FAKULTI TEKNOLOGI KEJURUTERAAN KELAUTAN DAN INFORMATIK

2022/2023

DATA STRUCTURE & ALGORITHM

Lab 5: Recursion, Linked List Queue & Deque

VERSION 1

STUDENT INFORMATION

PLEASE FILL IN YOUR DETAILS:

NAME: OMAR ISMAIL ABDJALEEL ALOMORY

MATRIC NUMBER:S6355

GROUP:K2

LAB:MP3

DATE:29/11/2022

TABLE OF CONTENTS

INSTRUCTIONS

Manual makmal ini adalah untuk kegunaan pelajar-pelajar Fakulti Teknologi Kejuruteraan Kelautan dan Informatik, Universiti Malaysia Terengganu (UMT) sahaja. Tidak dibenarkan mencetak dan mengedar manual ini tanpa kebenaran rasmi daripada penulis.

Sila ikuti langkah demi langkah sebagaimana yang dinyatakan di dalam manual.

This laboratory manual is for use by the students of the Faculty of Ocean Engineering Technology and Informatics, Universiti Malaysia Terengganu (UMT) only. It is not permissible to print and distribute this manual without the official authorisation of the author.

Please follow step by step, as described in the manual.

TASK 1: APPLY AND TEST THE IMPLEMENTATION OF RECURSION

OBJECTIVE

In this task, students must be able to:

- Apply the implementation recursion.
- Test the implementation

ESTIMATED TIME

[45 Minutes]

DEFINITION OF RECURSION

Recursion is a basic programming technique you can use in Java, in which a method calls itself to solve some problem. A method that uses this technique is recursive. Many programming problems can be solved only by recursion, and some problems that can be solved by other techniques are better solved by recursion.

One of the classic problems for introducing recursion is calculating the factorial of an integer. The factorial of any given integer — call it n so that you sound mathematical — is the product of all the integers from 1 to n. Thus, the factorial of 4 is 24: 4 x 3 x 2 x 1.

The recursive way to look at the factorial problem is to realize that the factorial for any given number n is equal to n times the factorial of n–1, provided that n is greater than 1. If n is 1, the factorial of n is 1.

This definition of factorial is recursive because the definition includes the factorial method itself. It also includes the most important part of any recursive method: an end condition. The end condition indicates when the recursive method should stop calling itself. In this case, when n is 1, it just returns 1. Without an end condition, the recursive method keeps calling itself forever.

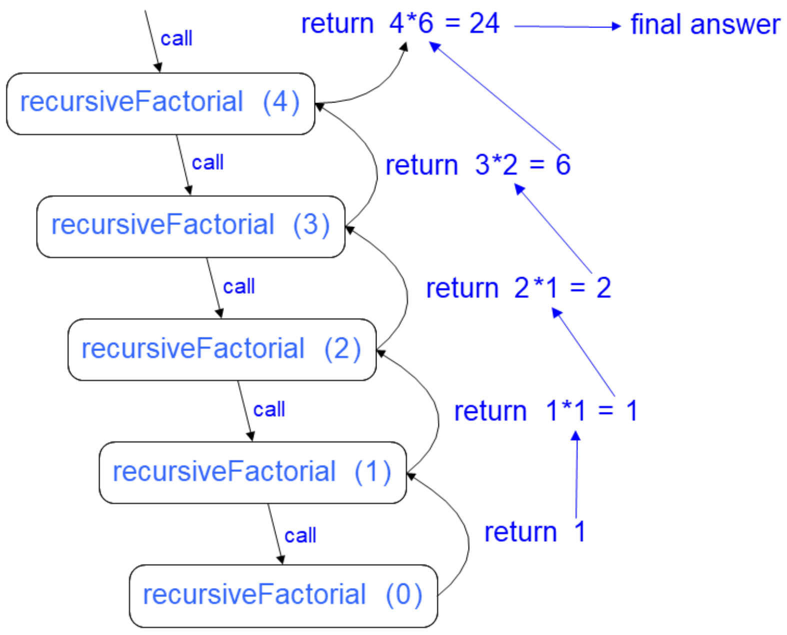


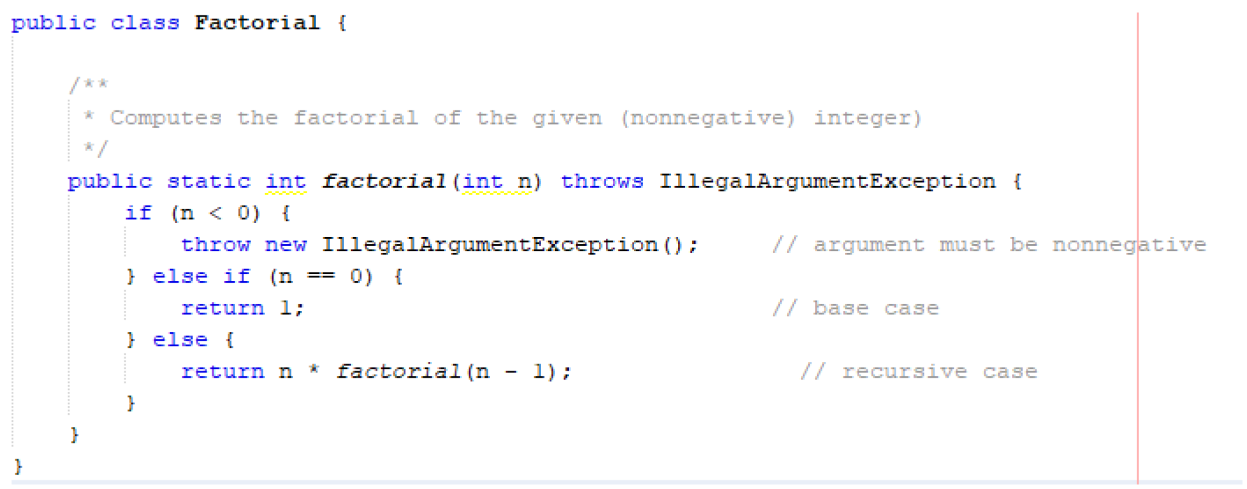
Figure 1: Recursion Trace for the factorial of 4

STEPS:

1. Open `Netbeans` and create new java application project.
2. Name your project as `RecursionExperiment` and click finish.
3. Change author profiles to :

```
a. Name :
b. Program: <put your program. E.g., SMSK(SE) or SMSK with
   IM
c. Course : CSF3104
d. Lab : <enter lab number>
e. Date : <enter lab date>
```

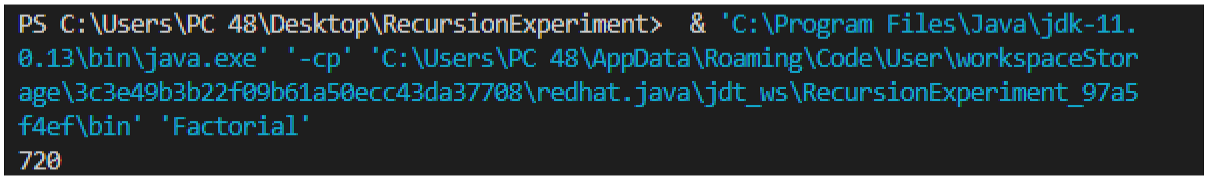
4. In the same `RecursionExperiment` project's package, create a new file named `Factorial.java` and insert the following codes:



```java
public class Factorial {

    /**
     * Computes the factorial of the given (nonnegative) integer)
     */
    public static int factorial(int n) throws IllegalArgumentException {
        if (n < 0) {
            throw new IllegalArgumentException();     // argument must be nonnegative
        } else if (n == 0) {
            return 1;                                 // base case
        } else {
            return n * factorial(n - 1);               // recursive case
        }
    }
}
```

5. Insert a main method in the same file to test the above codes. Put 6 as your input.

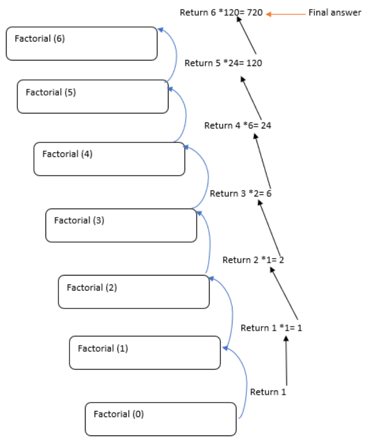
   Answer:



```
PS C:\Users\PC 48\Desktop\RecursionExperiment>  & 'C:\Program Files\Java\jdk-11.
0.13\bin\java.exe' '-cp' 'C:\Users\PC 48\AppData\Roaming\Code\User\workspaceStor
age\3c3e49b3b22f09b61a50ecc43da37708\redhat.java\jdt_ws\RecursionExperiment_97a5
f4ef\bin' 'Factorial'
720
```

6. Draw a recursion trace for an execution of your input.

   Hint: *You may draw using Ms Word/Ms Paint or draw it manually. Snap the photo of your drawing and upload it here.*



QUESTIONS

1. Where can we apply recursion in data structure?

   Answer:

Recusoin is used to to compute the factorial function, to determine whether a word is a palindrome, to compute powers of a number, to draw a type of fractal, and to solve the ancient Towers of Hanoi problem

2. Explain what is the meaning of 'base case' and 'recursive call'. What will happen when there is no base case in our recursive method?
   Answer:
   The base case, or halting case, of a function is the problem that we know the answer to, that can be solved without any more recursive calls. The base case is what stops the recursion from continuing on forever. For our example our base case is when n == 0 then it would stop there and return the value back to the top.

   A recursive call is one where procedure A calls itself or calls procedure B which then calls procedure A again. Each recursive call causes a new invocation of the procedure to be placed on the call stack. In our example we are calling the method factorial(n - 1) when it reach the base case it would stop calling it again.

TASK 2: IMPLEMENTING QUEUE USING LINKED LIST

OBJECTIVE

In this task, students will implement a queue using linked list as an alternative to array.

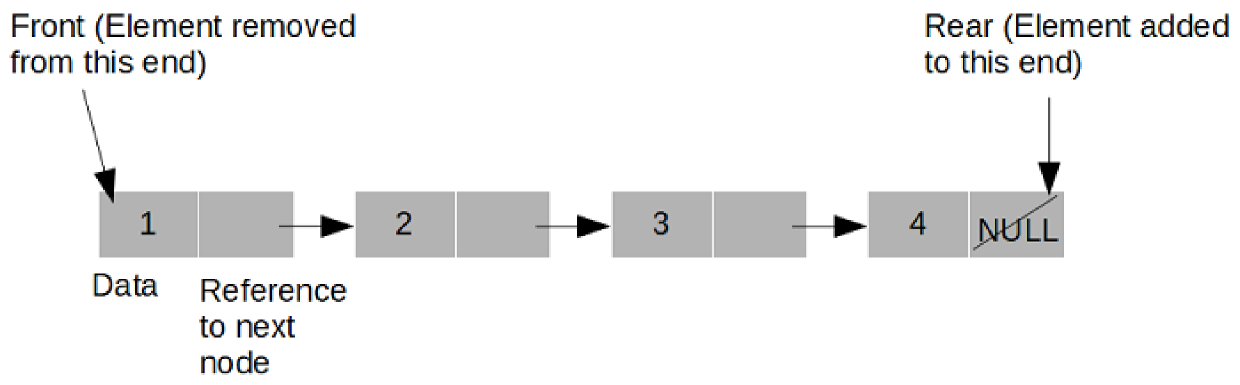
ESTIMATED TIME

[45 Minutes]

USING LINKED LIST FOR QUEUE

A queue is a First In First Out (FIFO) data structure where the first item inserted is the first to be removed. In a queue items are inserted at the rear and removed from the front of the queue.

Following image shows an implementation of Queue as linked list with front and rear references.


Front (Element removed from this end)
Rear (Element added to this end)
1
2
3
4
~~NULL~~
Data
Reference to next node

OPERATIONS IN QUEUE

As we know, there are three operations are implemented for a Queue:

insert: To insert an item at the rear of the queue.
remove: To remove an item from the front of the queue.
peek: Read value from the front of the queue without removing it.

STEPS:

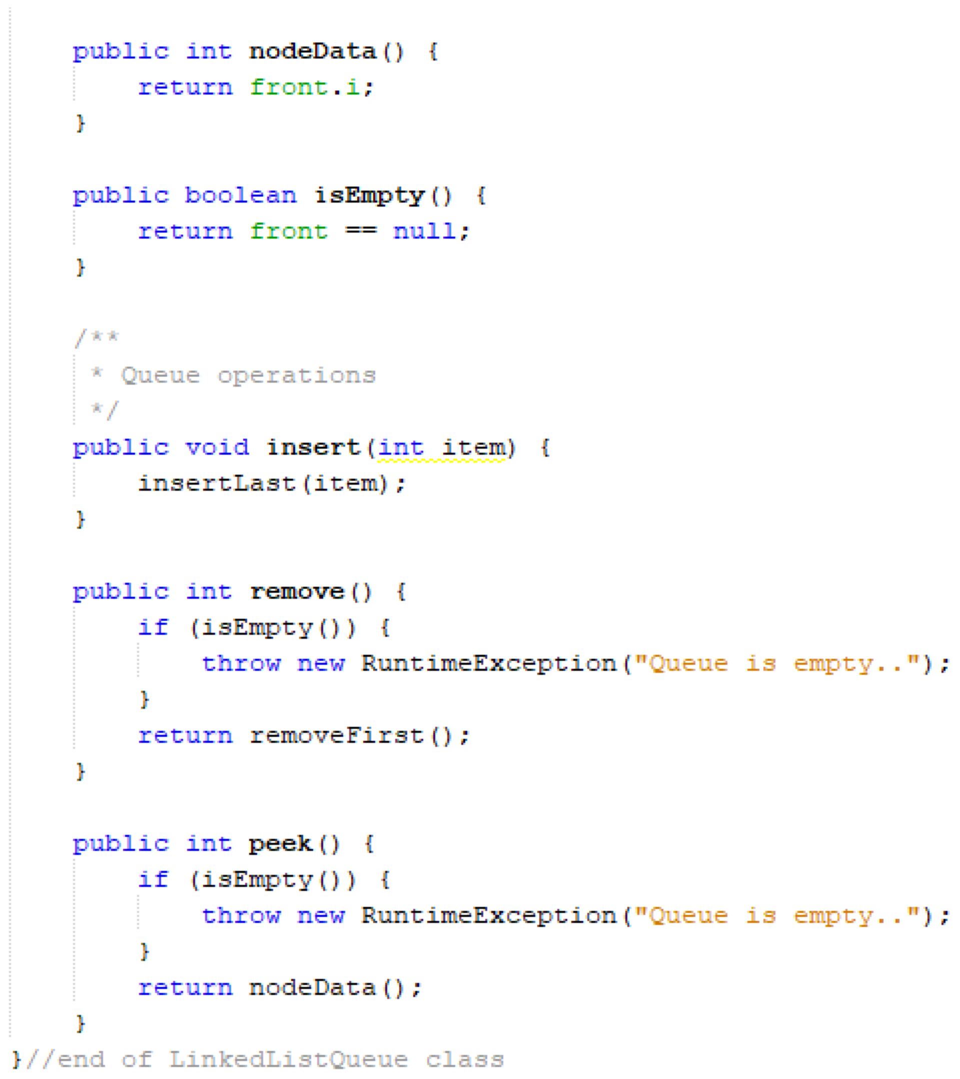
1. Open `Netbeans` and create new java application project.
2. Name your project as `LinkedListQExperiment` and click finish.
3. Change author profiles to :
```
a. Name :
b. Program: <put your program. E.g., SMSK(SE) or SMSK with
   IM
```

c. `Course : CSF3104`
d. `Lab : <enter lab number>`
e. `Date : <enter lab date>`

4. In the same `LinkedListQExperiment` project's package, create a new file named `LinkedListQueue.java` and insert the following codes:

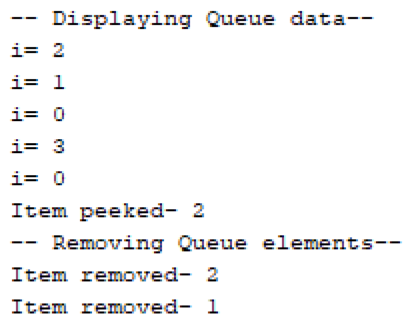
```java
public class LinkedListQueue {

    Node front;
    Node rear;

    public LinkedListQueue() {
        front = null;
        rear = null;
    }
    // Class for node

    private class Node {
        //data

        int i;
        Node next;

        Node(int i) {
            this.i = i;
        }

        public void displayData() {
            System.out.println("i= " + i);
        }
    }
```

```
/**
 * Linked list operations, keeping them separate from Queue operations
 *
 */
public void insertLast(int i) {
    Node newNode = new Node(i);
    if (isEmpty()) {
        front = newNode;
    } else {
        // previous last point to new node
        rear.next = newNode;
    }
    rear = newNode;
}

public int removeFirst() {

    int temp = front.i;
    // If no node left after deleting node
    if (front.next == null) {
        rear = null;
    }
    // front starts pointing to next element
    front = front.next;
    return temp;
}

// Method to traverse and display all nodes
public void displayList() {
    // Start from first node
    Node current = front;
    // loop till last node
    while (current != null) {
        current.displayData();
        current = current.next;
    }
}
```



```
    public int nodeData() {
        return front.i;
    }

    public boolean isEmpty() {
        return front == null;
    }

    /**
     * Queue operations
     */
    public void insert(int item) {
        insertLast(item);
    }

    public int remove() {
        if (isEmpty()) {
            throw new RuntimeException("Queue is empty..");
        }
        return removeFirst();
    }

    public int peek() {
        if (isEmpty()) {
            throw new RuntimeException("Queue is empty..");
        }
        return nodeData();
    }
}//end of LinkedListQueue class
```

5. Create a test class for the `LinkedListQueue.Use 2,1,0,3,0` as your input. Sample output:



```
-- Displaying Queue data--
i= 2
i= 1
i= 0
i= 3
i= 0
Item peeked- 2
-- Removing Queue elements--
Item removed- 2
Item removed- 1
```

6. Copy and paste your Java codes into text box below:

LinkedListQueue class

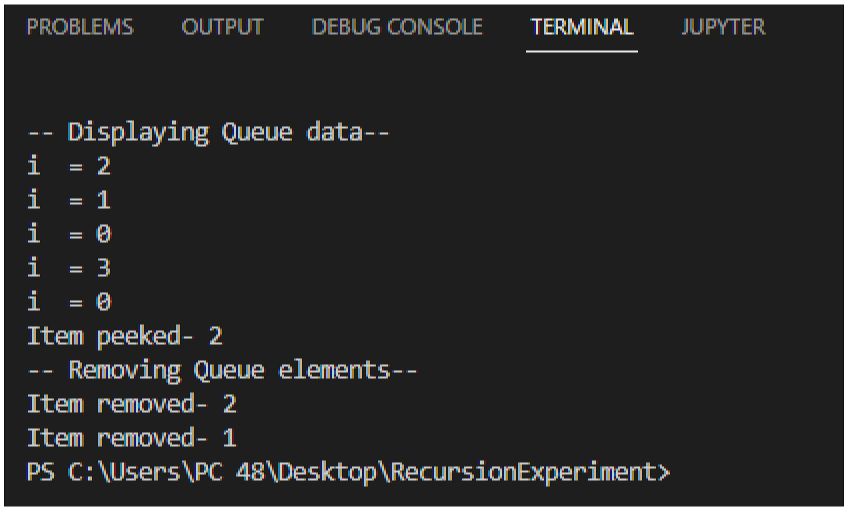
```
/*
 * Name: OMAR ISMAIL ALOMORY
 * Program: SMSK(KMA)
 * Course: CSF3013
 * Lab: MP3
 * Date: 29/11/2022
 */

public class LinkedListQueue {
    Node front, rear;

    public LinkedListQueue(){
        front = null;
        rear = null;
    }

    public class Node{
        int i ;
        Node next;

        Node(int i ){
            this.i = i;
        }
        public void displayData(){
            System.out.println("i  = "+ this.i);
        }
    }

    public void insertLast(int i ){
        Node newNode = new Node(i);
        if(isEmpty()){
            front = newNode;
        }else{
            rear.next = newNode;
        }
        rear =newNode;
    }
```

```
public int removeFirst(){
    int temp = front.i;
    if(front.next == null){
        rear = null;
    }
    front = front.next;
    return temp;
}

public void displayList(){
    Node current = front;
    while(current !=null){
        current.displayData();
        current = current.next;
    }
}

public int nodeData(){
    return front.i;
}
public boolean isEmpty(){
return front == null;
}

public void insert(int item){
    insertLast(item);
}

public int remove(){
    if(isEmpty()){
        throw new RuntimeException("Queue is empty...");
    }
    return removeFirst();
}
public int peek(){
    if(isEmpty()){
        throw new RuntimeException("Queue is empty...");
    }
    return nodeData();
}
```

```
}
```

LinkedListTest class

```
/*
 * Name: OMAR ISMAIL ALOMORY
 * Program: SMSK(KMA)
 * Course: CSF3013
 * Lab: MP3
 * Date: 29/11/2022
 */

public class LinkedListTest {
    public static void main(String[] args) {
        LinkedListQueue mine =  new LinkedListQueue();
        mine.insert(2);
        mine.insert(1);
        mine.insert(0);
        mine.insert(3);
        mine.insert(0);
        System.out.println("-- Displaying Queue data--");
        mine.displayList();
        System.out.println("Item peeked- "+mine.peek());

        System.out.println("-- Removing Queue elements--");
        System.out.println("Item removed- "+mine.remove());
        System.out.println("Item removed- "+mine.remove());
    }
}
```



```
-- Displaying Queue data--
i  = 2
i  = 1
i  = 0
i  = 3
i  = 0
Item peeked- 2
-- Removing Queue elements--
Item removed- 2
Item removed- 1
PS C:\Users\PC 48\Desktop\RecursionExperiment>
```

QUESTIONS

1. Describe the advantage(s) of using linked list for queue compared to array.
   Answer:
   Better use of Memory
   From a memory allocation point of view, linked lists are more efficient than arrays. Unlike arrays, the size for a linked list is not pre-defined, allowing the linked list to increase or decrease in size as the program runs.
   Fast Insertion/Deletion Time
   inserting a new node to the beginning or end of a linked list takes constant time (O(1)) as the only steps are to initialize a new node and then update the pointers. However, insertion/deletion to the middle of the linked list takes linear time (O(n)) as iteration over n elements is required to get to the correct location before inserting/deletion the node. On the other hand, in arrays, if we want to add an element at the beginning/end/middle of the array we need to shift the element to right/left in order to do the insertion, whereas deletion requires another mechanism, all of this can be applied if there is enough capacity otherwise we need to shift the array to another bigger array

2. Where can we apply queue in the software development?
   Answer:
   1. **Managing requests on a single shared resource such as CPU scheduling and disk scheduling.**
   2. **Handling hardware or real-time systems interrupts.**
   3. **Handling website traffic.**
   4. **Routers and switches in networking.**
   5. **Maintaining the playlist in media players.**

TASK 3: USING DEQUE BUILT-IN CLASS

OBJECTIVE

During this activity, students will apply double-ended queue (deque) in a simple programming task. Student should understand how deque works.

ESTIMATED TIME

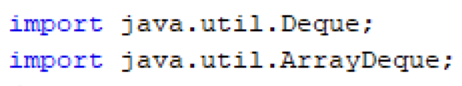
[30 Minutes]

INTRODUCTION

Java Deque Interface is a linear collection that supports element insertion and removal at both ends. Deque is an acronym for **"double ended queue".**

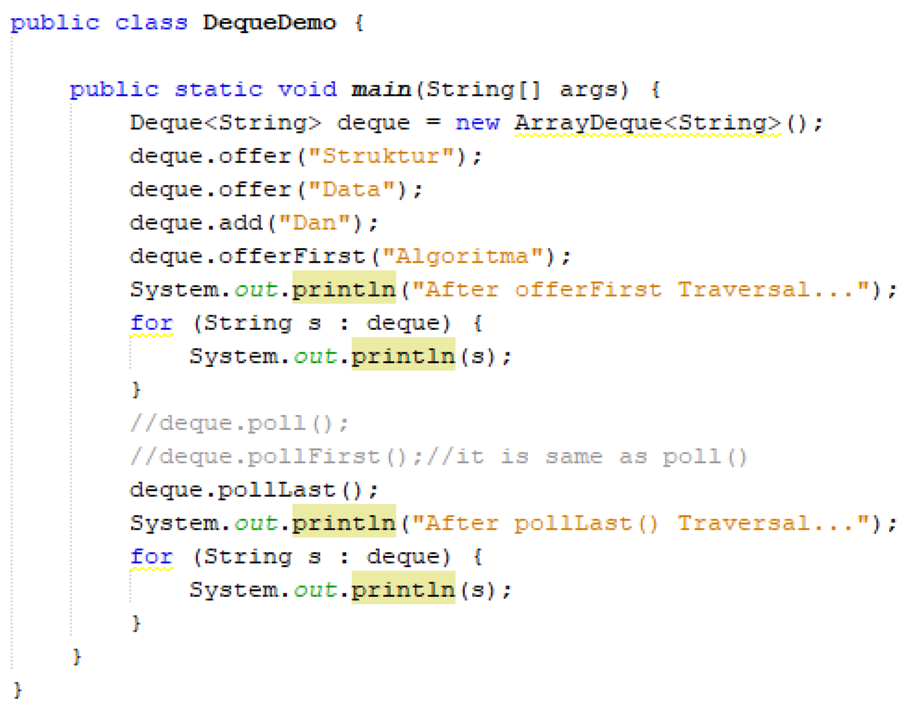
STEPS:

1. Open previously created `Netbeans` project.
2. Create a new class named `DequeDemo`.
3. In `DequeDemo.java`, import `Deque` and `ArrayDeque` class from `java.util` package:



```
import java.util.Deque;
import java.util.ArrayDeque;
```

4. Type the following codes into your Netbeans editor:



```
public class DequeDemo {

    public static void main(String[] args) {
        Deque<String> deque = new ArrayDeque<String>();
        deque.offer("Struktur");
        deque.offer("Data");
        deque.add("Dan");
        deque.offerFirst("Algoritma");
        System.out.println("After offerFirst Traversal...");
        for (String s : deque) {
            System.out.println(s);
        }
        //deque.poll();
        //deque.pollFirst();//it is same as poll()
        deque.pollLast();
        System.out.println("After pollLast() Traversal...");
        for (String s : deque) {
            System.out.println(s);
        }
    }
}
```

5. Save, compile and run the codes. Observe the output and explain how the below methods work:
    a. offer()
    b. add()
    c. offerFirst()
    d. pollLast()

    Answer:

    a. offer() to add a specific element at the end of the Deque.
    b. add() to add a specific element at the end of the Deque
    c. offerFirst() to insert a specific element at the front of this Deque
    d. pollLast() retrieve or fetch and remove the last element of the Deque

QUESTIONS

1. Explain the difference between linear queue and double-ended queue.

Answer:

In the case of a linear queue, the element added in the first position is going to be deleted in the first position. The order of operations performed on any element is fixed i.e, FIFO. A double-ended queue is a linear list in which all insertions and deletions are made at the end of the list. Therefore, more general than a stack or queue and is a sort of FLIFLO (first in last in or first out last out). A double-ended deque has two variants, viz., input restricted deque and output restricted deque.

TASK 4: PROGRAMMING ASSIGNMENT USING DEQUE

OBJECTIVE

To implement a PhoneBook class and create a simple program to save and retrieve a Phone Book records.

ESTIMATED TIME

[30 Minutes]

INSTRUCTIONS:

Given here is the structure of a class named PhoneBook:

PhoneBook
id: int name: String phoneNumber: String

1. Implement the above class using Java. Create a parameterized constructor that accepts three parameters, as described in the PhoneBook class diagram.
2. Next, create a demo class to insert and traverse records in the `PhoneBook`. Use Deque and ArrayQueue in your implementation.
3. Use the following dataset to test your program:
    a. Pendaftar UMT: 096684342
    b. Jabatan Pengurusan Akademi UMT: 096684219
    c. Fakulti Teknologi Kejuruteraan Kelautan dan Informatik: 096683220
4. **Bonus task**: Create a search function to retrieve a record from your PhoneBook program. As example, user enters "Pendaftar" as an input, and then your program will display "096684342" as the output.
5. Copy and paste your finished codes into the below text box:
    PhoneBook class

```
/**
 * @author Omar Alomory
 * Program: SMSK(KMA) K2
 * Course: CSF3013
 * Lab: MP3
 * Date: 29/11/2022
 */
```

```
package phonebook;

public class PhoneBook {

   private int Id;
   private String name;
   private String phoneNo;

   public PhoneBook() {}
   public PhoneBook(int id, String name, String phoneNo) {
         this.Id=id;
         this.name = name;
         this.phoneNo = phoneNo;
   }
   public String getName() {
         return name;
   }
   public void setName(String name) {
         this.name = name;
   }
   public String getPhoneNo() {
         return phoneNo;
   }
   public void setPhoneNo(String phoneNo) {
         this.phoneNo = phoneNo;
   }
   public String toString() {
         return this.name+": "+this.phoneNo;
   }
}
```

PhoneBookDemo class

```
/**
 * @author Omar Alomory
 * Program: SMSK(KMA) K2
 * Course: CSF3013
 * Lab: MP3
 * Date: 19/11/2022
 */
```

```
package phonebook;
import java.util.Deque;
import java.util.Scanner;
import java.util.ArrayDeque;
import java.util.InputMismatchException;

public class PhoneBookDemo {
    // simple method to display the menu and guide user in how to
search
   public static void menu() {
         System.out.println("Welcome here is the menu:");
         System.out.println("1_ for exit");
         System.out.println("2_ for search");
   }
   public static void main(String[] args) {
            // initializing Deque and inserting values to it.
         Deque <PhoneBook> deque = new ArrayDeque<PhoneBook>();
         deque.offer(new PhoneBook(1,"Pandafter UMT","096684342"));
         deque.offer(new PhoneBook(2,"Jabatan Pengurusan Akademi
UMT","096684219"));
         deque.offer(new PhoneBook(3,"Fakulti Teknologi Kejuruteraan
Kelautan dan Informatik","096683220"));

                // initializing scanners to get user input(Integer,
String)
         Scanner in = new Scanner(System.in);
         Scanner inString = new Scanner(System.in);

                // initializing the variables that would be used for
searching in 'phonebook'
         String answerInString = "";
         boolean check = true;
         int answer = 0 ,count = 0;

                // try to confirm of user input (Integer only)
                try{
         while(check) {
              menu();
              answer = in.nextInt();
              switch(answer) {
              case 1:  // if input is 1 terminate the program
                   check = false;
                   break;
              case 2: // if 2 do the searching

                   answerInString = "";
                   System.out.print("Please enter Name to search:");
                   answerInString = inString.nextLine();
                                count = 0; // to see if search 
values is found(would be used later)
                                System.out.println("----------------
------------------------");
                   for(PhoneBook e: deque) {
                                /* here we are 
converting(temporarily) the user input and the values in the 
                                    phonebook to lowercase efficient 
search and then comparing the values*/

   if(e.getName().toLowerCase().contains(answerInString.toLowerCase(
)) && 
                                   // although we have spaces in our 
phonebook list but it won't accept that

(!answerInString.isBlank())) {
                              System.out.println(e);
                                                count++;
                        }
                   }
                                // if the input is just empty string 
or no value is found
                   if(answerInString.isBlank() || count ==0) {
                        System.out.println("Sorry we could not find 
what u looking for...");
                                        System.out.println("--------
--------------------------------");
                   }
                                System.out.println("----------------
------------------------\n\n");
                   break;
```

```
                default:
                     System.out.println("Invalid input?");
                }

           }
           System.out.println("Thank you for using our Program...");
                  //terminate  the program if the input is not Integer
                  }catch(InputMismatchException n){
                             System.out.println("--------------------
--------------------");
                             System.out.println("Please Enter a
Number !!!");

                             System.out.println("--------------------
---------------------\n\n");
                         }
  }
}
```

Finally, read the instruction regarding submission carefully. Submit your answer using the link provided in Oceania UMT. Please ensure your codes are submitted to the correct group.