Sukkur IBA University
Department of Computer Science

DATA STRUCTURES
Lab04 – Queues and Stacks

Saif Hassan

READ IT FIRST

Prior to start solving the problems in these assignments, please give full concentration on following points.

1. WORKING – This is individual lab. If you are stuck in a problem contact your teacher, but, in mean time start doing next question (don't waste time).
2. DEADLINE – Mentioned on LMS
3. SUBMISSION – This assignment needs to be submitted in a soft copy.
4. WHERE TO SUBMIT – Please visit your LMS.
5. WHAT TO SUBMIT – .docx and pdf file.

KEEP IT WITH YOU!

1. Indent your code inside the classes and functions. It's a good practice!
2. It is not bad if you keep your code indented inside the loops, if and else blocks as well.
3. Comment your code, where it is necessary.

Read the entire question. Don't jump to the formula directly.

Read the entire question. Don't jump to the formula directly.

Task 1: Implementing a Stack using an Array

```java
// Create a class for implementing a stack using an array.
public class ArrayStack {

    // Constructor to initialize the stack with a given size.
    public ArrayStack(int size) {
        // ...
    }

    // Push an element onto the stack.
    public void push(int value) {
        // ...
    }

    // Pop and return the top element from the stack.
    public int pop() {
        // ...
    }

    // Check if the stack is empty.
    public boolean isEmpty() {
        // ...
    }

    // Peek at the top element of the stack without removing it.
    public int peek() {
        // ...
    }
}
```

Task 2: Testing the Array Stack

```java
// Create a class for testing the array-based stack.
public class ArrayStackTest {
    public static void main(String[] args) {
        // Create an instance of ArrayStack.

        // Push elements onto the stack.

        // Peek at the top element.

        // Pop elements from the stack and print them.
    }
}

```

Task 3: Implementing a Stack using a Linked List

```java
// Create a class for implementing a stack using a linked list.
public class LinkedListStack {

    // Push an element onto the stack.
    public void push(int value) {
        // ...
    }

    // Pop and return the top element from the stack.
    public int pop() {
        // ...
    }

    // Check if the stack is empty.
    public boolean isEmpty() {
        // ...
    }

    // Peek at the top element of the stack without removing it.
    public int peek() {
        // ...
    }
}

```

Task 4: Testing the Linked List Stack

```java
// Create a class for testing the linked list-based stack.
public class LinkedListStackTest {
    public static void main(String[] args) {
        // Create an instance of LinkedListStack.

        // Push elements onto the stack.

        // Peek at the top element.

        // Pop elements from the stack and print them.
    }
}

```

Task 7: Implement a Backward and Forward Navigation using Two Stacks

Design a simple web browser navigation system using two stacks: one for backward navigation and one for forward navigation. Create a class that allows users to navigate a hypothetical web browser history using the **back()** and **forward()** methods.

```java
public class WebBrowserNavigator {
    public WebBrowserNavigator() {
        // Initialize the two stacks for backward and forward navigation.
        // ...
    }

    public void visitPage(String url) {
        // Implement a method to visit a new page and update the stacks.
        // ...
    }

    public void back() {
        // Implement a method to go back in the browser history.
        // ...
    }

    public void forward() {
        // Implement a method to go forward in the browser history.
        // ...
    }

    public static void main(String[] args) {
        // Test the web browser navigation using the methods above.
        // ...
    }
}

```

Task 9: Implement a Min Stack

Design a stack that supports the **push, pop, top, and getMin** operations, all with O(1) time complexity. Create a class called **MinStack** that implements this special stack.

```
public class MinStack {
    // Implement the MinStack class with the specified operations.
    // ...
}

```

Task 10: Implementing a Queue using an Array

```java
public class ArrayQueue {
    private int[] queue;
    private int front;
    private int rear;
    private int capacity;

    public ArrayQueue(int capacity);

    public void enqueue(int item);

    public int dequeue();

    public boolean isEmpty();

    public boolean isFull();
}

```

Scenario 1: Basic Operations

```java
public static void main(String[] args) {
    ArrayQueue queue = new ArrayQueue(10);

    queue.enqueue(5);
    queue.enqueue(10);
    queue.enqueue(15);
    queue.enqueue(20);
    queue.enqueue(25);

    queue.dequeue();
    queue.dequeue();

    queue.enqueue(30);

    // Print elements remaining in the queue
    while (!queue.isEmpty()) {
        System.out.println(queue.dequeue());
    }
}

```

Scenario 2: Edge Cases

```java
public static void main(String[] args) {
    ArrayQueue queue = new ArrayQueue(3);

    queue.enqueue(1);
    queue.enqueue(2);
    queue.enqueue(3);

    // Try to enqueue one more item (should display an error message)
    queue.enqueue(4);

    // Dequeue all three items and display them
    while (!queue.isEmpty()) {
        System.out.println(queue.dequeue());
    }
}
```

```
```

Scenario 3: Empty Queue

```java
public static void main(String[] args) {
    ArrayQueue queue = new ArrayQueue(5);

    // Check if the queue is empty (it should be)
    System.out.println("Is the queue empty? " + queue.isEmpty());

    queue.enqueue(42);

    // Check if the queue is empty (it should not be)
    System.out.println("Is the queue empty? " + queue.isEmpty());

    // Dequeue the item
    int item = queue.dequeue();

    // Check if the queue is empty again (it should be)
    System.out.println("Is the queue empty? " + queue.isEmpty());
}

```

Task 11: Apply LinkedList to Task10 and use your LinkedList class to implement queue

Task 12 (Print Spooling): Print Spooling use queues to manage print jobs in a print server. Print jobs are placed in a queue and printed one by one to avoid overloading the printer.

import java.util.Queue;
import java.util.LinkedList;

```java
class PrintSpooler {
    private Queue printQueue = new Queue<>(); // Queue class which you designed.

    public void addPrintJob(String document);

    public void print();

    public boolean isEmpty();

    public int size();
}

```

<u>Main Class</u>

```java
public class PrintSpoolingMain {
    public static void main(String[] args) {
        PrintSpooler spooler = new PrintSpooler();

        // Scenario 1: Add print jobs to the spooler
        spooler.addPrintJob("Document 1");
        spooler.addPrintJob("Document 2");
        spooler.addPrintJob("Document 3");

        System.out.println("Print Queue Size: " + spooler.size());

        // Scenario 2: Simulate printing
        while (!spooler.isEmpty()) {
            spooler.print();

            // Simulate printing time
            try {
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }

        // Scenario 3: Handling an empty queue
        spooler.print(); // Should print "Print queue is empty."

        System.out.println("All print jobs have been processed.");
    }
}

```