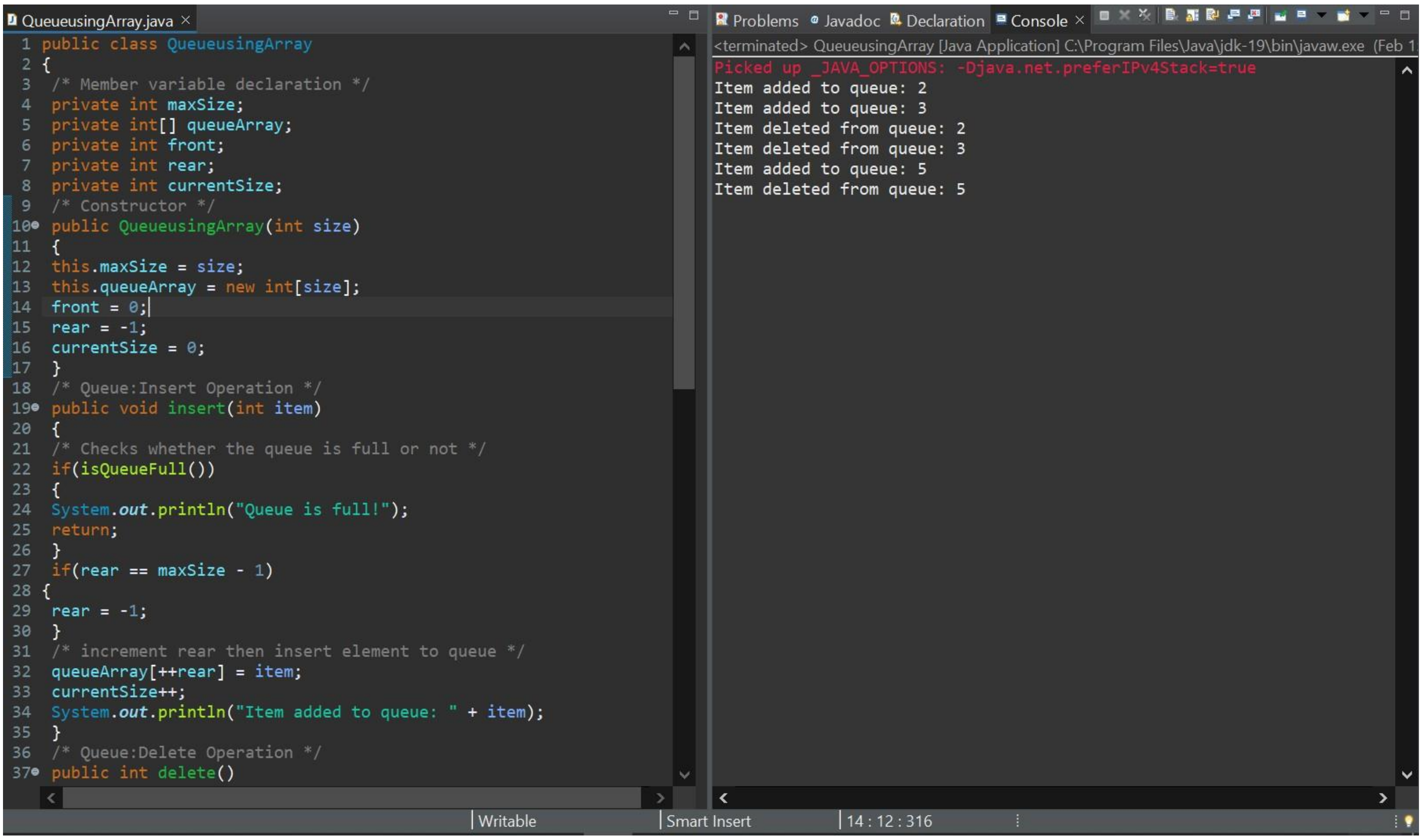
LAB SHEET -2

AIM 1: Understanding the concepts of Stack, Queue (10 points)

1. **Write Java programs to implement a queue using an array.**



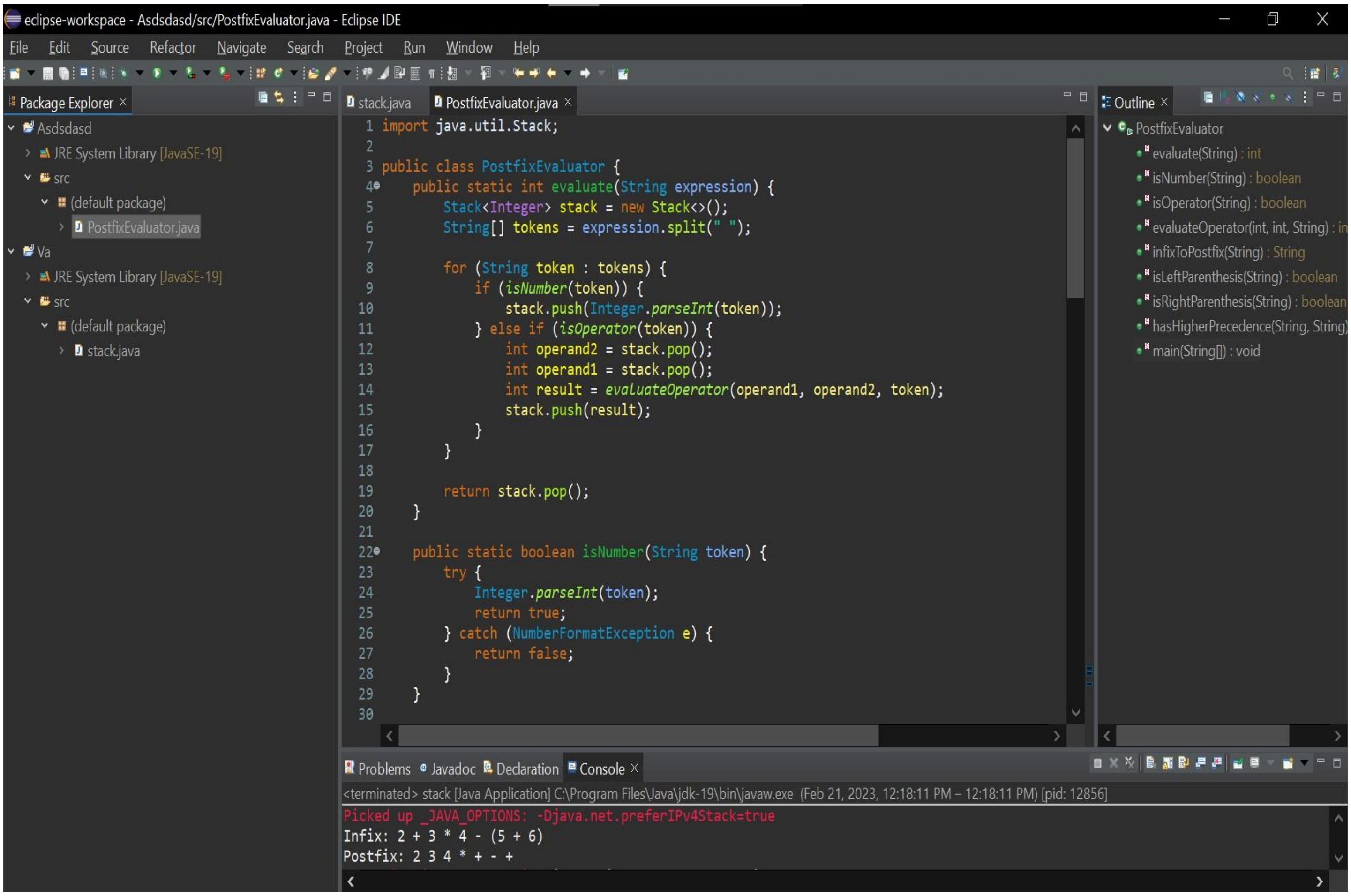
```java
public class QueueusingArray
{
 /* Member variable declaration */
 private int maxSize;
 private int[] queueArray;
 private int front;
 private int rear;
 private int currentSize;
 /* Constructor */
 public QueueusingArray(int size)
 {
 this.maxSize = size;
 this.queueArray = new int[size];
 front = 0;
 rear = -1;
 currentSize = 0;
 }
 /* Queue:Insert Operation */
 public void insert(int item)
 {
 /* Checks whether the queue is full or not */
 if(isQueueFull())
 {
 System.out.println("Queue is full!");
 return;
 }
 if(rear == maxSize - 1)
{
 rear = -1;
 }
 /* increment rear then insert element to queue */
 queueArray[++rear] = item;
 currentSize++;
 System.out.println("Item added to queue: " + item);
 }
 /* Queue:Delete Operation */
 public int delete()
```

```
<terminated> QueueusingArray [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (Feb 1
Picked up _JAVA_OPTIONS: -Djava.net.preferIPv4Stack=true
Item added to queue: 2
Item added to queue: 3
Item deleted from queue: 2
Item deleted from queue: 3
Item added to queue: 5
Item deleted from queue: 5
```

1. Code:

```java
public class QueueusingArray
{
/* Member variable declaration */
private int maxSize;
private int[] queueArray;
private int front;
private int rear;
private int currentSize;
/* Constructor */
public QueueusingArray(int size)
{
this.maxSize = size;
this.queueArray = new int[size];
front = 0;
rear = -1;
currentSize = 0;
}
/* Queue:Insert Operation */
public void insert(int item)
{
/* Checks whether the queue is full or not */
if(isQueueFull())
{
System.out.println("Queue is full!");
return;
}
if(rear == maxSize - 1)
{
rear = -1;
}
/* increment rear then insert element to queue */
queueArray[++rear] = item;
currentSize++;
System.out.println("Item added to queue: " + item);
}
/* Queue:Delete Operation */
public int delete()
{
/* Checks whether the queue is empty or not */
if(isQueueEmpty())
{
throw new RuntimeException("Queue is empty");
}
/* retrieve queue element then increment */
int temp = queueArray[front++];
if(front == maxSize)
{
front = 0;
}
currentSize--;
return temp;
}
/* Queue:Peek Operation */
public int peek()
{
return queueArray[front];
}
/* Queue:isFull Operation */
public boolean isQueueFull()
{
return (maxSize == currentSize);
}
/* Queue:isEmpty Operation */
public boolean isQueueEmpty()
{
return (currentSize == 0);
}
/* Driver Code */
public static void main(String[] args)
{
QueueusingArray queue = new QueueusingArray(10);
queue.insert(2);
queue.insert(3);
System.out.println("Item deleted from queue: " + queue.delete());
System.out.println("Item deleted from queue: " + queue.delete());
queue.insert(5);
System.out.println("Item deleted from queue: " + queue.delete());
}
}
```

2. **Write a java program that reads an infix expression, converts the expression to postfix form and then evaluates the postfix expression (use stack ADT).**


eclipse-workspace - Asdsdasd/src/PostfixEvaluator.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Asdsdasd
JRE System Library [JavaSE-19]
src
(default package)
PostfixEvaluator.java
Va
JRE System Library [JavaSE-19]
src
(default package)
stack.java
stack.java
PostfixEvaluator.java
Outline
PostfixEvaluator
evaluate(String) : int
isNumber(String) : boolean
isOperator(String) : boolean
evaluateOperator(int, int, String) : in
infixToPostfix(String) : String
isLeftParenthesis(String) : boolean
isRightParenthesis(String) : boolean
hasHigherPrecedence(String, String)
main(String[]) : void
Problems
Javadoc
Declaration
Console

```
import java.util.Stack;

public class PostfixEvaluator {
    public static int evaluate(String expression) {
        Stack<Integer> stack = new Stack<>();
        String[] tokens = expression.split(" ");

        for (String token : tokens) {
            if (isNumber(token)) {
                stack.push(Integer.parseInt(token));
            } else if (isOperator(token)) {
                int operand2 = stack.pop();
                int operand1 = stack.pop();
                int result = evaluateOperator(operand1, operand2, token);
                stack.push(result);
            }
        }

        return stack.pop();
    }

    public static boolean isNumber(String token) {
        try {
            Integer.parseInt(token);
            return true;
        } catch (NumberFormatException e) {
            return false;
        }
    }

```

```
<terminated> stack [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (Feb 21, 2023, 12:18:11 PM – 12:18:11 PM) [pid: 12856]
Picked up _JAVA_OPTIONS: -Djava.net.preferIPv4Stack=true
Infix: 2 + 3 * 4 - (5 + 6)
Postfix: 2 3 4 * + - +
```

2.Code:

```
import java.util.Stack;

public class PostfixEvaluator {

public static int evaluate(String expression) {

Stack<Integer> stack = new Stack<>();

String[] tokens = expression.split(" ");

for (String token : tokens) {

if (isNumber(token)) {

stack.push(Integer.parseInt(token));

} else if (isOperator(token)) {

int operand2  =  stack.pop();

int operand1 = stack.pop();
int result = evaluateOperator(operand1, operand2, token);

stack.push(result);

}

}

return stack.pop();

}

public static boolean isNumber(String token) {

try {

Integer.parseInt(token);

return true;

} catch (NumberFormatException e) {

return false;

}

}

public static boolean isOperator(String token) {

return "+-*/".contains(token);

}

public static int evaluateOperator(int operand1, int operand2, String operator) {

switch (operator) {

case "+":

return operand1 + operand2;

case "-":

return operand1 - operand2;

case "*":

return operand1 * operand2;

case "/":

return operand1 / operand2;

default:

throw new IllegalArgumentException("Invalid operator: " + operator);
```

```
}
}
public static String infixToPostfix(String expression) {
StringBuilder output = new StringBuilder();
Stack<String> stack = new Stack<>();
String[] tokens = expression.split(" ");
for (String token : tokens) {
if (isNumber(token)) {
output.append(token).append(" ");
} else if (isOperator(token)) {
while (!stack.empty() && !isLeftParenthesis(stack.peek()) && hasHigherPrecedence(stack.peek(), token)) {
output.append(stack.pop()).append(" ");
}
stack.push(token);
} else if (isLeftParenthesis(token)) {
stack.push(token);
} else if (isRightParenthesis(token)) {
while (!stack.empty() && !isLeftParenthesis(stack.peek())) {
output.append(stack.pop()).append(" ");
}
stack.pop();
}
}
while (!stack.empty()) {
output.append(stack.pop()).append(" ");
}
return output.toString().trim();
}
public static boolean isLeftParenthesis(String token) {
```

```java
return token.equals("(");

}

public static boolean isRightParenthesis(String token) {

return token.equals(")");

}

public static boolean hasHigherPrecedence(String operator1, String operator2) {

if (operator1.equals("*") || operator1.equals("/")) {

return true;

} else if (operator1.equals("+") || operator1.equals("-")) {

return (operator2.equals("+") || operator2.equals("-")) ? true : false;

}

return false;

}

public static void main(String[] args) {

String infix = "2 + 3 * 4 - (5 + 6)";

String postfix = infixToPostfix(infix);

System.out.println("Infix: " + infix);

System.out.println("Postfix: " + postfix);

int result = evaluate(postfix);

System.out.println("Result: " + result);

}

}
```