

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

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

// Constructor to initialize a queue

EX8\_2(**int** size){

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

capacity = size;

front = 0;

rear = -1;

count = 0;

}

// Utility function to dequeue the front element

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

// check for queue underflow

**if**(isEmpty()) {

System.***out***.println("Underflow\nProgram 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) {

// check for queue overflow

**if**(isFull()) {

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

System.*exit*(-1);

}

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

rear = (rear + 1) % capacity;

arr[rear] = item;

count++;

}

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

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

**if**(isEmpty()) {

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

System.*exit*(-1);

}

**return** arr[front];

}

// 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);

}

}

**class** Main{

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

// create a queue of capacity 5

EX8\_2 queue = **new** EX8\_2(5);

queue.enqueue(1);

queue.enqueue(2);

queue.enqueue(3);

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

queue.dequeue();

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

System.***out***.println("The queue size is " + queue.size());

queue.dequeue();

queue.dequeue();

**if** (queue.isEmpty()) {

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

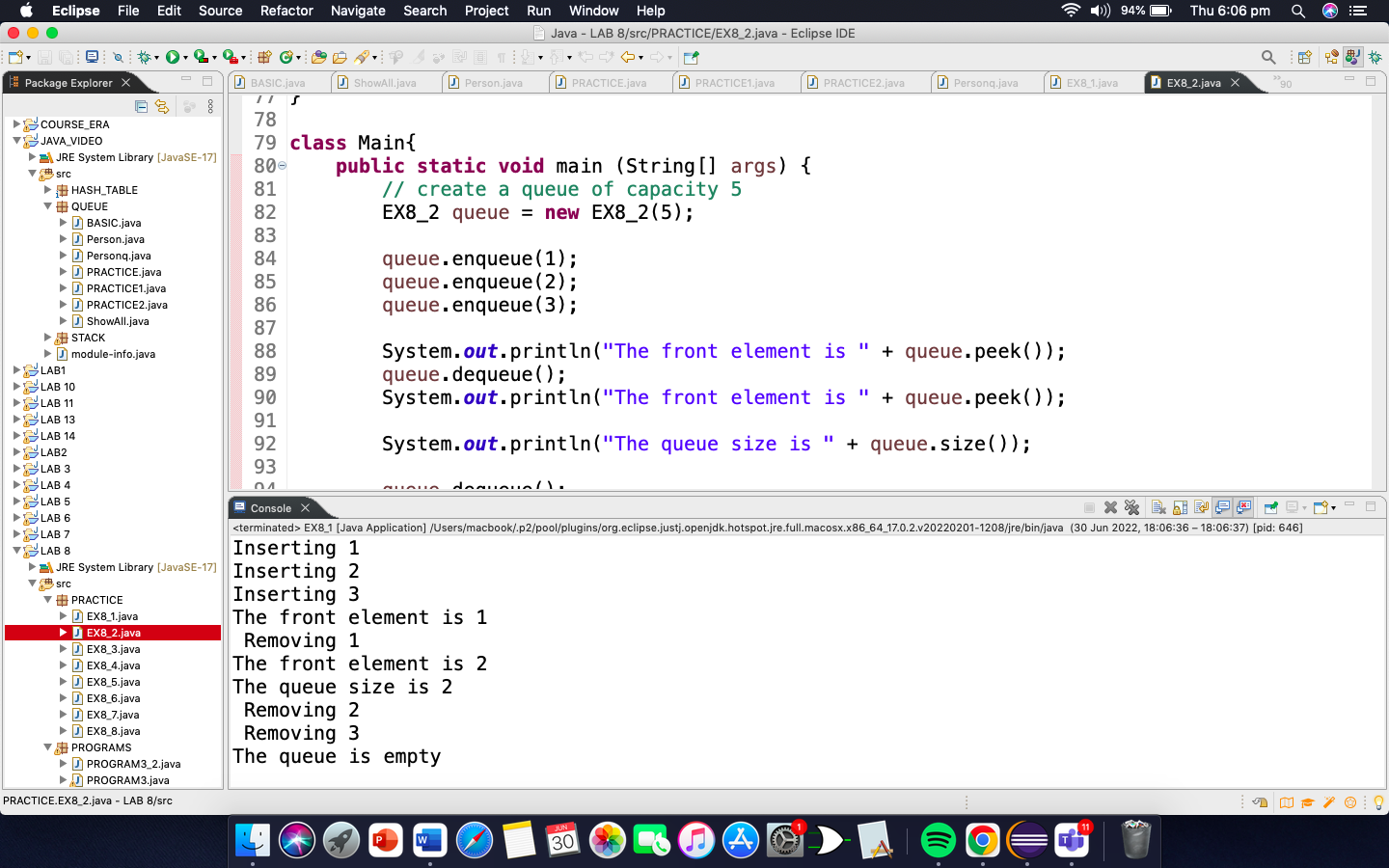
}

**else** {

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

}

}



Implementing the PriorityQueue Class:

package PRACTICE;

import java.util.Queue;

import java.util.PriorityQueue;

public class EX8\_3 {

public static void main(String[] args) {

// Creating Queue using the PriorityQueue class

Queue<Integer> queue = new PriorityQueue<>();

// offer elements to the Queue

queue.offer(5);

queue.offer(1);

queue.offer(2);

// Access elements of the Queue

int accessedNumber = queue.peek();

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

// Remove elements from the Queue

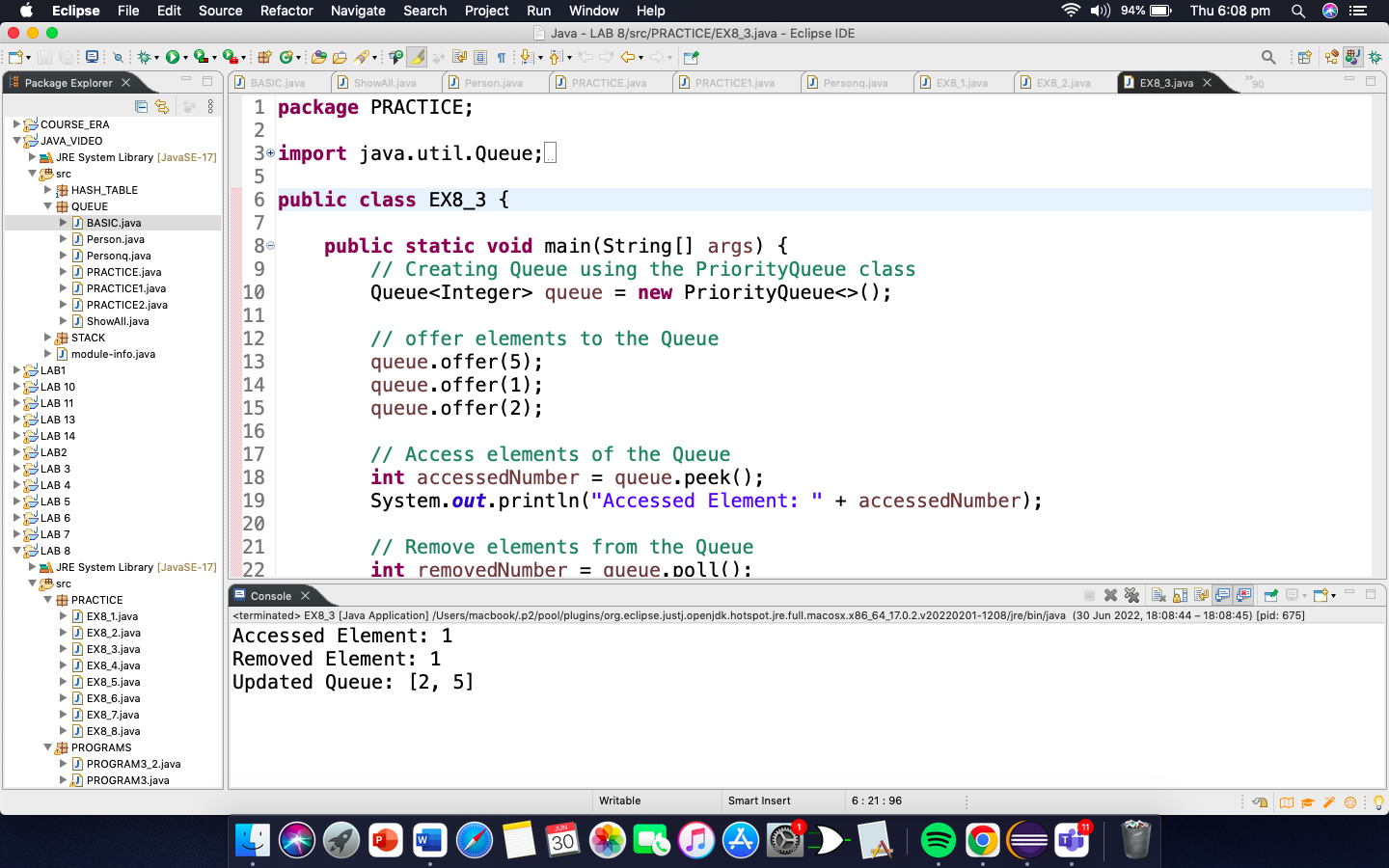
int removedNumber = queue.poll();

System.out.println("Removed Element: " + removedNumber);

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

}

}



Insert Elements to PriorityQueue:

**package** PRACTICE;

**import** java.util.PriorityQueue;

**public** **class** EX8\_4 {

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

//Creating a priority Queue

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

//using the add() method

queue.add(4);

queue.add(2);

System.***out***.println("Priority Queue " + queue);

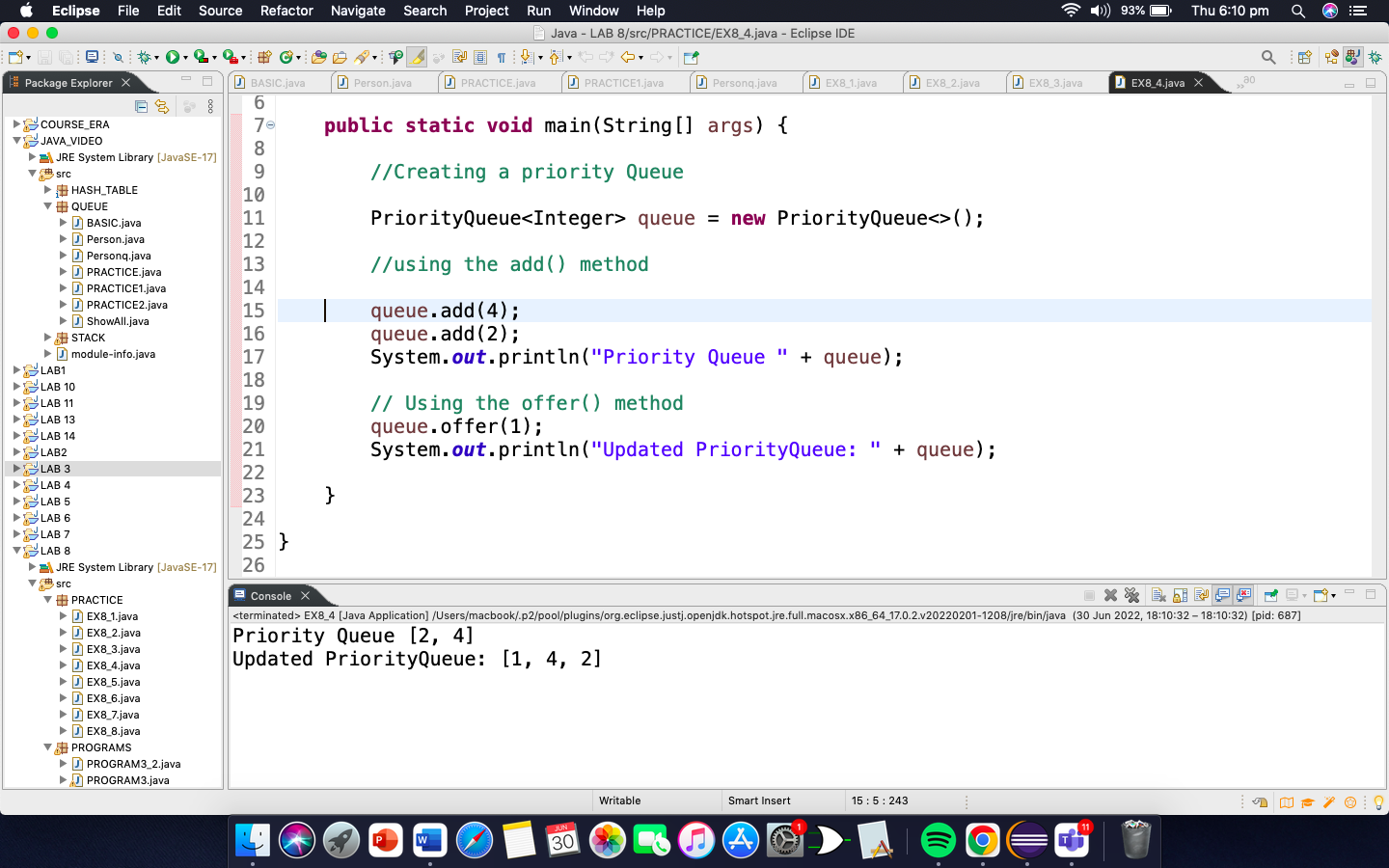
// Using the offer() method

queue.offer(1);

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

}

}



Access PriorityQueue Elements:

**package** PRACTICE;

**import** java.util.PriorityQueue;

**public** **class** EX8\_5 {

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

//Creating a priority Queue

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

//using the add() method

queue.add(4);

queue.add(2);

queue.add(1);

System.***out***.println("Priority Queue " + queue);

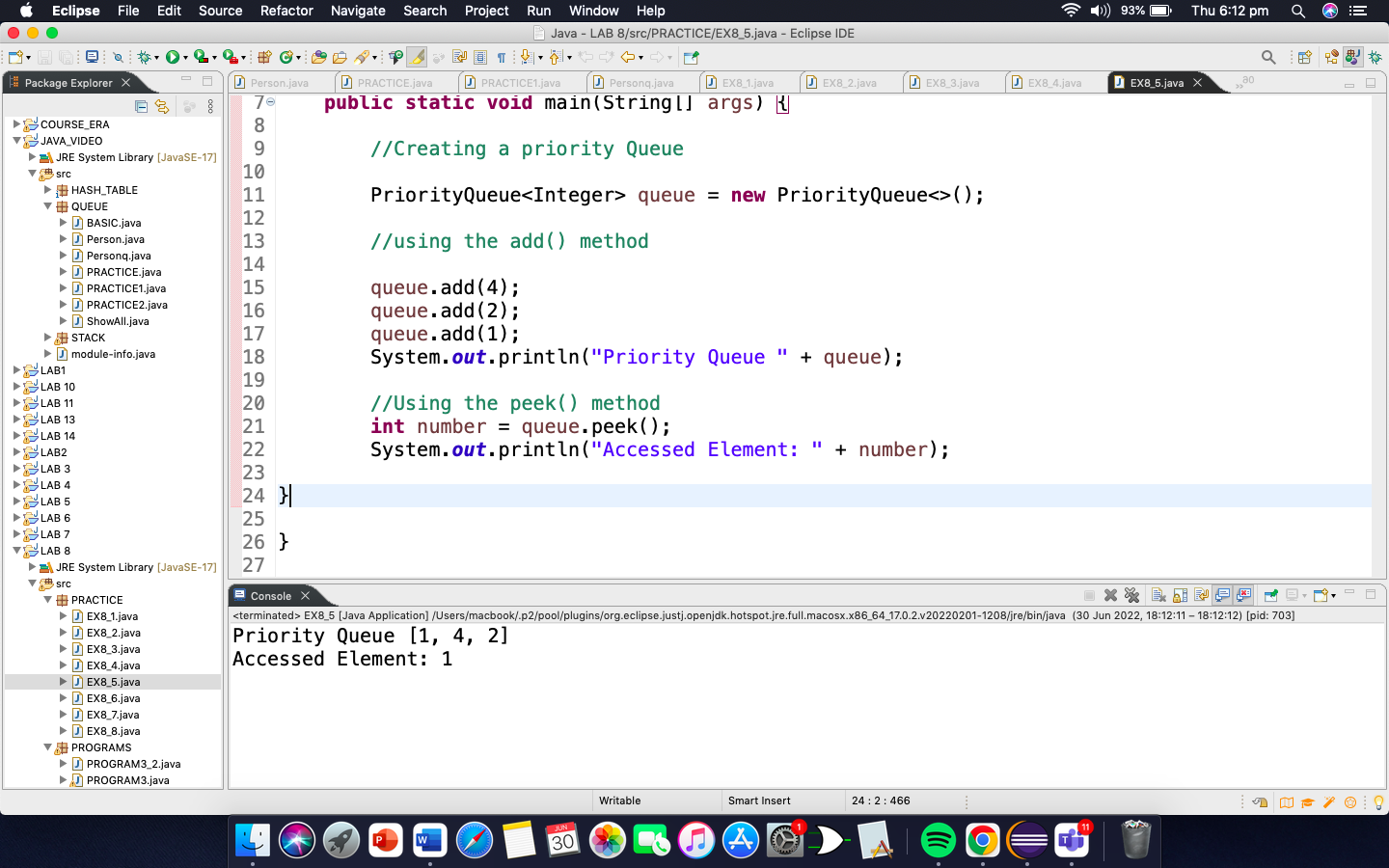
//Using the peek() method

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

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

}

}



EXERCISE 8.6

**package** PRACTICE;

**import** java.util.PriorityQueue;

**public** **class** EX8\_6 {

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

//Creating a priority Queue

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

//using the add() method

queue.add(4);

queue.add(2);

queue.add(1);

System.***out***.println("Priority Queue " + queue);

//Using the remove() method

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

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

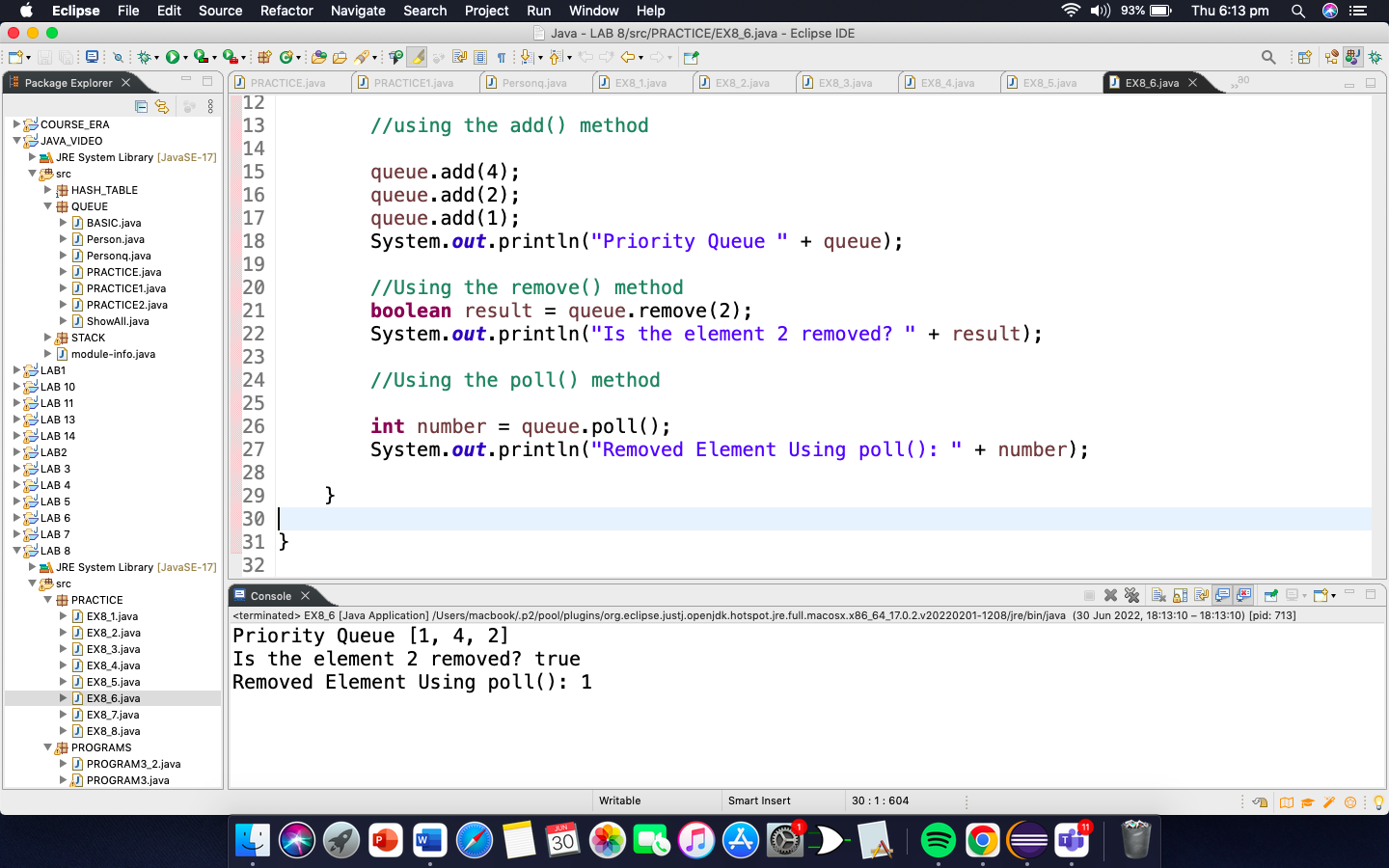
//Using the poll() method

**int** number = queue.poll();

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

}

}



EXERCISE 8.7

package PRACTICE;

import java.util.PriorityQueue;

import java.util.Iterator;

public class EX8\_7 {

public static void main(String[] args) {

//Creating a priority Queue

PriorityQueue<Integer> queue = new PriorityQueue<>();

//using the add() method

queue.add(4);

queue.add(2);

queue.add(1);

System.out.println("Priority Queue " + queue);

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

//Using the iterator() method

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

while(iterate.hasNext()) {

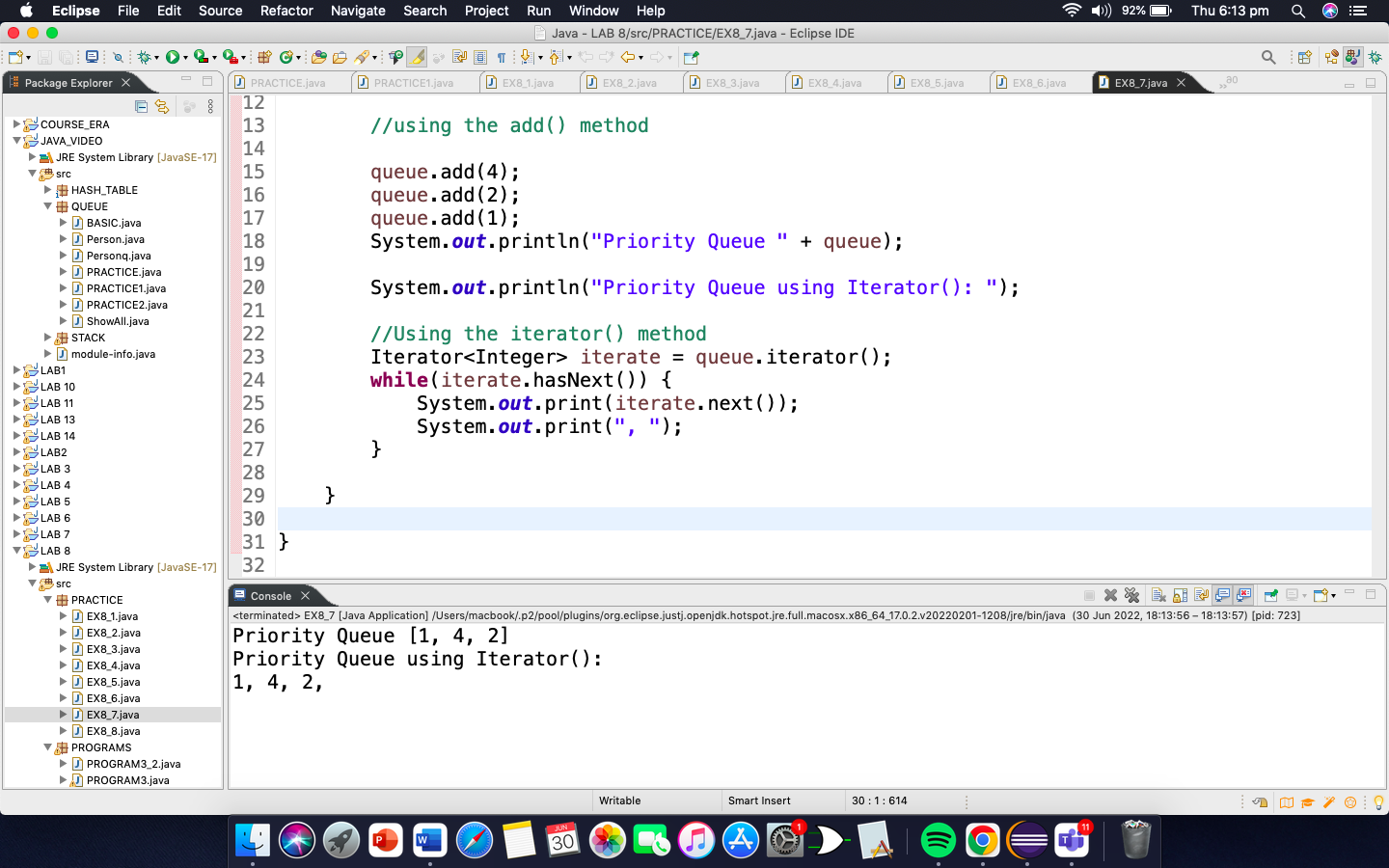
System.out.print(iterate.next());

System.out.print(", ");

}

}

}



Iterating Over a PriorityQueue:

package PRACTICE;

import java.util.PriorityQueue;

import java.util.Comparator;

public class EX8\_8 {

public static void main(String[] args) {

//Creating a priority Queue

PriorityQueue<Integer> queue = new PriorityQueue<>();

//using the add() method

queue.add(4);

queue.add(2);

queue.add(1);

queue.add(3);

System.out.println("Priority Queue " + queue);

}

}

class CustomComparator implements Comparator<Integer> {

@Override

public int compare(Integer number1, Integer number2) {

int value = number1.compareTo(number2);

//element are sorted in reverse order

if(value > 0) {

return -1;

}

else if(value < 0) {

return 1;

}

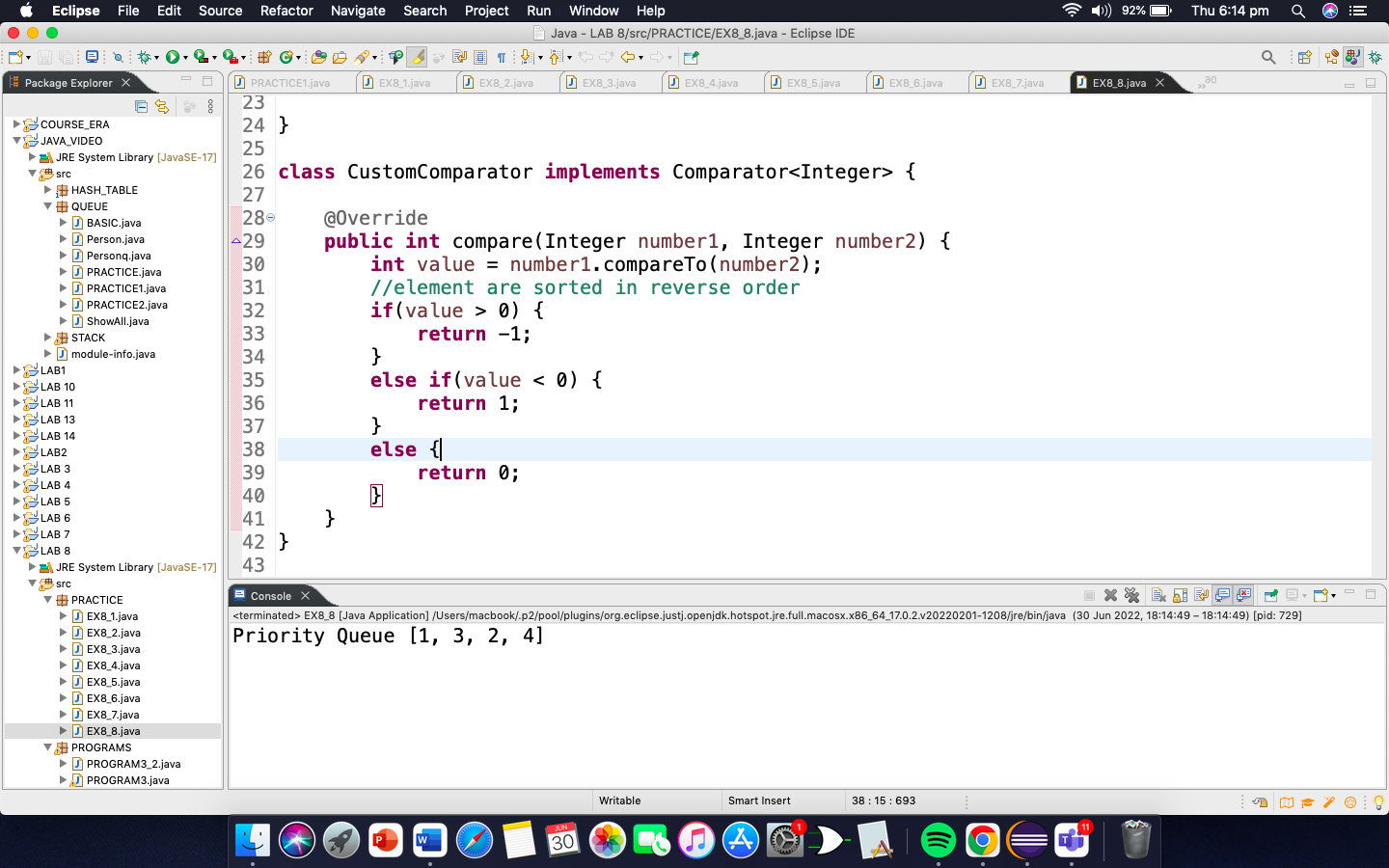
else {

return 0;

}

}

}



PROGRAMMING EXERCISE

1. Write the application in which you think queues can be used?

Queues can be used as for the application of ticket counter.

1. Write the code of the application you have discussed in question 1.

**package** QUEUE;

**public** **class** PRACTICE2 {

**private** String name;

**private** String customerNo;

**private** **int** Ticket;

**public** PRACTICE2(String name, String customerNo, **int** Ticket) {

**this**.name = name;

**this**.customerNo = customerNo;

**this**.Ticket = Ticket;

}

@Override

**public** String toString() {

**return** "[name=" + name + ", customerNo=" + customerNo + ", Ticket=" + Ticket + "]";

}

}

**package** QUEUE;

**public** **class** PRACTICE {

**private** **int**[] q;

**private** **int** total;

**private** **int** size;

**private** **int** front;

**private** **int** rear;

**public** PRACTICE() {

size = 100;

total = 0;

front = 0;

rear = 0;

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

}

**public** **boolean** enqueue(**int** item) {

**if**(isFull()) {

**return** **false**;

}**else** {

total++;

q[rear] = item;

rear++;

**return** **true**;

}

}

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

**int** item = q[front];

total--;

front++;

**return** item;

}

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

**return** (total == size);

}

}

**class** Mainss{

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

PRACTICE queue = **new** PRACTICE();

queue.enqueue(1290);

queue.enqueue(1291);

queue.enqueue(1292);

queue.enqueue(1293);

}

}

**package** QUEUE;

**public** **class** PRACTICE1 {

**private** PRACTICE2[] q;

**private** **int** total;

**private** **int** size;

**private** **int** front;

**private** **int** rear;

**public** PRACTICE1() {

size = 100;

total = 0;

front = 0;

rear = 0;

q = **new** PRACTICE2[size];

}

**public** PRACTICE1(**int** size) {

**this**.size = 100;

total = 0;

front = 0;

rear = 0;

q = **new** PRACTICE2[**this**.size];

}

**public** **boolean** enqueue(PRACTICE2 item) {

**if**(isFull()) {

**return** **false**;

}**else** {

total++;

q[rear] = item;

rear = (rear + 1) % size;

**return** **true**;

}

}

**public** PRACTICE2 dequeue() {

PRACTICE2 item = q[front];

total--;

front = (front + 1) % size;

**return** item;

}

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

**return** (total == size);

}

**public** **void** showAll() {

**int** f = front;

**if**(total !=0) {

**for**(**int** i=0; i<total; i++) {

System.***out***.println(" " + q[f].toString());

f = (f + 1) % size;

}

}

}

}

**class** main2{

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

PRACTICE queue = **new** PRACTICE();

queue.enqueue(1290);

queue.enqueue(1291);

queue.enqueue(1292);

queue.enqueue(1293);

PRACTICE2 p1 = **new** PRACTICE2("ahmad", "123", queue.dequeue() );

PRACTICE2 p2 = **new** PRACTICE2("zuhair", "124", queue.dequeue());

PRACTICE2 p3 = **new** PRACTICE2("mashood", "125", queue.dequeue());

PRACTICE2 p4 = **new** PRACTICE2("izan", "126", queue.dequeue());

PRACTICE1 queue1 = **new** PRACTICE1();

queue1.enqueue(p1);

queue1.enqueue(p2);

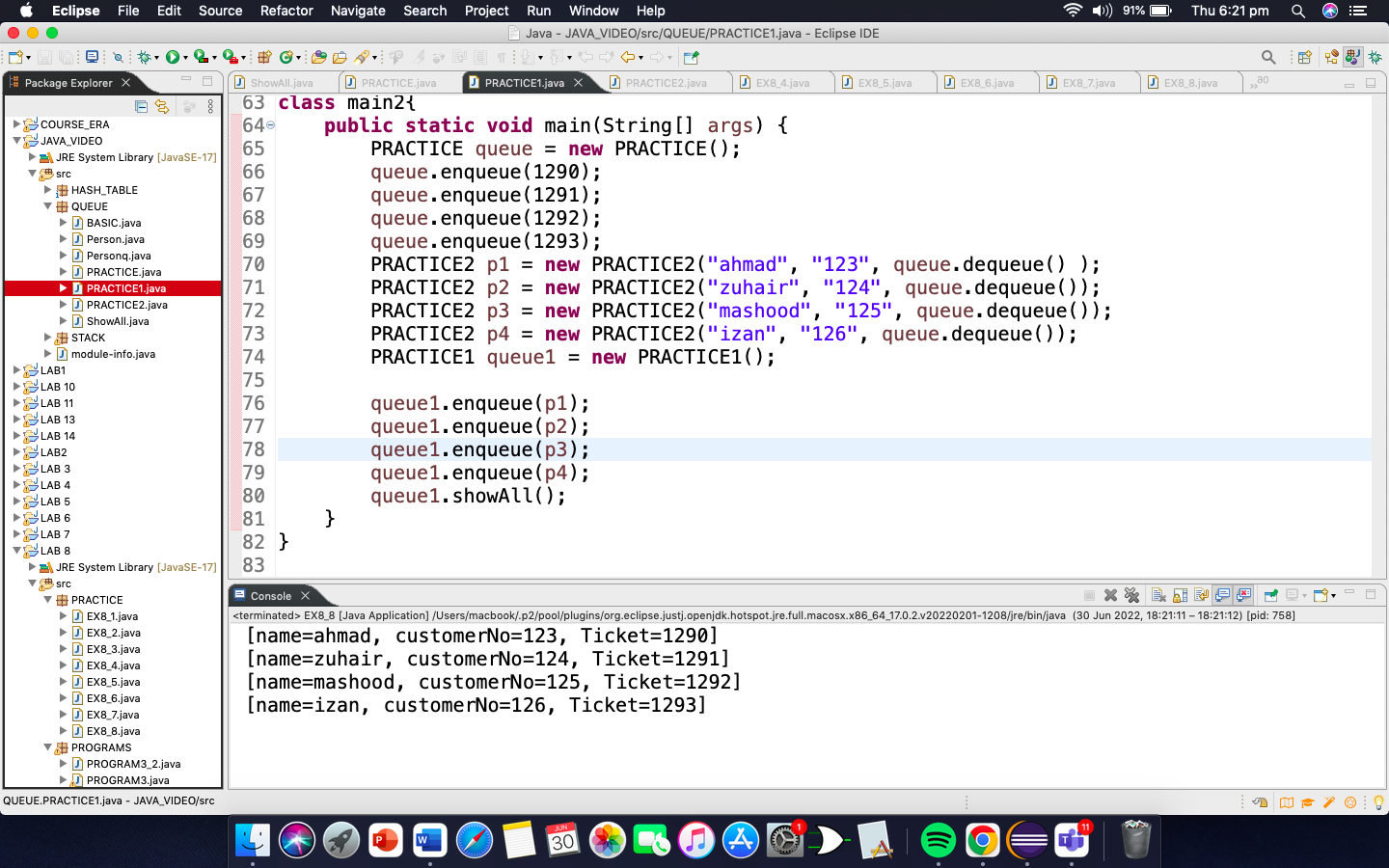
queue1.enqueue(p3);

queue1.enqueue(p4);

queue1.showAll();

}

}



1. In bank when old aged people come, no matter the token number you can give them priority how can you managed it using priority queues. Code the application.

**package** PROGRAMS;

**public** **class** PROGRAM3\_2 {

**private** **int**[] q;

**private** **int** total;

**private** **int** size;

**private** **int** front;

**private** **int** rear;

**public** PROGRAM3\_2() {

size = 100;

total = 0;

front = 0;

rear = 0;

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

}

**public** **boolean** enqueue(**int** item) {

**if**(isFull()) {

**return** **false**;

}**else** {

total++;

q[rear] = item;

rear++;

**return** **true**;

}

}

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

**int** item = q[front];

total--;

front++;

**return** item;

}

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

**return** (total == size);

}

}

package PROGRAMS;

import java.util.Collections;

import java.util.PriorityQueue;

import java.util.Scanner;

public class PROGRAM3 {

public static void main(String[] args) {

PriorityQueue<Integer> queue = new PriorityQueue<>(Collections.reverseOrder());

PROGRAM3\_2 queue1 = new PROGRAM3\_2();

Scanner input = new Scanner(System.in);

for(int i=0; i<=3; i++ ) {

System.out.println("Enter your age");

int q = input.nextInt();

queue.add(q);

}

queue1.enqueue(1);

queue1.enqueue(2);

queue1.enqueue(3);

queue1.enqueue(4);

for(int i=0; i<=3; i++ ) {

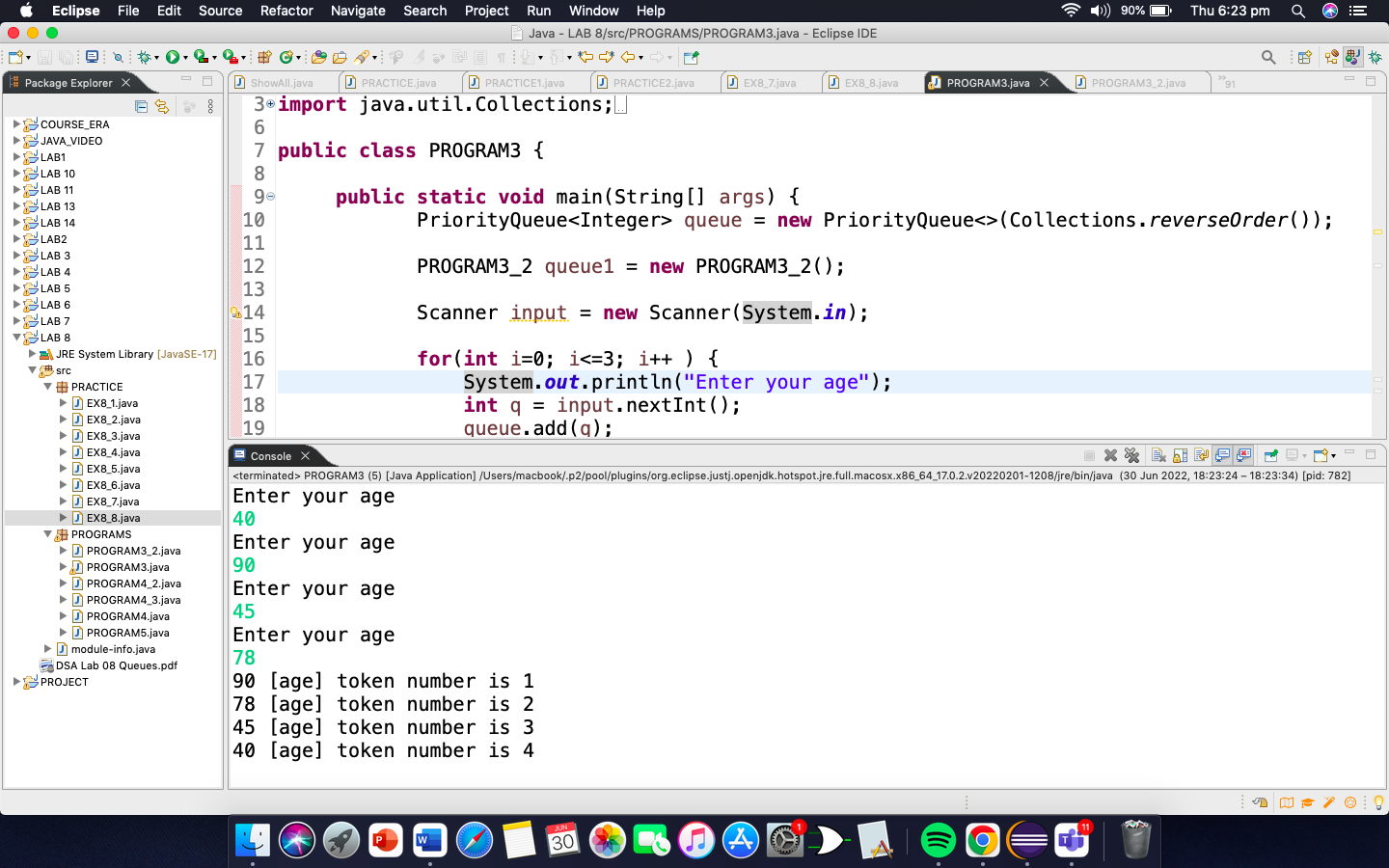
System.out.print(queue.poll() + " [age]");

System.out.println(" token number is " + queue1.dequeue());

}

}

}



1. Write an algorithm and then program that can show the number of users who can use printer for their printing After certain time a manager can access for printing his priority is high how can you manage this application using priority queue.

**package** PROGRAMS;

**public** **class** PROGRAM4 {

**public** String Designation;

**public** **int** rank;

**public** PROGRAM4(String Designation, **int** rank) {

**this**.Designation = Designation;

**this**.rank = rank;

}

**public** Object getName() {

**return** Designation + " rank number is " + rank;

}

}

**package** PROGRAMS;

**import** java.util.Comparator;

**public** **class** PROGRAM4\_3 **implements** Comparator<PROGRAM4>{

**public** **int** compare(PROGRAM4 s1, PROGRAM4 s2) {

**if** (s1.rank > s2.rank)

**return** 1;

**else** **if** (s1.rank < s2.rank)

**return** -1;

**return** 0;

}

}

**package** PROGRAMS;

**import** java.util.PriorityQueue;

**public** **class** PROGRAM4\_2 {

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

PriorityQueue<PROGRAM4> pq = **new**

PriorityQueue<PROGRAM4>(5, **new** PROGRAM4\_3());

PROGRAM4 p1 = **new** PROGRAM4("Manager", 2);

pq.add(p1);

PROGRAM4 p2 = **new** PROGRAM4("Employee", 4);

pq.add(p2);

PROGRAM4 p3 = **new** PROGRAM4("Employee" ,5);

pq.add(p3);

PROGRAM4 p4 = **new** PROGRAM4("Employee" ,6);

pq.add(p4);

PROGRAM4 p5 = **new** PROGRAM4("helper", 8);

pq.add(p5);

System.***out***.println("Printer used by any person in the firm in their priority order");

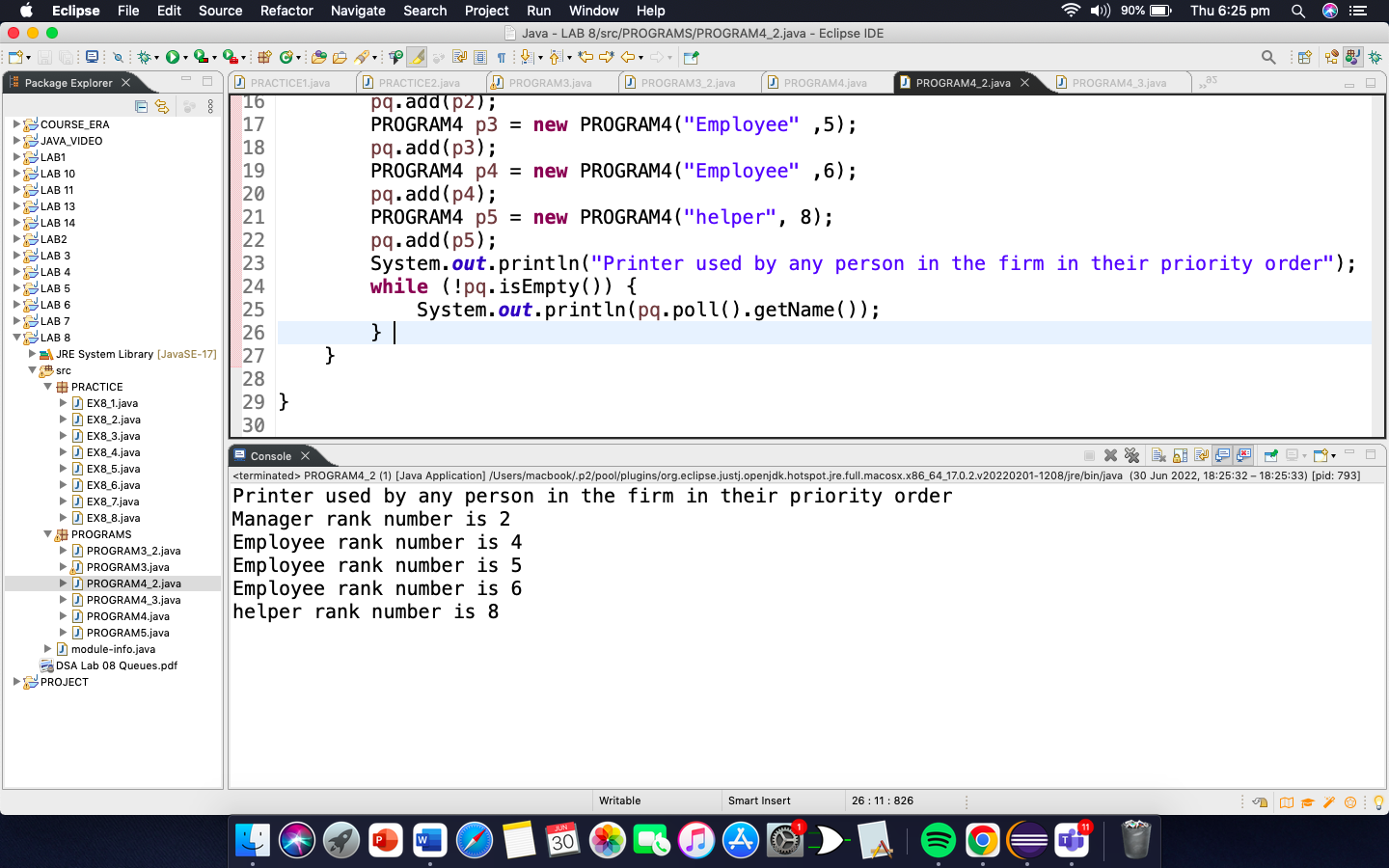
**while** (!pq.isEmpty()) {

System.***out***.println(pq.poll().getName());

}

}

}



1. Write an algorithm and then program to clear all the elements from the queues, and priority queues

**package** PROGRAMS;

**public** **class** PROGRAM5 {

**private** **int**[] arr;

**private** **int** front;

**private** **int** rear;

**private** **int** capacity;

**private** **int** count;

PROGRAM5(**int** size){

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

capacity = size;

front = 0;

rear = -1;

count = 0;

}

// Utility function to dequeue the front element

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

// check for queue underflow

**if**(isEmpty()) {

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

System.*exit*(-1);

}

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

front = (front + 1) % capacity;

count--;

**return** x;

}

// Utility function to add an item to the queue

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

// check for queue overflow

**if**(isFull()) {

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

System.*exit*(-1);

}

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

rear = (rear + 1) % capacity;

arr[rear] = item;

count++;

}

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

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

**return** (count == 0);

}

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

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

**return** (count == capacity);

}

}

**class** Main{

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

// create a queue of capacity 5

PROGRAM5 queue = **new** PROGRAM5(5);

queue.enqueue(1);

queue.enqueue(2);

queue.enqueue(3);

System.***out***.println("Clearing queue");

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

queue.dequeue();

}

**if** (queue.isEmpty()) {

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

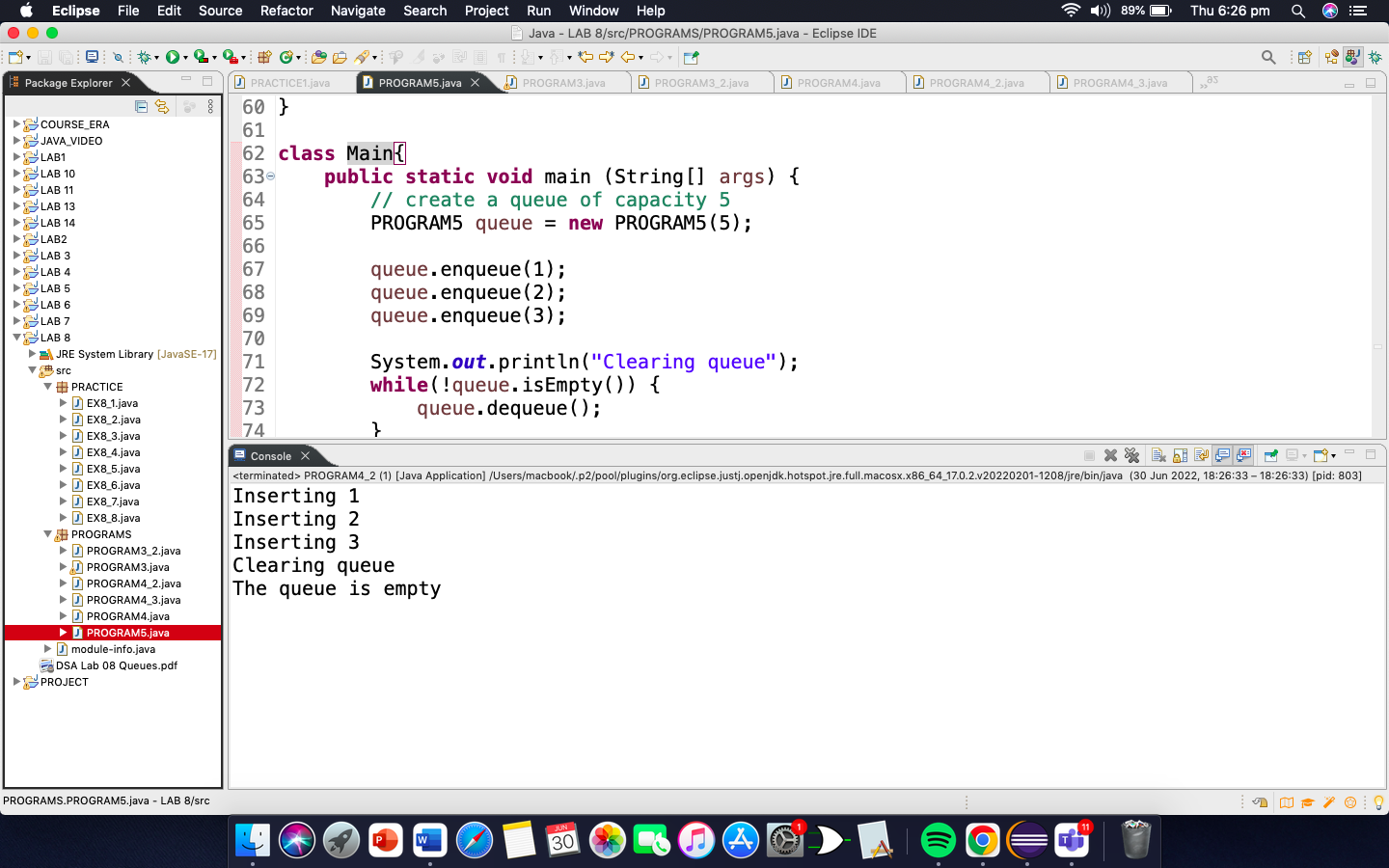
}

**else** {

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

}

}



1. Discuss in details the operations you can perform on queues and priority queues

Queue operations may involve initializing or defining the queue, utilizing it, and then completely erasing it from the memory. Here we shall try to understand the basic operations associated with queues −

* **enqueue()** − add (store) an item to the queue.
* **dequeue()** − remove (access) an item from the queue.

Few more functions are required to make the above-mentioned queue operation efficient. These are −

* **peek()** − Gets the element at the front of the queue without removing it.
* **isfull()** − Checks if the queue is full.
* **isempty()** − Checks if the queue is empty.

In queue, we always dequeue (or access) data, pointed by **front** pointer and while enqueing (or storing) data in the queue we take help of **rear** pointer.