**Nicholas White** Assignment 3 - Stacks and Queues

Exercise 1 - Implement a Queue with a Deque

**QueueFromDequeImpl.java**

Class which contains work on these methods:

- enqueue
- dequeue
- peek
- isEmpty
- isFull
- display

Exercise 2 - Evaluation of Arithmetic Expressions

**ArithmeticExpressionEvaluator.java**

Class which contains work on these methods:

- evaluateArithmeticExpression

Exercise 3 (Bonus) - Delimiter Matching

**DelimiterValidation.java**

Class which contains work on these methods:

- checkfile

Exercise 4 - Shared Memory

**DoubleStackImpl.java**

Class which contains work on these methods:

- pushIn
- popIn
- pushOut
- popOut
- isEmpty
- isFull
- display
- peekIn
- isEmptyIn
- peekOut
- isEmptyOut

Exercise 6 - Queueing Stars

**AgesOfHollywood.java**

Class that contains work on these methods:

- parseTextFile

**QueueFromDequeImpl.java**

```
package sq_exercises;
import stacksandqueues.*;

public class QueueFromDequeImpl<E>
                    implements MyQueueIF<E> {

   MyDequeIF<E> myDeque;

   public QueueFromDequeImpl(E[] array) {
      myDeque = new MyDequeImpl<>(array);
   }

   @Override
   public void enqueue(E element) throws FullStructureException {
      if (isFull()){
         throw new FullStructureException("Queue is full.");
      }
      myDeque.insertRight(element);
   }

   @Override
   public E dequeue() throws EmptyStructureException {
      if (isEmpty()){
         throw new EmptyStructureException("Queue is empty.");
      }
      return(myDeque.removeLeft());
```

```
    }

    @Override
    public E peek() throws EmptyStructureException {
        if (isEmpty()){
            throw new EmptyStructureException("Queue is empty.");
        }
        return (myDeque.peekLeft());
    }

    @Override
    public boolean isEmpty() {
        return myDeque.isEmpty();
    }

    @Override
    public boolean isFull() {
        return myDeque.isFull();
    }

    @Override
    public void display() {
        myDeque.display();
    }
}
```

**ArithmeticExpressionEvaluator.java**

```
package sq_exercises;

import stacksandqueues.*;

public class ArithmeticExpressionEvaluator {

    MyStackIF<Character> MyCharStack;
    MyStackIF<Double> MyValueStack;
    Double[] array = new Double[1000];
    Character[] array2 = new Character[1000];
    double value = 0;

    public ArithmeticExpressionEvaluator() {
        MyCharStack = new MyStackImpl<>(array2);
        MyValueStack = new MyStackImpl<>(array);

    }

    public Double evaluateArithmeticExpression(String s) {
        try{
            do{
                for (int i=0; i<s.length();i++){
                    if (Character.isDigit(s.charAt(i))){
                        int digit = Character.getNumericValue(s.charAt(i));
                        MyValueStack.push((double)digit);
                    }
                    else if (s.charAt(i) == '(' || s.charAt(i) == '*' || s.charAt(i) == '/'
                            || s.charAt(i) == '+' || s.charAt(i) == '-'){
                        MyCharStack.push(s.charAt(i));
                    }
                    else if (s.charAt(i) == ')'){
                        MyCharStack.push(s.charAt(i));
                        value = MyValueStack.pop();
                        MyCharStack.pop();
                        while (MyCharStack.peek() != '('){
                            switch (MyCharStack.peek()){
                                case '*':
                                    value = MyValueStack.pop() * value;
                                    MyCharStack.pop();

                                    break;
                                case '/':
                                    value = MyValueStack.pop() / value;
                                    MyCharStack.pop();
```

```
                        break;
                    case '+':
                        value = MyValueStack.pop() + value;
                        MyCharStack.pop();

                        break;
                    case '-':
                        value = MyValueStack.pop() - value;
                        MyCharStack.pop();

                        break;
                }
            }
            if (MyCharStack.peek() == '('){
                MyValueStack.push(value);
                MyCharStack.pop();
            }
        }
    }

    } while (!MyCharStack.isEmpty());
    }
    catch (FullStructureException e){}
    catch (EmptyStructureException e){}
    return value;
  }
}
```

**DelimiterValidation.java**

```
package sq_exercises;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import stacksandqueues.*;

public class DelimiterValidation {


    MyStackIF<Character> myStack;
    Character[] array = new Character[100];
    String line = "";

    public DelimiterValidation() {
        myStack = new MyStackImpl<>(array);
    }

    public void checkFile(String pathname) {
        try{
            Scanner sc = new Scanner(new File(pathname));
            do{
                line = sc.nextLine();
                //System.out.println(line);
                for(int i=0;i<line.length();i++){
                    char a = line.charAt(i);
                    switch (a){
                        case '{':
                        case '[':
                        case '(':
                            myStack.push(a);
                            break;
                        case '}':
                        case ']':
                        case ')':
                            if (!myStack.isEmpty()){
                                char b = myStack.pop();
                                if (a == '}' && b != '{' ||
                                    a == ']' && b !='['  ||
                                    a == ')' && b != '('){
                                    System.out.println("Missing match for "+a+" at line "+i);
                                }
                            }
                            else{
```

```
                        System.out.println("Missing match for "+a+" at line "+i);
                    }
                    break;
                default:
                    break;
            }
        }
    } while (sc.hasNextLine());
    if (!myStack.isEmpty()){
        System.out.println("Missing end delimiter.");
    }
    else{
        System.out.println("File search completed. Delimiters are all matched.");
    }

}

catch (FileNotFoundException e){}
catch (FullStructureException e){}
catch (EmptyStructureException e){}
}
}
```

**DoubleStackImpl.java**

```
package sq_exercises;

import stacksandqueues.EmptyStructureException;
import stacksandqueues.FullStructureException;


public class DoubleStackImpl<E> implements DoubleStackIF<E> {

    E[] array;
    int sizeIn, sizeOut;

    public DoubleStackImpl(E[] array) {
        this.array = array;
        this.sizeIn = 0;
        this.sizeOut = ((array.length-1)/2);
    }

    @Override
    public void pushIn(E element) throws FullStructureException {
        if (!isFull() || sizeIn != ((array.length)-1)/2){
            array[sizeIn] = element;
            sizeIn++;
        }
        else{
            throw new FullStructureException("In Stack is Full.");
        }
    }

    @Override
    public E popIn() throws EmptyStructureException {
        if (isEmptyIn())
            throw new EmptyStructureException("In Stack is Empty.");
        sizeIn--;
        return array[sizeIn];
    }

    @Override
    public void pushOut(E element) throws FullStructureException {
        if (!isFull()){
            array[sizeOut] = element;
            sizeOut++;
        }
        else{
            throw new FullStructureException("Out Stack is Full.");
```

```
    }
}

@Override
public E popOut() throws EmptyStructureException {
    if (isEmptyOut())
        throw new EmptyStructureException("Out Stack is Empty.");
    sizeOut--;
    return array[sizeOut];
}

public boolean isEmpty() {
    return (sizeIn == 0 && sizeOut == (array.length/2));
}

@Override
public boolean isFull() {
    return (sizeIn == ((array.length/2)-1) && sizeOut == array.length-1);
}

@Override
public void display() {
    System.out.println("In Stack: ");
    for (int i=0;i<sizeIn;i++){
        System.out.print(array[i] +" ");
    }
    System.out.println("Out Stack: ");
    for (int j=array.length/2;j<sizeOut;j++){
        System.out.print(array[j] +" ");
    }
}

@Override
public E peekIn() throws EmptyStructureException {
    if (isEmptyIn())
        throw new EmptyStructureException("In Stack is Empty.");
    return array[sizeIn - 1];
}

@Override
public boolean isEmptyIn() {
    return (sizeIn == 0);
}

@Override
public E peekOut() throws EmptyStructureException {
```

```
        if (isEmptyOut())
            throw new EmptyStructureException("Out Stack is Empty.");
        return array[sizeOut - 1];
    }

    @Override
    public boolean isEmptyOut() {
        return (sizeOut == array.length/2);
    }
}
```

**AgesOfHollywood.java**

```
package sq_exercises;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import stacksandqueues.*;

public class AgesOfHollywood{

   String line = "";
   int count = 0;
   MyPriorityQueueIF<String> myQueue;
   HollywoodCelebrity hollywood = new HollywoodCelebrity();
   String[] array = new String[100];

   public AgesOfHollywood() {
      //this.array = array;
      myQueue = new MyPriorityQueueImpl<>(array);
   }

   public void parseTextFile(String pathname) {
      try{
         Scanner sc = new Scanner(new File(pathname));

         do{
            line = sc.nextLine();
```

```
            //line = line.replaceAll("\\n", "");
            hollywood.setFirstName(line.substring(0, line.indexOf(' ')));
            hollywood.setLastName(line.substring(line.indexOf(' '), line.lastIndexOf(' ')));

hollywood.setYearOfBirth(Integer.parseInt(line.substring(line.indexOf('1'),line.length()))
);
            myQueue.insert(hollywood.toString(), hollywood.getYearOfBirth());
        } while (sc.hasNextLine());
        myQueue.display();


    }
    catch(FileNotFoundException e){}
    catch(FullStructureException e){}
  }
}
```