Trường Đại Học Quốc Tế - ĐHQG TP.HCM

LAB REPORT

Course: Algorithms & Data Structures LAB 3

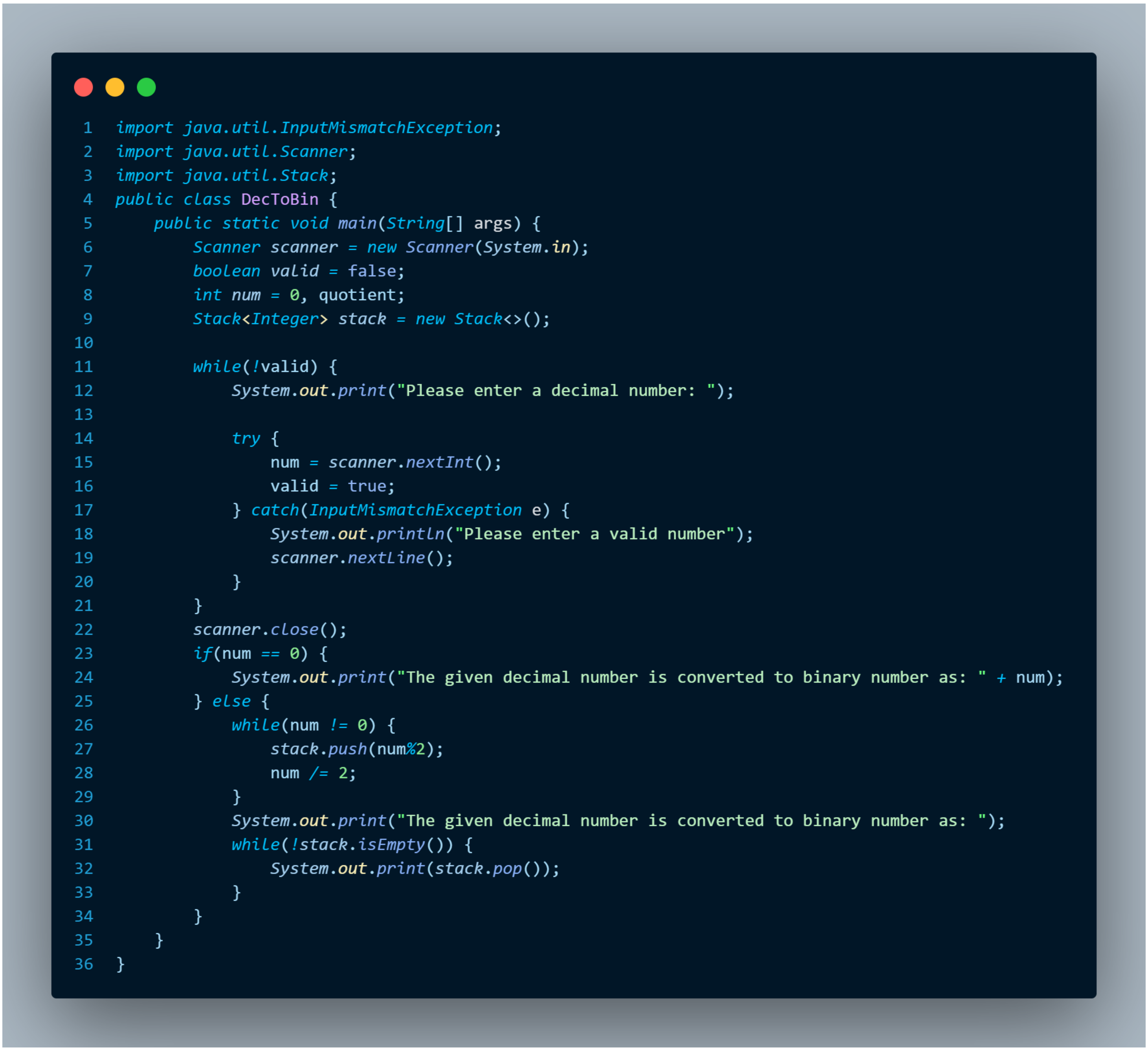
Full Name: Trần Minh Phúc . . . .

Student's ID: ITCSIU24070 . . . .

1 Problem 1

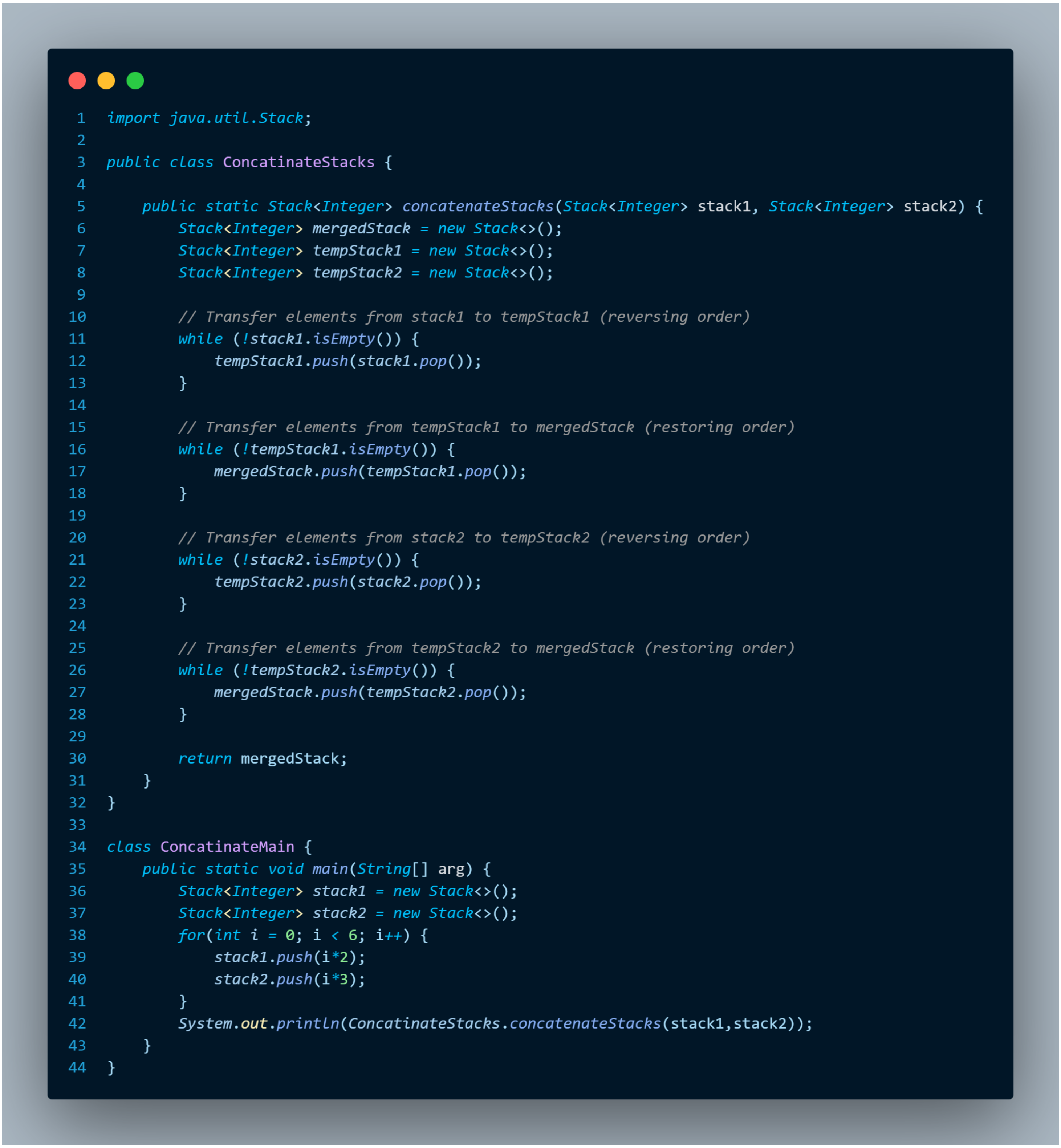
Write a program to

1.1 Convert a decimal number and convert it to octal form



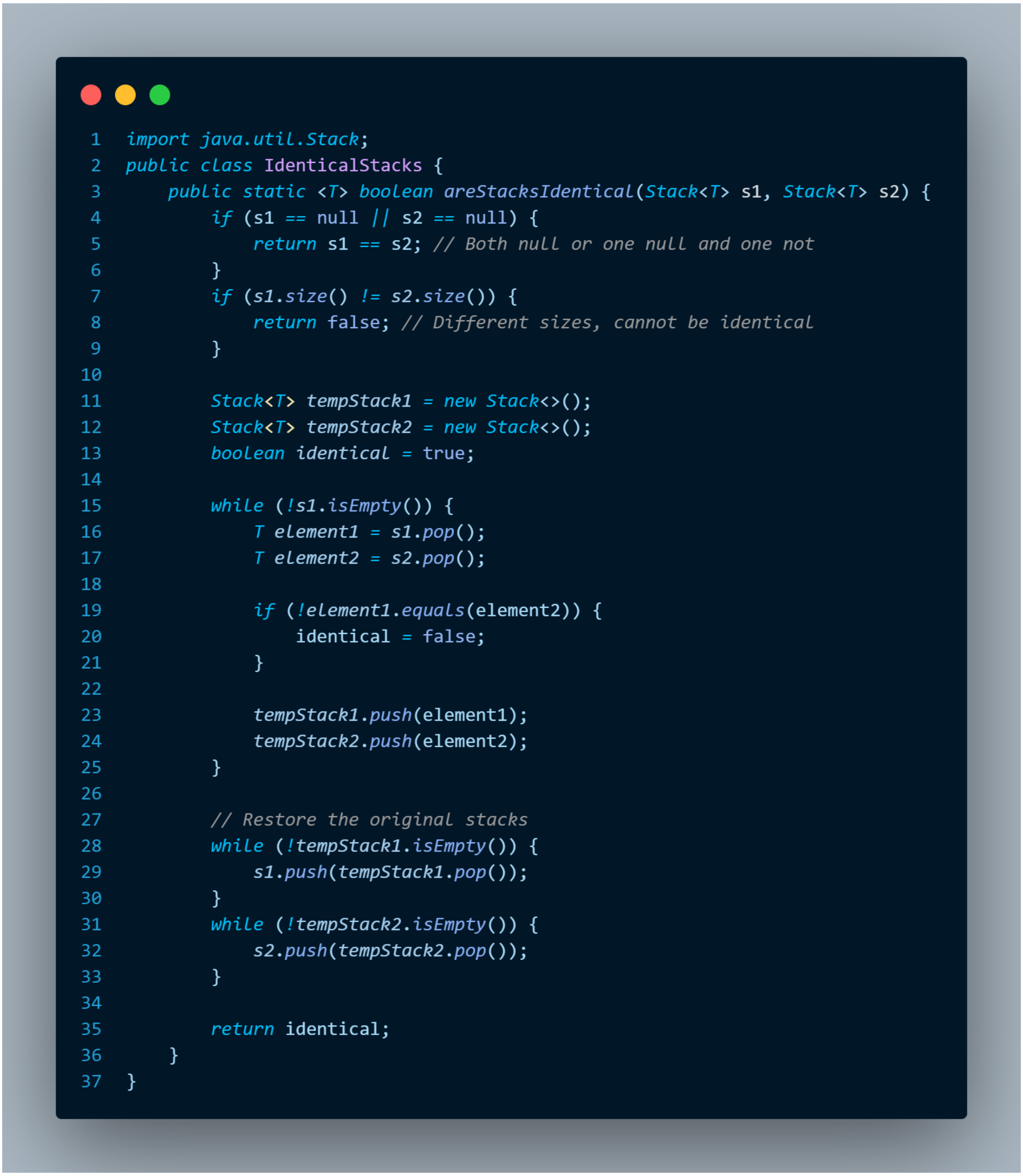
```java
import java.util.InputMismatchException;
import java.util.Scanner;
import java.util.Stack;
public class DecToBin {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        boolean valid = false;
        int num = 0, quotient;
        Stack<Integer> stack = new Stack<>();

        while(!valid) {
            System.out.print("Please enter a decimal number: ");

            try {
                num = scanner.nextInt();
                valid = true;
            } catch(InputMismatchException e) {
                System.out.println("Please enter a valid number");
                scanner.nextLine();
            }
        }
        scanner.close();
        if(num == 0) {
            System.out.print("The given decimal number is converted to binary number as: " + num);
        } else {
            while(num != 0) {
                stack.push(num%2);
                num /= 2;
            }
            System.out.print("The given decimal number is converted to binary number as: ");
            while(!stack.isEmpty()) {
                System.out.print(stack.pop());
            }
        }
    }
}
```

1.2 Concatenate two stacks



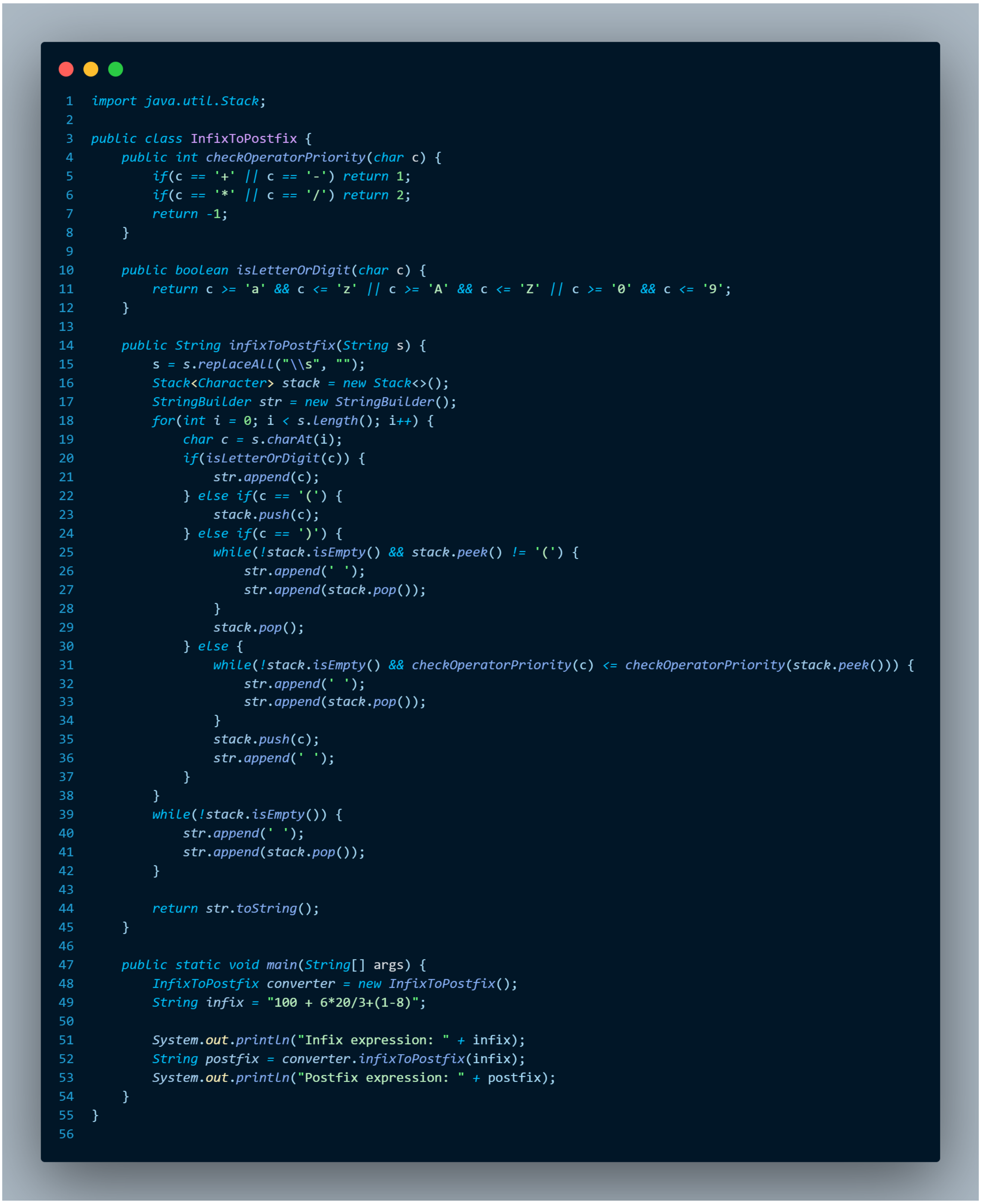
```java
import java.util.Stack;

public class ConcatinateStacks {

    public static Stack<Integer> concatenateStacks(Stack<Integer> stack1, Stack<Integer> stack2) {
        Stack<Integer> mergedStack = new Stack<>();
        Stack<Integer> tempStack1 = new Stack<>();
        Stack<Integer> tempStack2 = new Stack<>();

        // Transfer elements from stack1 to tempStack1 (reversing order)
        while (!stack1.isEmpty()) {
            tempStack1.push(stack1.pop());
        }

        // Transfer elements from tempStack1 to mergedStack (restoring order)
        while (!tempStack1.isEmpty()) {
            mergedStack.push(tempStack1.pop());
        }

        // Transfer elements from stack2 to tempStack2 (reversing order)
        while (!stack2.isEmpty()) {
            tempStack2.push(stack2.pop());
        }

        // Transfer elements from tempStack2 to mergedStack (restoring order)
        while (!tempStack2.isEmpty()) {
            mergedStack.push(tempStack2.pop());
        }

        return mergedStack;
    }
}

class ConcatinateMain {
    public static void main(String[] arg) {
        Stack<Integer> stack1 = new Stack<>();
        Stack<Integer> stack2 = new Stack<>();
        for(int i = 0; i < 6; i++) {
            stack1.push(i*2);
            stack2.push(i*3);
        }
        System.out.println(ConcatinateStacks.concatenateStacks(stack1,stack2));
    }
}
```

1.3 Determine if the contents of one stack are identical to that of another

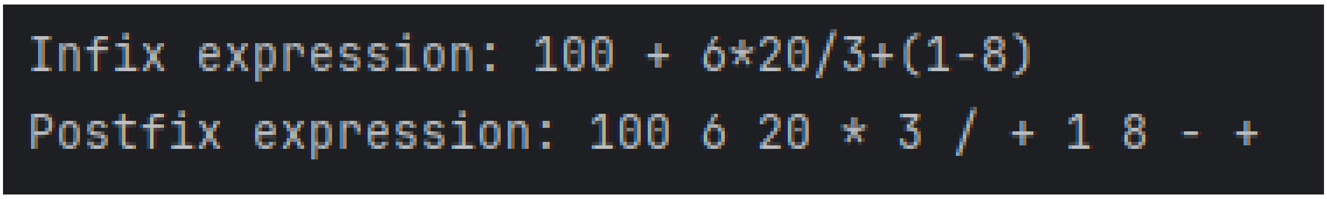


```java
import java.util.Stack;
public class IdenticalStacks {
    public static <T> boolean areStacksIdentical(Stack<T> s1, Stack<T> s2) {
        if (s1 == null || s2 == null) {
            return s1 == s2; // Both null or one null and one not
        }
        if (s1.size() != s2.size()) {
            return false; // Different sizes, cannot be identical
        }

        Stack<T> tempStack1 = new Stack<>();
        Stack<T> tempStack2 = new Stack<>();
        boolean identical = true;

        while (!s1.isEmpty()) {
            T element1 = s1.pop();
            T element2 = s2.pop();

            if (!element1.equals(element2)) {
                identical = false;
            }

            tempStack1.push(element1);
            tempStack2.push(element2);
        }

        // Restore the original stacks
        while (!tempStack1.isEmpty()) {
            s1.push(tempStack1.pop());
        }
        while (!tempStack2.isEmpty()) {
            s2.push(tempStack2.pop());
        }

        return identical;
    }
}
```

2 Problem 2



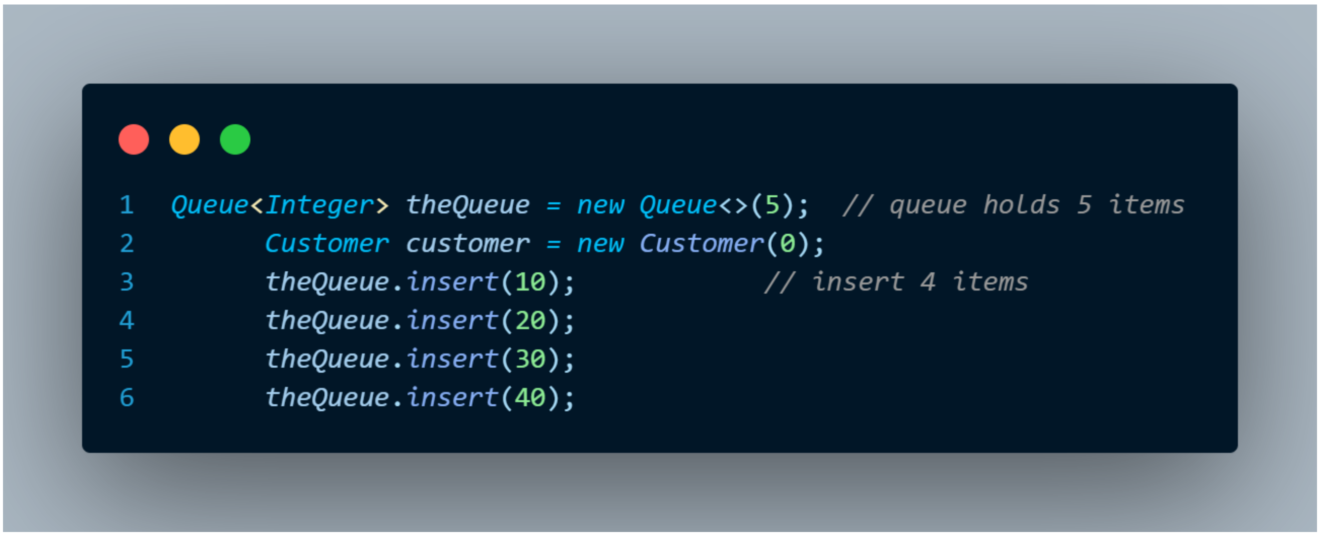
```java
import java.util.Stack;

public class InfixToPostfix {
    public int checkOperatorPriority(char c) {
        if(c == '+' || c == '-') return 1;
        if(c == '*' || c == '/') return 2;
        return -1;
    }

    public boolean isLetterOrDigit(char c) {
        return c >= 'a' && c <= 'z' || c >= 'A' && c <= 'Z' || c >= '0' && c <= '9';
    }

    public String infixToPostfix(String s) {
        s = s.replaceAll("\\s", "");
        Stack<Character> stack = new Stack<>();
        StringBuilder str = new StringBuilder();
        for(int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            if(isLetterOrDigit(c)) {
                str.append(c);
            } else if(c == '(') {
                stack.push(c);
            } else if(c == ')') {
                while(!stack.isEmpty() && stack.peek() != '(') {
                    str.append(' ');
                    str.append(stack.pop());
                }
                stack.pop();
            } else {
                while(!stack.isEmpty() && checkOperatorPriority(c) <= checkOperatorPriority(stack.peek())) {
                    str.append(' ');
                    str.append(stack.pop());
                }
                stack.push(c);
                str.append(' ');
            }
        }
        while(!stack.isEmpty()) {
            str.append(' ');
            str.append(stack.pop());
        }

        return str.toString();
    }

    public static void main(String[] args) {
        InfixToPostfix converter = new InfixToPostfix();
        String infix = "100 + 6*20/3+(1-8)";

        System.out.println("Infix expression: " + infix);
        String postfix = converter.infixToPostfix(infix);
        System.out.println("Postfix expression: " + postfix);
    }
}

```



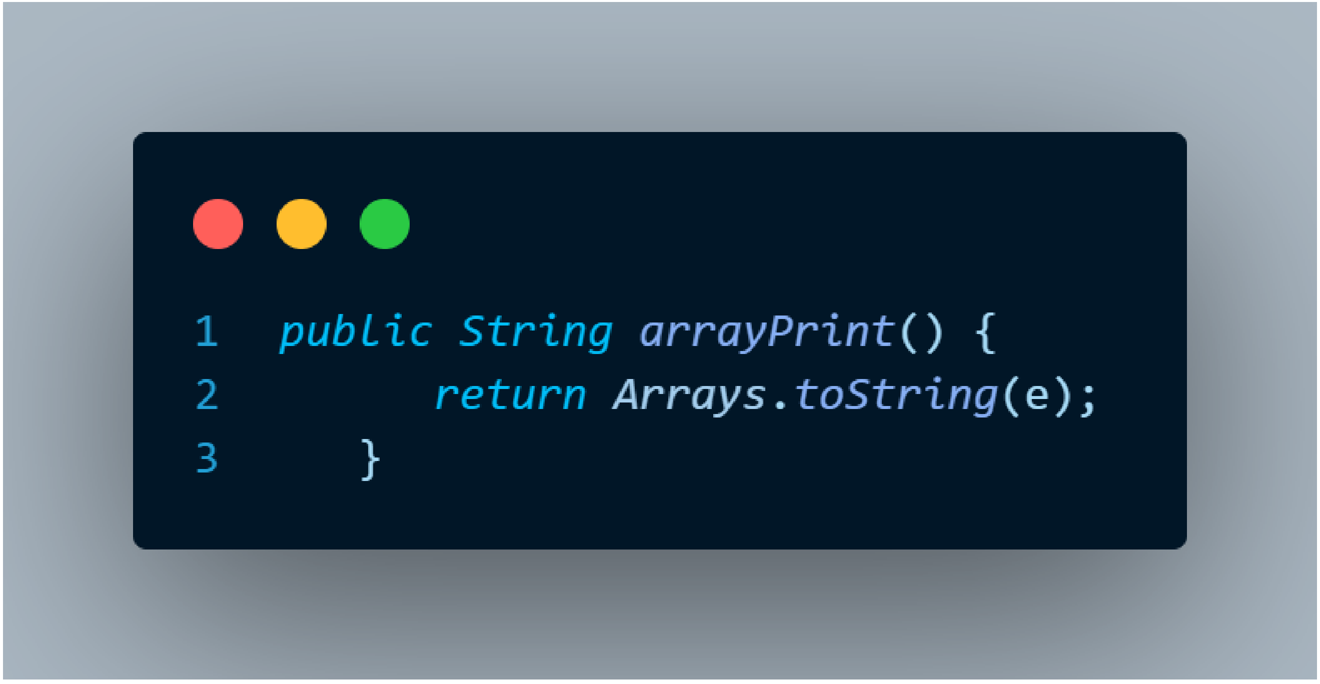
```
Infix expression: 100 + 6*20/3+(1-8)
Postfix expression: 100 6 20 * 3 / + 1 8 - +
```

3 QueueApp.java

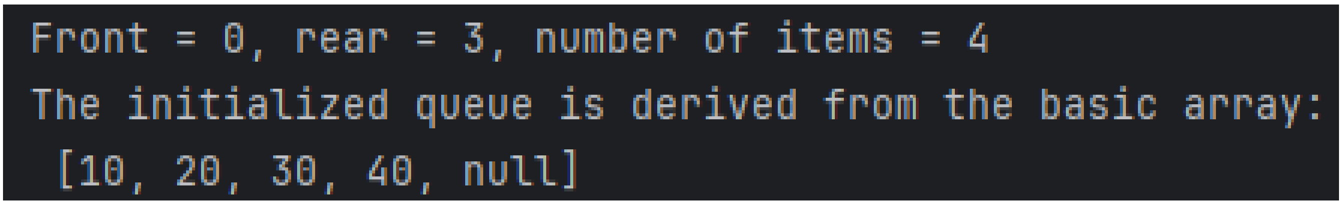
3.1 Method to display the queue array and the front and rear indices



```java
Queue<Integer> theQueue = new Queue<>(5);  // queue holds 5 items
      Customer customer = new Customer(0);
      theQueue.insert(10);              // insert 4 items
      theQueue.insert(20);
      theQueue.insert(30);
      theQueue.insert(40);
```



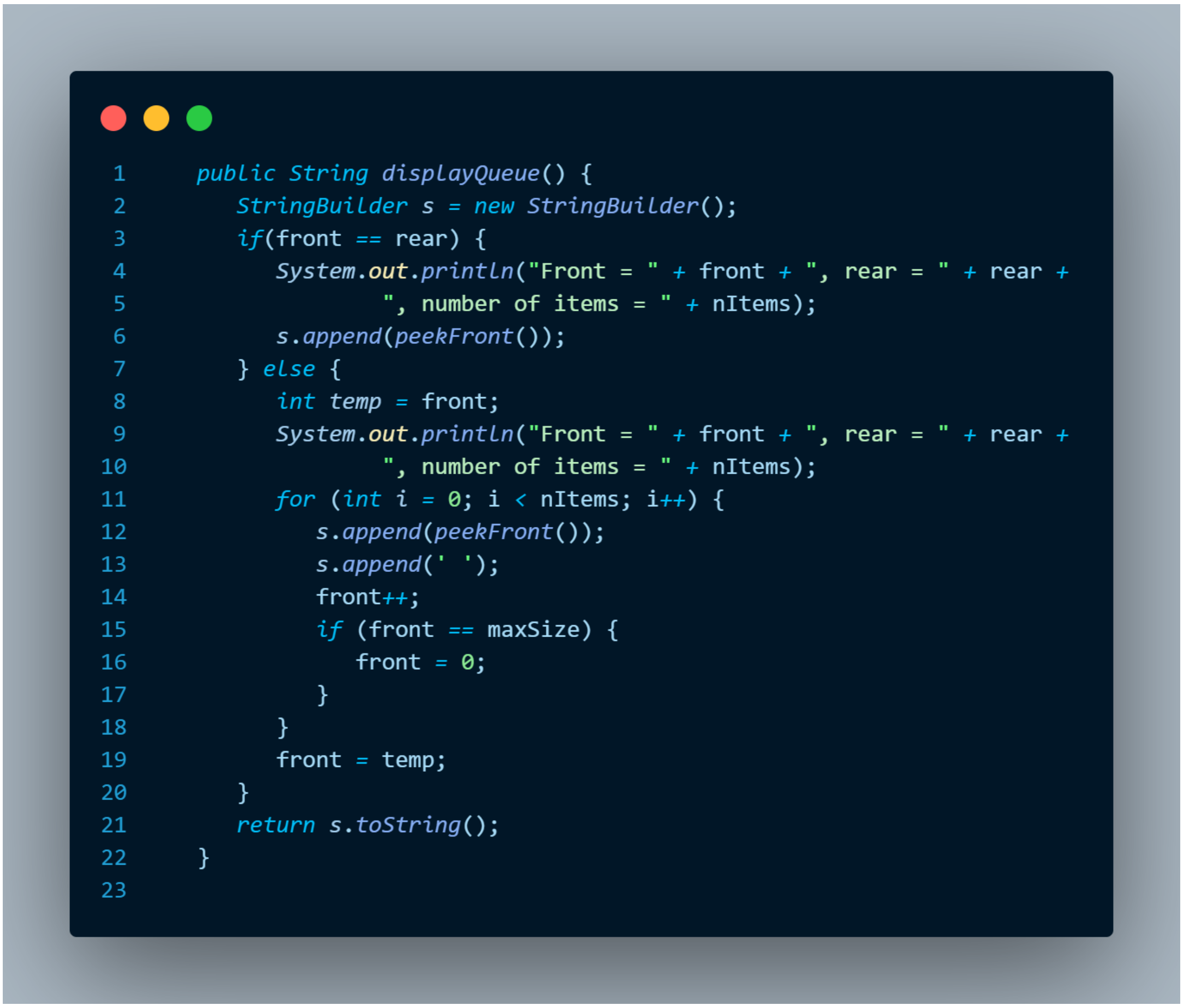
```java
public String arrayPrint() {
        return Arrays.toString(e);
    }
```



```
Front = 0, rear = 3, number of items = 4
The initialized queue is derived from the basic array:
 [10, 20, 30, 40, null]
```

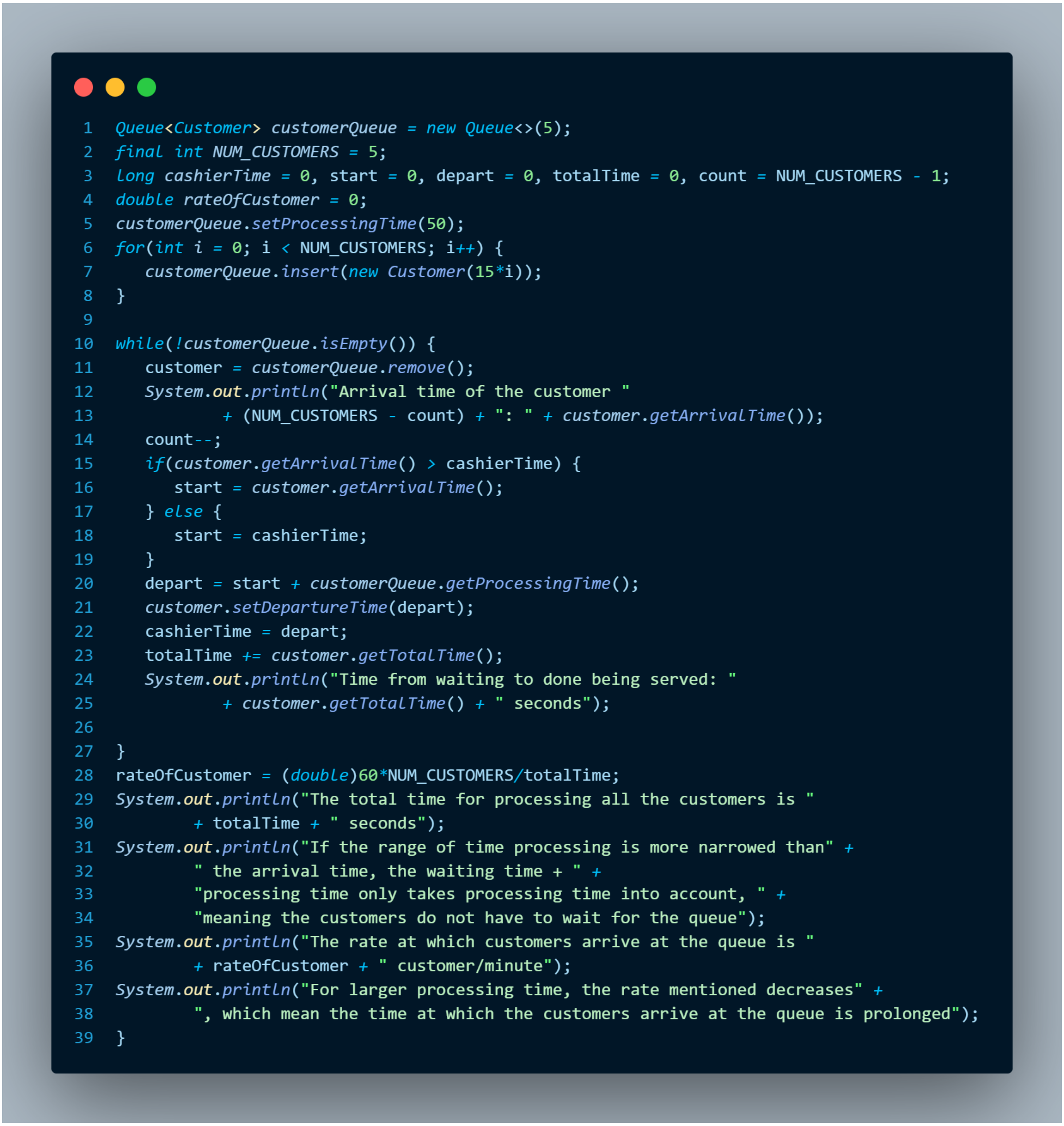
In this scenario, as the underlying array's index get to the wraparound, it is being reset so that the queue can add item at the rear.

3.2 Method to display the queue using loops

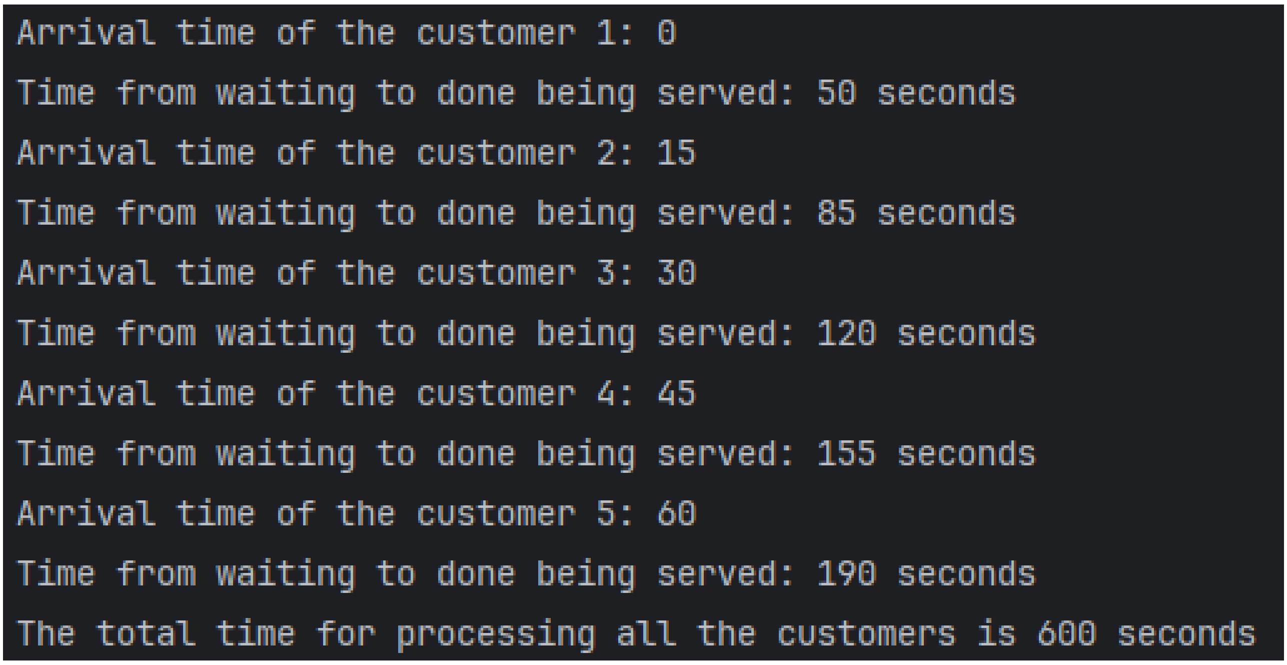


```java
public String displayQueue() {
   StringBuilder s = new StringBuilder();
   if(front == rear) {
      System.out.println("Front = " + front + ", rear = " + rear +
              ", number of items = " + nItems);
      s.append(peekFront());
   } else {
      int temp = front;
      System.out.println("Front = " + front + ", rear = " + rear +
              ", number of items = " + nItems);
      for (int i = 0; i < nItems; i++) {
         s.append(peekFront());
         s.append(' ');
         front++;
         if (front == maxSize) {
            front = 0;
         }
      }
      front = temp;
   }
   return s.toString();
}
```

3.3 Simulation



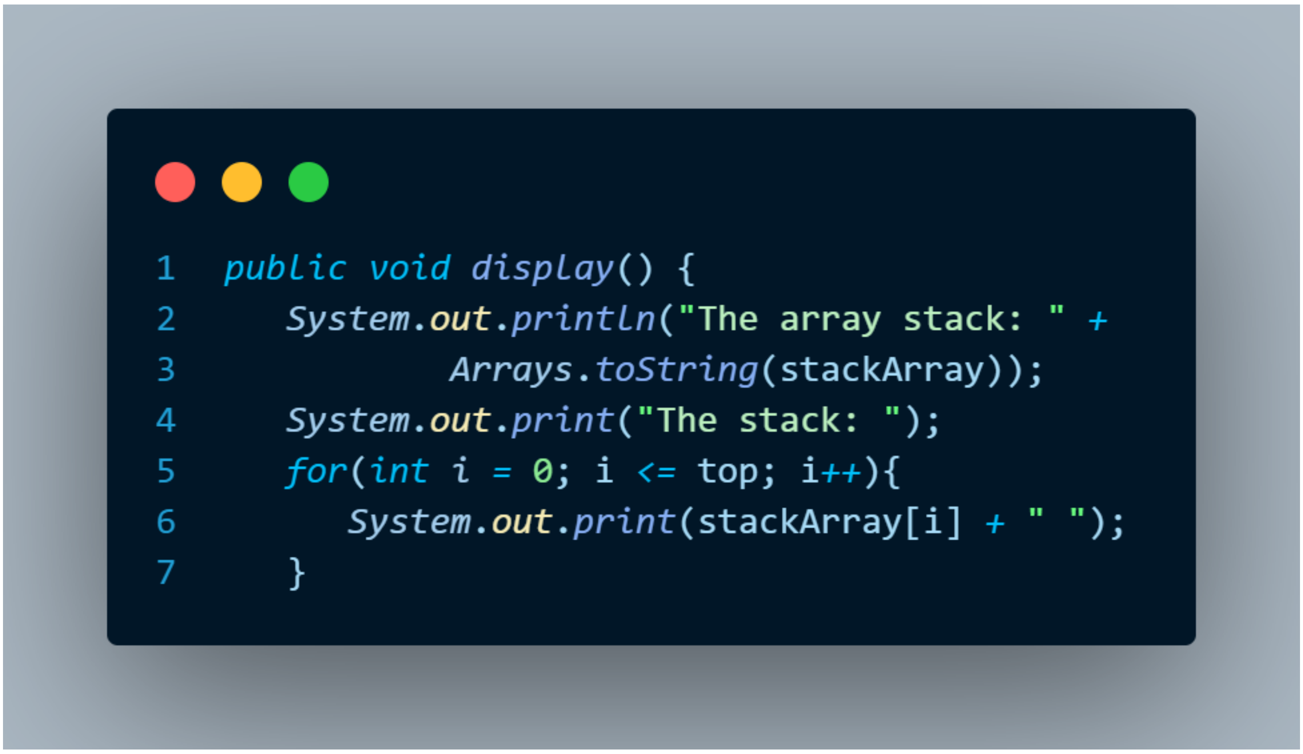
```java
Queue<Customer> customerQueue = new Queue<>(5);
final int NUM_CUSTOMERS = 5;
long cashierTime = 0, start = 0, depart = 0, totalTime = 0, count = NUM_CUSTOMERS - 1;
double rateOfCustomer = 0;
customerQueue.setProcessingTime(50);
for(int i = 0; i < NUM_CUSTOMERS; i++) {
   customerQueue.insert(new Customer(15*i));
}

while(!customerQueue.isEmpty()) {
   customer = customerQueue.remove();
   System.out.println("Arrival time of the customer "
           + (NUM_CUSTOMERS - count) + ": " + customer.getArrivalTime());
   count--;
   if(customer.getArrivalTime() > cashierTime) {
      start = customer.getArrivalTime();
   } else {
      start = cashierTime;
   }
   depart = start + customerQueue.getProcessingTime();
   customer.setDepartureTime(depart);
   cashierTime = depart;
   totalTime += customer.getTotalTime();
   System.out.println("Time from waiting to done being served: "
           + customer.getTotalTime() + " seconds");

}
rateOfCustomer = (double)60*NUM_CUSTOMERS/totalTime;
System.out.println("The total time for processing all the customers is "
        + totalTime + " seconds");
System.out.println("If the range of time processing is more narrowed than" +
        " the arrival time, the waiting time + " +
        "processing time only takes processing time into account, " +
        "meaning the customers do not have to wait for the queue");
System.out.println("The rate at which customers arrive at the queue is "
        + rateOfCustomer + " customer/minute");
System.out.println("For larger processing time, the rate mentioned decreases" +
        ", which mean the time at which the customers arrive at the queue is prolonged");
}
```



```
Arrival time of the customer 1: 0
Time from waiting to done being served: 50 seconds
Arrival time of the customer 2: 15
Time from waiting to done being served: 85 seconds
Arrival time of the customer 3: 30
Time from waiting to done being served: 120 seconds
Arrival time of the customer 4: 45
Time from waiting to done being served: 155 seconds
Arrival time of the customer 5: 60
Time from waiting to done being served: 190 seconds
The total time for processing all the customers is 600 seconds
```

4 StackApp.java

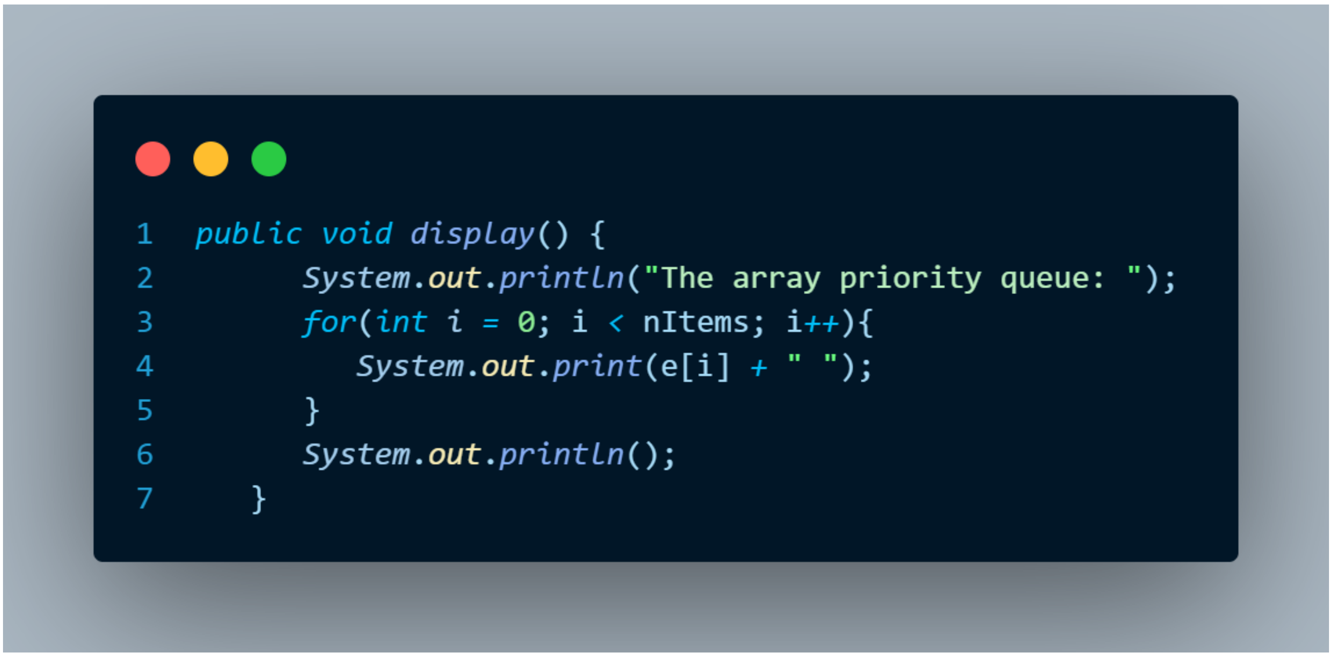
4.1 Method to display the stack array and the stack itself



```java
public void display() {
    System.out.println("The array stack: " +
            Arrays.toString(stackArray));
    System.out.print("The stack: ");
    for(int i = 0; i <= top; i++){
        System.out.print(stackArray[i] + " ");
    }
```

5 PriorityQApp.java

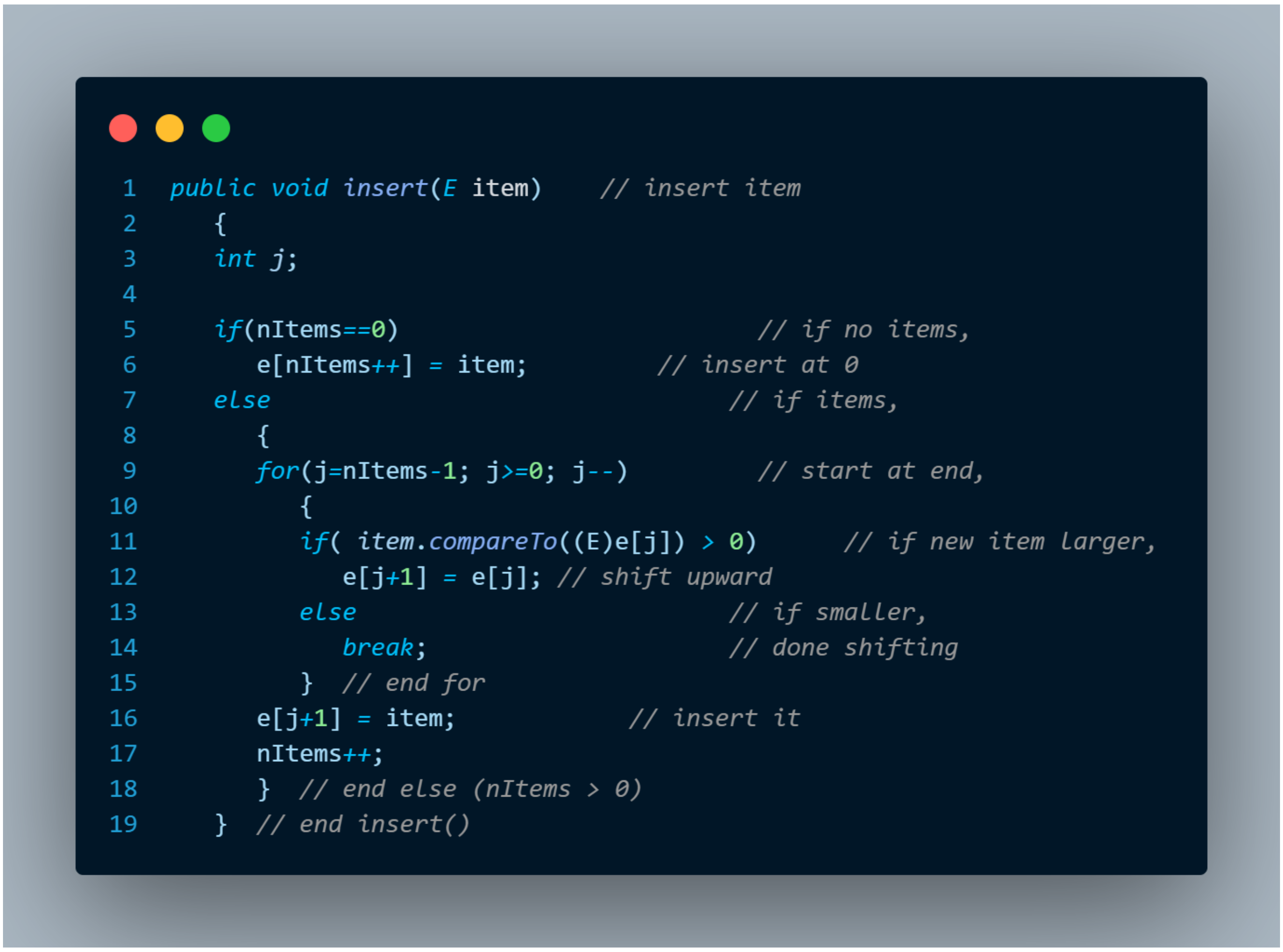
5.1 Method to display the queue



```java
public void display() {
        System.out.println("The array priority queue: ");
        for(int i = 0; i < nItems; i++){
           System.out.print(e[i] + " ");
        }
        System.out.println();
    }
```

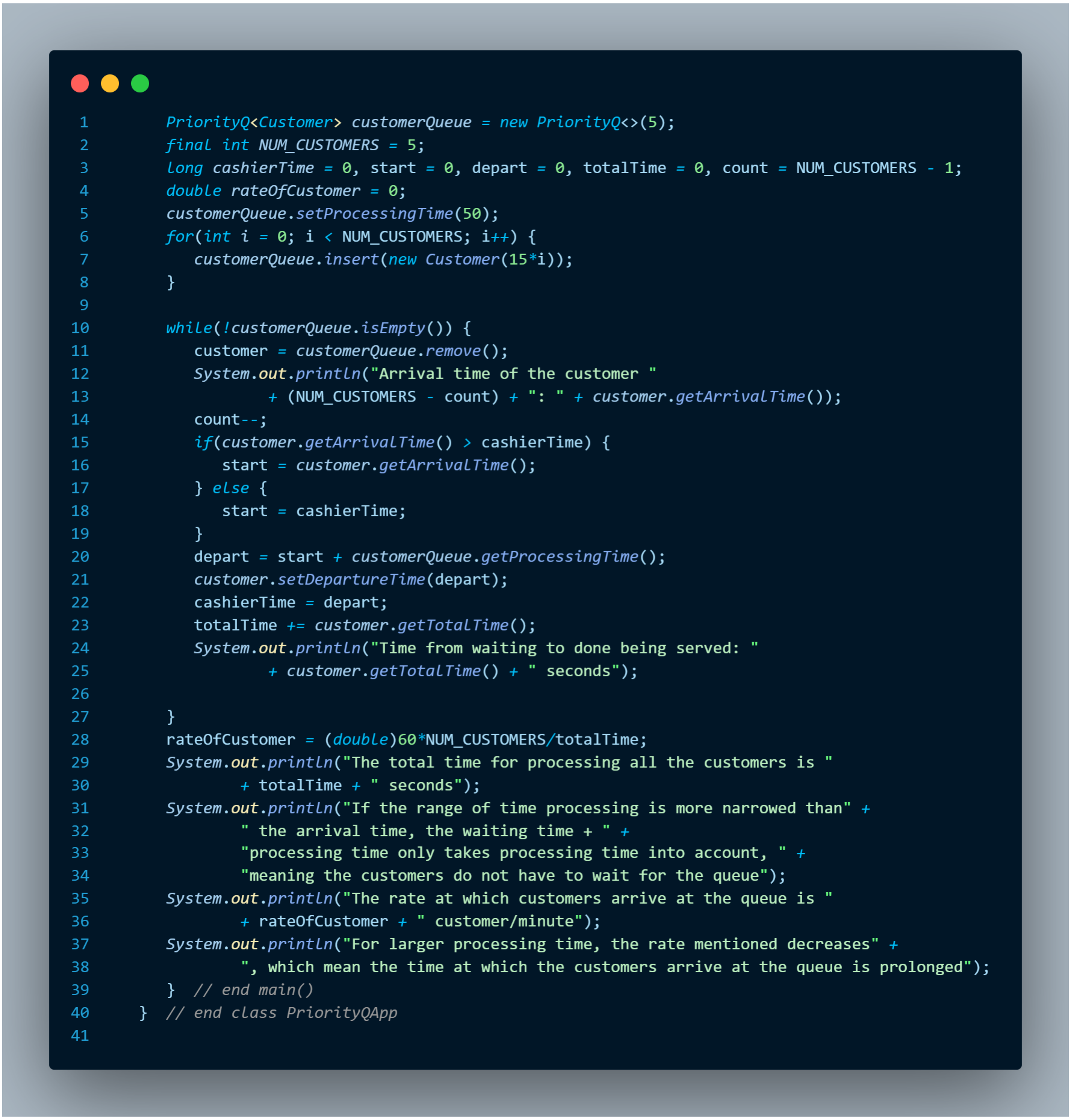
5.2 Compare queue and priority queue insertion method

For inserting at the rear using Priority Queue, it is less efficient than basic Queue since its insertion also sorts the queue.



```
public void insert(E item)    // insert item
   {
   int j;

   if(nItems==0)                         // if no items,
      e[nItems++] = item;         // insert at 0
   else                                // if items,
      {
      for(j=nItems-1; j>=0; j--)         // start at end,
         {
         if( item.compareTo((E)e[j]) > 0)      // if new item larger,
            e[j+1] = e[j]; // shift upward
         else                          // if smaller,
            break;                     // done shifting
         }  // end for
      e[j+1] = item;            // insert it
      nItems++;
      }  // end else (nItems > 0)
   }  // end insert()
```

5.3 Priority simulation



```
   PriorityQ<Customer> customerQueue = new PriorityQ<>(5);
   final int NUM_CUSTOMERS = 5;
   long cashierTime = 0, start = 0, depart = 0, totalTime = 0, count = NUM_CUSTOMERS - 1;
   double rateOfCustomer = 0;
   customerQueue.setProcessingTime(50);
   for(int i = 0; i < NUM_CUSTOMERS; i++) {
      customerQueue.insert(new Customer(15*i));
   }

   while(!customerQueue.isEmpty()) {
      customer = customerQueue.remove();
      System.out.println("Arrival time of the customer "
              + (NUM_CUSTOMERS - count) + ": " + customer.getArrivalTime());
      count--;
      if(customer.getArrivalTime() > cashierTime) {
         start = customer.getArrivalTime();
      } else {
         start = cashierTime;
      }
      depart = start + customerQueue.getProcessingTime();
      customer.setDepartureTime(depart);
      cashierTime = depart;
      totalTime += customer.getTotalTime();
      System.out.println("Time from waiting to done being served: "
              + customer.getTotalTime() + " seconds");

   }
   rateOfCustomer = (double)60*NUM_CUSTOMERS/totalTime;
   System.out.println("The total time for processing all the customers is "
           + totalTime + " seconds");
   System.out.println("If the range of time processing is more narrowed than" +
           " the arrival time, the waiting time + " +
           "processing time only takes processing time into account, " +
           "meaning the customers do not have to wait for the queue");
   System.out.println("The rate at which customers arrive at the queue is "
           + rateOfCustomer + " customer/minute");
   System.out.println("For larger processing time, the rate mentioned decreases" +
           ", which mean the time at which the customers arrive at the queue is prolonged");
   } // end main()
}  // end class PriorityQApp
```

This is the end of the report