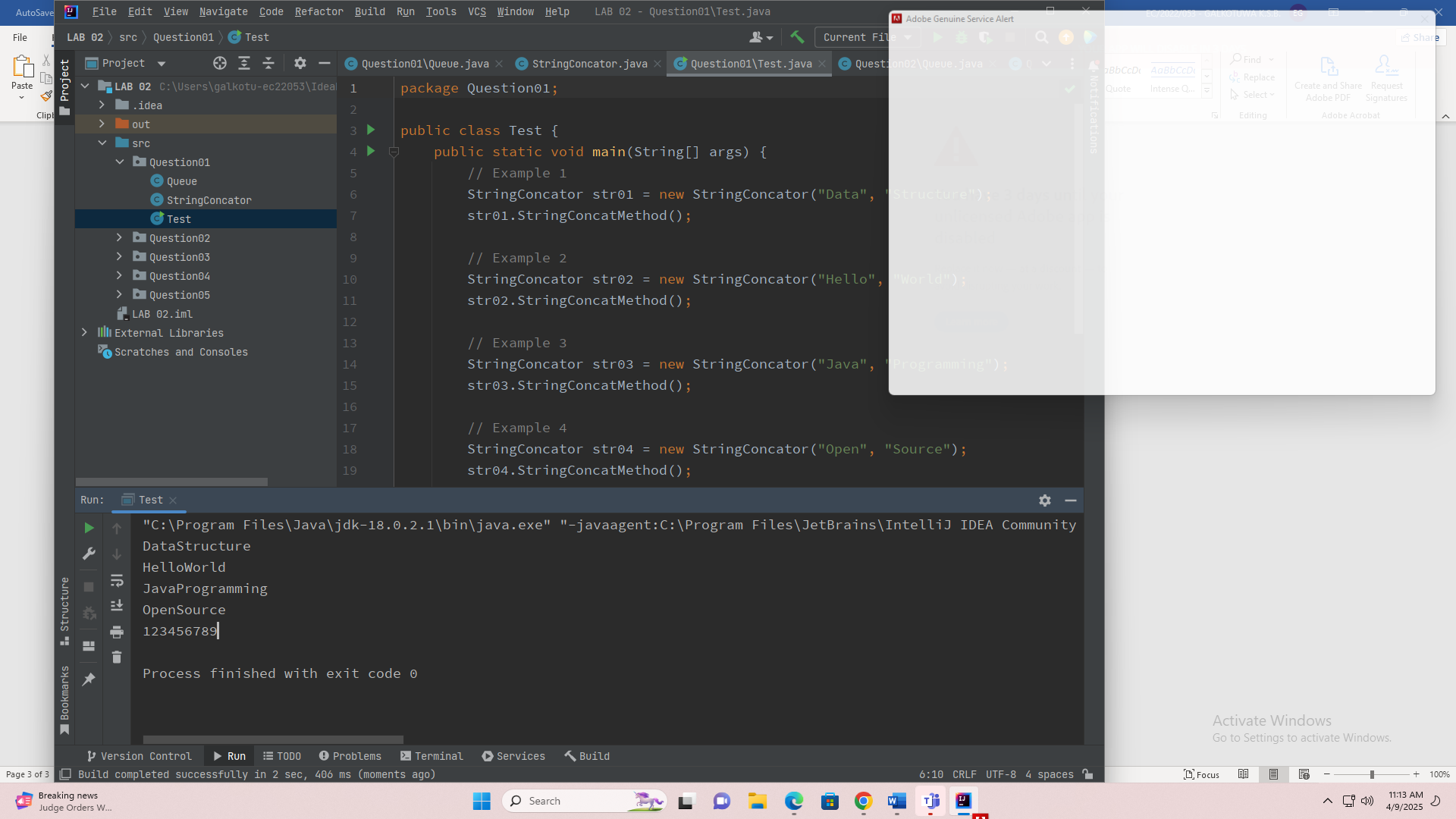
Question 01.

package Question01;  
public class **Queue** {  
 int front;  
 int count;  
 char[] queue;  
 int maxSize;  
 int rear;  
 public Queue(int maxSize) {  
 this.front = 0;  
 this.count = 0;  
 this.queue = new char[maxSize];  
 this.maxSize = maxSize;  
 this.rear = -1;  
 }  
 boolean IsQueueEmpty(){  
 if (rear<front)  
 return true;  
 else  
 return false;  
 }  
 boolean IsQueueFull() {  
 if (rear == maxSize - 1) {  
 return true;  
 }  
 return false;  
 }  
 void Append(char item) {  
 if (IsQueueFull()) {  
 System.*out*.printf("\nQueue is full\n");  
 } else {  
 queue[++rear] = (char) item;  
 count++;  
 }  
 }  
 char Serve() {  
 if (IsQueueEmpty()) {  
 System.*out*.printf("\nQueue is empty\n");  
 return 0;  
 }  
 else {  
 char x = queue[front++];  
 count--;  
 return x;  
 }  
 }  
}

package Question01;  
import java.util.Arrays;  
public class **StringConcator** {  
  
 String str01;  
 String str02;  
  
 public StringConcator(String str01, String str02) {  
 this.str01 = str01;  
 this.str02 = str02;  
 }  
  
 public String StringConcatMethod() {  
 Queue q4str01 = new Queue(str01.length());  
 Queue q4str02 = new Queue(str02.length());  
 Queue q4all = new Queue(q4str01.maxSize + q4str02.maxSize);  
  
 while (q4all.IsQueueEmpty()){  
 char[] q1charArray =str01.toCharArray();  
 char[] q2charArray =str02.toCharArray();  
 for (char i : q1charArray) {  
 q4all.Append(i);  
 }  
 for (char i2:q2charArray) {  
 q4all.Append(i2);  
  
 }  
 }  
 System.*out*.println(q4all.queue);  
 return Arrays.*toString*(q4all.queue);  
  
 }  
}

package Question01;  
  
public class **Test** {  
 public static void main(String[] args) {  
 // Example 1  
 StringConcator str01 = new StringConcator("Data", "Structure");  
 str01.StringConcatMethod();  
  
 // Example 2  
 StringConcator str02 = new StringConcator("Hello", "World");  
 str02.StringConcatMethod();  
  
 // Example 3  
 StringConcator str03 = new StringConcator("Java", "Programming");  
 str03.StringConcatMethod();  
  
 // Example 4  
 StringConcator str04 = new StringConcator("Open", "Source");  
 str04.StringConcatMethod();  
  
 // Example 5  
 StringConcator str05 = new StringConcator("12345", "6789");  
 str05.StringConcatMethod();  
  
 }  
}

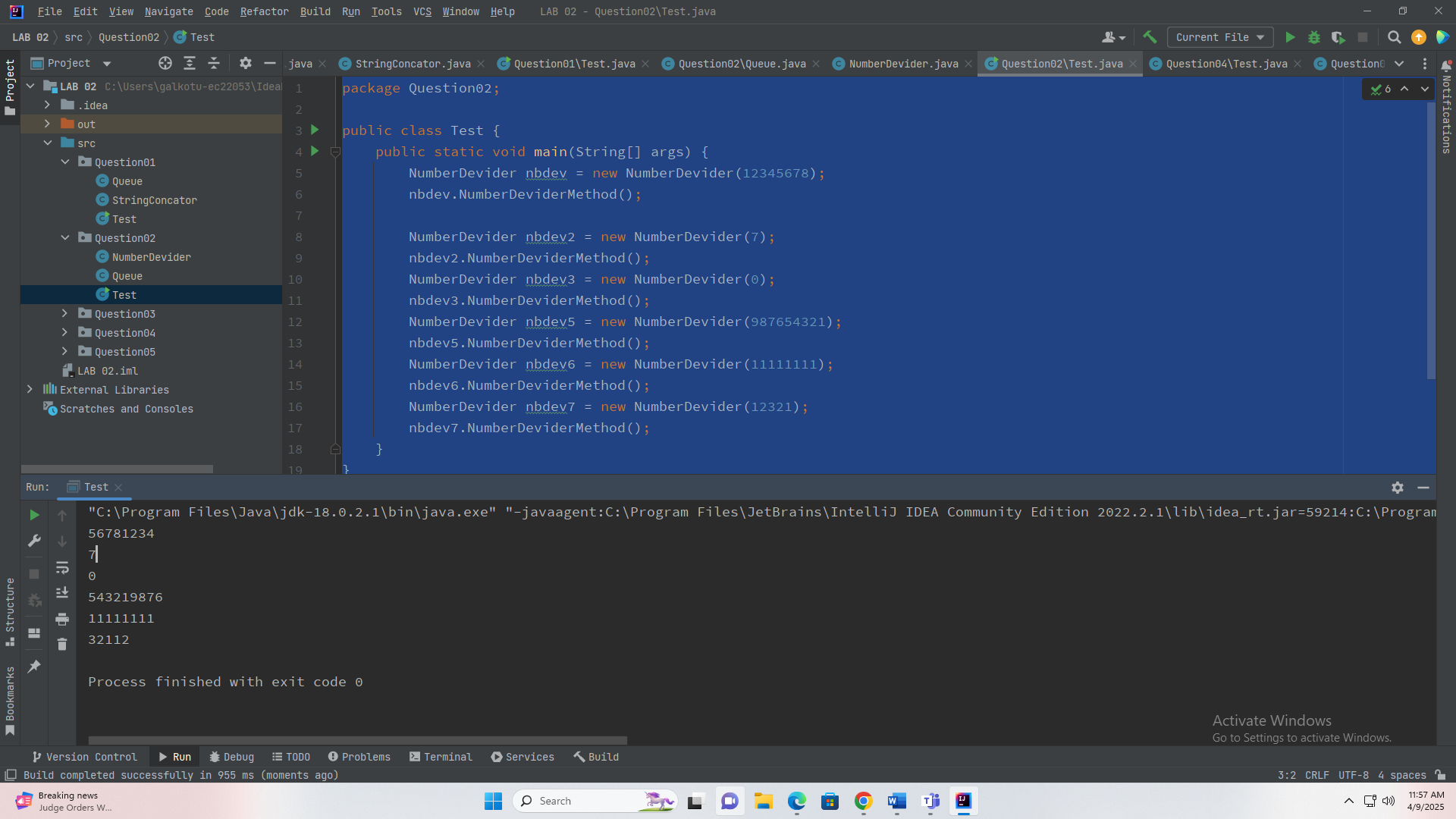


Question 02.

package Question02;  
public class **Queue** {  
 int front;  
 int count;  
 char[] queue;  
 int maxSize;  
 int rear;  
 public Queue(int maxSize) {  
 this.front = 0;  
 this.count = 0;  
 this.queue = new char[maxSize];  
 this.maxSize = maxSize;  
 this.rear = -1;  
 }  
 boolean IsQueueEmpty(){  
 if (rear<front)  
 return true;  
 else  
 return false;  
 }  
 boolean IsQueueFull() {  
 if (rear == maxSize - 1) {  
 return true;  
 }  
 return false;  
 }  
 void Append(char item) {  
 if (IsQueueFull()) {  
 System.*out*.printf("\nQueue is full\n");  
 } else {  
 queue[++rear] = (char) item;  
 count++;  
 }  
 }  
 char Serve() {  
 if (IsQueueEmpty()) {  
 System.*out*.printf("\nQueue is empty\n");  
 return 0;  
 }  
 else {  
 char x = queue[front++];  
 count--;  
 return x;  
 }  
 }  
}

package Question02;  
public class **NumberDevider** {  
 int numberSize ;  
 int Number;  
 int midValue;  
 public NumberDevider(int number) {  
 char[] digitList = Integer.*toString*(number).toCharArray();  
 this.Number = number;  
 this.numberSize = digitList.length;  
 this.midValue = numberSize/2;  
 }  
 public int NumberDeviderMethod(){  
 String Numb = Integer.*toString*(Number);  
 char[] digitList = Numb.toCharArray();  
 Queue mainQueueHoldAllNumbers = new Queue(numberSize);  
 while (mainQueueHoldAllNumbers.IsQueueEmpty()) {  
 for (int i = midValue; i < digitList.length; i++) {  
 mainQueueHoldAllNumbers.Append(digitList[i]);  
 }  
 for (int i = 0; i < midValue; i++) {  
 mainQueueHoldAllNumbers.Append(digitList[i]);  
 }  
 }  
 System.*out*.println(mainQueueHoldAllNumbers.queue);  
  
 return Integer.*parseInt*(String.*valueOf*(mainQueueHoldAllNumbers.queue));  
 }  
  
}

package Question02;  
  
public class **Test** {  
 public static void main(String[] args) {  
 NumberDevider nbdev = new NumberDevider(12345678);  
 nbdev.NumberDeviderMethod();  
 NumberDevider nbdev2 = new NumberDevider(7);  
 nbdev2.NumberDeviderMethod();  
 NumberDevider nbdev3 = new NumberDevider(0);  
 nbdev3.NumberDeviderMethod();  
 NumberDevider nbdev5 = new NumberDevider(987654321);  
 nbdev5.NumberDeviderMethod();  
 NumberDevider nbdev6 = new NumberDevider(11111111);  
 nbdev6.NumberDeviderMethod();  
 NumberDevider nbdev7 = new NumberDevider(12321);  
 nbdev7.NumberDeviderMethod();  
 }  
}



Question 03.

package **Question03**;

public class Queue {

int front;

int count;

char[] queue;

int maxSize;

int rear;

public Queue(int maxSize) {

this.front = 0;

this.count = 0;

this.queue = new char[maxSize];

this.maxSize = maxSize;

this.rear = -1;

}

boolean IsQueueEmpty(){

if (rear < front)

return true;

else

return false;

}

boolean IsQueueFull() {

if (rear == maxSize - 1) {

return true;

}

return false;

}

void Append(char item) {

if (IsQueueFull()) {

System.out.printf("\nQueue is full\n");

} else {

queue[++rear] = (char) item;

count++;

}

}

char Serve() {

if (IsQueueEmpty()) {

System.out.printf("\nQueue is empty\n");

return 0;

}

else {

char x = queue[front++];

count--;

return x;

}

}

}

package **Question03**;

public class VowelFinder {

String inputString;

public VowelFinder(String inputString) {

this.inputString = inputString.toUpperCase();

}

public void findVowelsInOrder() {

Queue resultQueue = new Queue(5);

if (inputString.contains("A")) {

resultQueue.Append('A');

}

if (inputString.contains("E")) {

resultQueue.Append('E');

}

if (inputString.contains("I")) {

resultQueue.Append('I');

}

if (inputString.contains("O")) {

resultQueue.Append('O');

}

if (inputString.contains("U")) {

resultQueue.Append('U');

}

System.out.print("Output: ");

while (!resultQueue.IsQueueEmpty()) {

System.out.print(resultQueue.Serve() + " ");

}

System.out.println();

}

public boolean isVowel(char c) {

c = Character.toUpperCase(c);

return c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U';

}

}

package Question03;

public class **Test** {

public static void main(String[] args) {

// Example 1

VowelFinder example1 = new VowelFinder("DIALOGUE");

System.out.println("Input: " + example1.inputString);

example1.findVowelsInOrder();

// Example 2

VowelFinder example2 = new VowelFinder("COMPUTER");

System.out.println("Input: " + example2.inputString);

example2.findVowelsInOrder();

// Example 3

VowelFinder example3 = new VowelFinder("PROGRAMMING");

System.out.println("Input: " + example3.inputString);

example3.findVowelsInOrder();

// Example 4

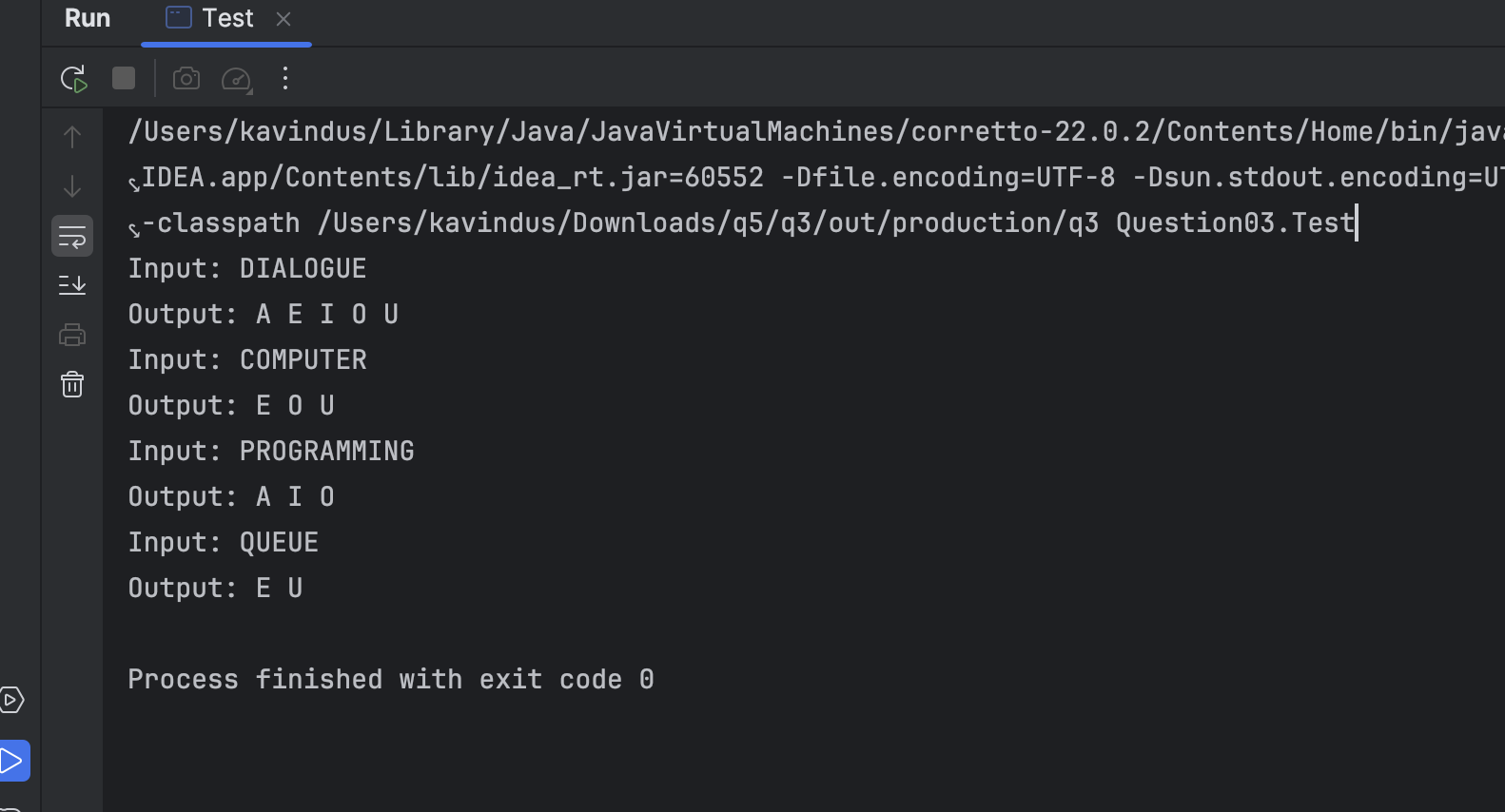
VowelFinder example4 = new VowelFinder("QUEUE");

System.out.println("Input: " + example4.inputString);

example4.findVowelsInOrder();

}

}



Question 04.

package Question04;

public class **Node** {

int data;

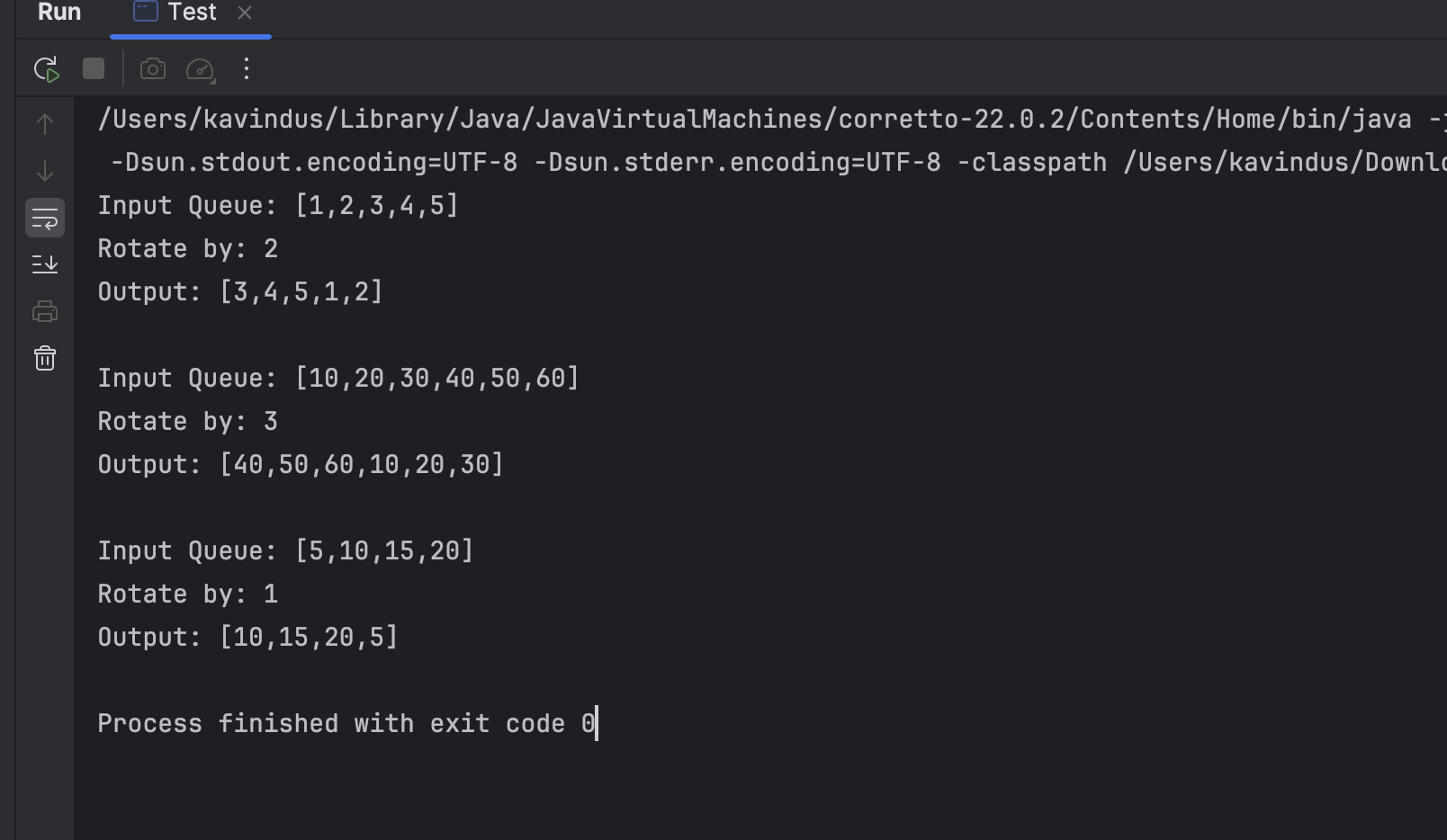
Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}  
  


package Question04;

public class **LinkedQueue** {

private Node front;

private Node rear;

private int size;

public LinkedQueue() {

this.front = null;

this.rear = null;

this.size = 0;

}

public boolean isEmpty() {

return front == null;

}

public void enqueue(int data) {

Node newNode = new Node(data);

if (isEmpty()) {

front = newNode;

} else {

rear.next = newNode;

}

rear = newNode;

size++;

}

public int dequeue() {

if (isEmpty()) {

System.out.println("Queue is empty");

return -1;

}

int data = front.data;

front = front.next;

if (front == null) {

rear = null;

}

size--;

return data;

}

public int size() {

return size;

}

public void display() {

if (isEmpty()) {

System.out.println("[]");

return;

}

StringBuilder sb = new StringBuilder("[");

Node current = front;

while (current != null) {

sb.append(current.data);

current = current.next;

if (current != null) {

sb.append(",");

}

}

sb.append("]");

System.out.println(sb.toString());

}

}

package Question04;

public class **QueueRotator** {

LinkedQueue queue;

int rotateBy;

public QueueRotator(int[] elements, int rotateBy) {

this.queue = new LinkedQueue();

this.rotateBy = rotateBy;

for (int element : elements) {

queue.enqueue(element);

}

}

public void rotate() {

if (queue.isEmpty() || rotateBy <= 0 || rotateBy % queue.size() == 0) {

return;

}

rotateBy = rotateBy % queue.size();

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

int temp = queue.dequeue();

queue.enqueue(temp);

}

}

public void displayQueue() {

queue.display();

}

}

package Question04;

public class **Test** {

public static void main(String[] args) {

// Example 1

int[] elements1 = {1, 2, 3, 4, 5};

QueueRotator rotator1 = new QueueRotator(elements1, 2);

System.out.print("Input Queue: ");

rotator1.displayQueue();

System.out.print("Rotate by: 2\nOutput: ");

rotator1.rotate();

rotator1.displayQueue();

System.out.println();

// Example 2

int[] elements2 = {10, 20, 30, 40, 50, 60};

QueueRotator rotator2 = new QueueRotator(elements2, 3);

System.out.print("Input Queue: ");

rotator2.displayQueue();

System.out.print("Rotate by: 3\nOutput: ");

rotator2.rotate();

rotator2.displayQueue();

System.out.println();

// Example 3

int[] elements3 = {5, 10, 15, 20};

QueueRotator rotator3 = new QueueRotator(elements3, 1);

System.out.print("Input Queue: ");

rotator3.displayQueue();

System.out.print("Rotate by: 1\nOutput: ");

rotator3.rotate();

rotator3.displayQueue();

}

}

Question 05.

package Question05;

public class **CharNode** {

char data;

CharNode next;

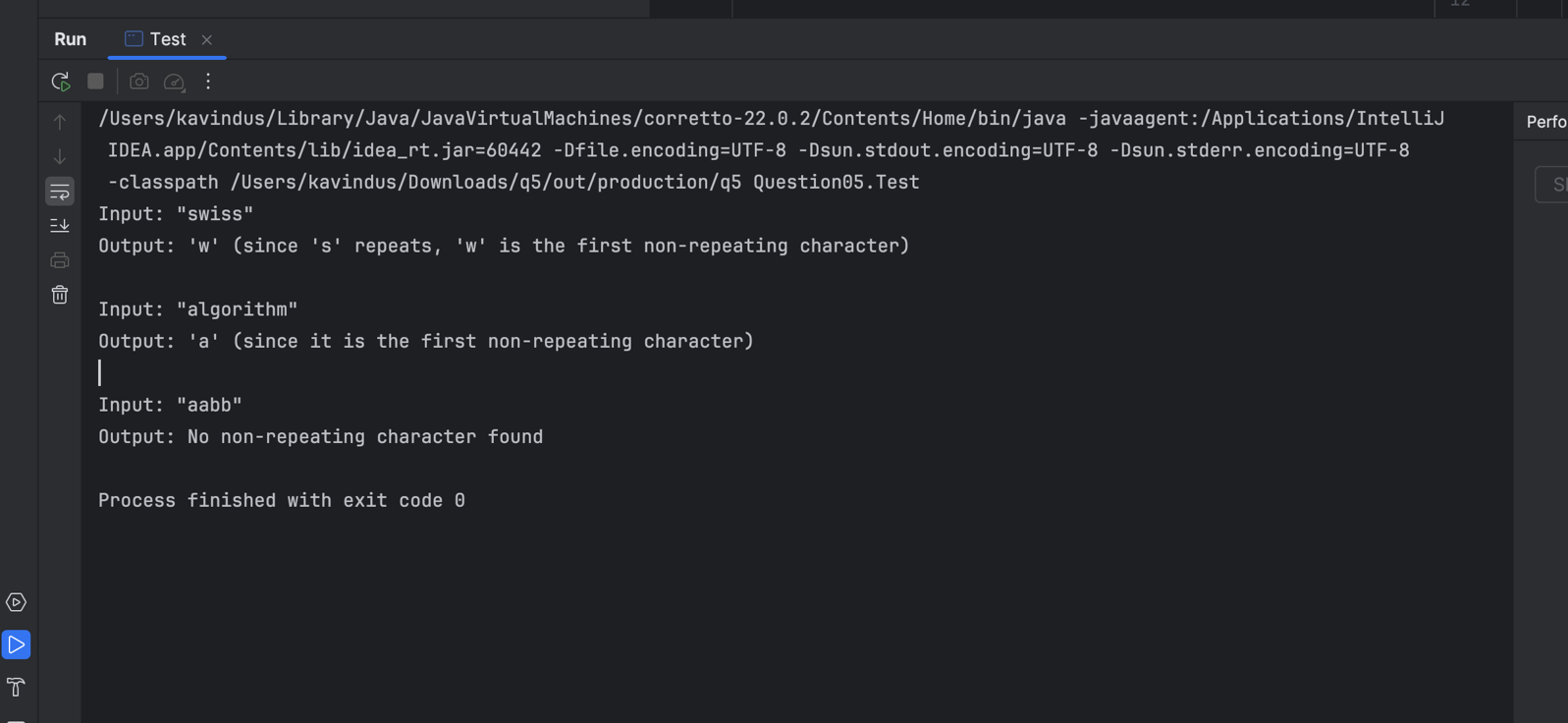
public CharNode(char data) {

this.data = data;

this.next = null;

}

}



package Question05;

public class **LinkedCharQueue** {

private CharNode front;

private CharNode rear;

public LinkedCharQueue() {

this.front = null;

this.rear = null;

}

public boolean isEmpty() {

return front == null;

}

public void enqueue(char data) {

CharNode newNode = new CharNode(data);

if (isEmpty()) {

front = newNode;

} else {

rear.next = newNode;

}

rear = newNode;

}

public char dequeue() {

if (isEmpty()) {

System.out.println("Queue is empty");

return '\0';

}

char data = front.data;

front = front.next;

if (front == null) {

rear = null;

}

return data;

}

public char peek() {

if (isEmpty()) {

System.out.println("Queue is empty");

return '\0';

}

return front.data;

}

public void remove(char c) {

if (isEmpty()) {

return;

}

if (front.data == c) {

front = front.next;

if (front == null) {

rear = null;

}

return;

}

CharNode current = front;

CharNode prev = null;

while (current != null && current.data != c) {

prev = current;

current = current.next;

}

if (current != null) {

prev.next = current.next;

if (current == rear) {

rear = prev;

}

}

}

}

package Question05;

public class **NonRepeatingCharFinder** {

String inputString;

public NonRepeatingCharFinder(String inputString) {

this.inputString = inputString;

}

public char findFirstNonRepeating() {

LinkedCharQueue queue = new LinkedCharQueue();

int[] charCount = new int[256];

for (int i = 0; i < inputString.length(); i++) {

char c = inputString.charAt(i);

charCount[c]++;

if (charCount[c] == 1) {

queue.enqueue(c);

}

else if (charCount[c] == 2) {

queue.remove(c);

}

}

if (!queue.isEmpty()) {

return queue.peek();

}

return '\0';

}

}

package Question05;

public class **Test** {

public static void main(String[] args) {

// Example 1

NonRepeatingCharFinder finder1 = new NonRepeatingCharFinder("swiss");

char result1 = finder1.findFirstNonRepeating();

System.out.println("Input: \"" + finder1.inputString + "\"");

System.out.println("Output: '" + result1 + "' (since 's' repeats, '" + result1 + "' is the first non-repeating character)");

System.out.println();

// Example 2

NonRepeatingCharFinder finder2 = new NonRepeatingCharFinder("algorithm");

char result2 = finder2.findFirstNonRepeating();

System.out.println("Input: \"" + finder2.inputString + "\"");

System.out.println("Output: '" + result2 + "' (since it is the first non-repeating character)");

System.out.println();

// Example 3

NonRepeatingCharFinder finder3 = new NonRepeatingCharFinder("aabb");

char result3 = finder3.findFirstNonRepeating();

System.out.println("Input: \"" + finder3.inputString + "\"");

if (result3 == '\0') {

System.out.println("Output: No non-repeating character found");

} else {

System.out.println("Output: '" + result3 + "' (since it is the first non-repeating character)");

}

}

}