Experiment Report: Implementing Stacks and Queues in Java

name: Jiongyi Liao
student number: 1220011562

Experiment One: Custom Stack Implementation

In the first experiment, we implemented a custom stack (`MyStack`) using two queues to simulate stack operations. The stack supports push (pushing elements onto the stack), pop (removing elements from the stack), top (viewing the top element of the stack), and empty (checking if the stack is empty) operations. This experiment successfully demonstrated basic stack operations.

- Result:
  Push: 1, 2, 3
  Pop: 3, 2, 1
  Is Stack Empty: true

Experiment Two: Custom Circular Queue Implementation

In the second experiment, we implemented a custom circular queue (`Queue`), supporting enqueue (adding elements to the queue), dequeue (removing elements from

the queue), peek (viewing the front element of the queue), isEmpty (checking if the queue is empty), and isFull (checking if the queue is full) operations. This experiment showcased fundamental circular queue operations.

- Result:
  Enqueue: 10, 20, 30, 40, 50
  Dequeue: 10, 20, 30
  Is Queue Empty: false
  Is Queue Full: false

Experiment Three: Dynamic Stack Implementation with Automatic Expansion

In the third experiment, we implemented a dynamic stack (`DynamicStack`) that automatically expands its capacity when full. The stack supports push (pushing elements onto the stack), pop (removing elements from the stack), peek (viewing the top element of the stack), isEmpty (checking if the stack is empty), and size (getting the size of the stack) operations. This experiment demonstrated dynamic stack operations with automatic capacity expansion.

- Result:
  Push: 10, 20, 30
  Pop: 30, 20, 10
  Is Stack Empty: true

Experiment Four: Static Sized Stack Implementation

In the fourth experiment, we implemented a static-sized stack (`StaticStack`) where the stack size is fixed, and new elements cannot be pushed when the stack is full. The stack supports push (pushing elements onto the stack), pop (removing elements from the stack), peek (viewing the top element of the stack), isEmpty (checking if the stack is empty), and isFull (checking if the stack is full) operations. This experiment showcased operations on a stack with a static size.

- Result:
  Push: 10, 20, 30, 40, 50
  Is Stack Full: true
  Top Element: 50
  Pop: 50, 40, 30, 20, 10

Is Stack Empty: true

Code:

```
1.package h3;

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

class MyStack {
    private Queue<Integer> queue;

    public MyStack() {
        queue = new LinkedList<>();
    }

    public void push(int x) {
        int size = queue.size();
        queue.offer(x);
        for (int i = 0; i < size; i++) {
            int element = queue.poll();
            queue.offer(element);
        }
    }

    public int pop() {
        return queue.poll();
    }

    public int top() {
        return queue.peek();
    }

    public boolean empty() {
        return queue.isEmpty();
    }
}

public class Main {
    public static void main(String[] args) {
        MyStack stack = new MyStack();
        stack.push(1);
        stack.push(2);
        stack.push(3);
```

```
            System.out.println(stack.pop());    // Output: 3
            System.out.println(stack.pop());    // Output: 2
            System.out.println(stack.pop());    // Output: 2


            System.out.println(stack.empty());    // Output: false
        }
}               2.package h3;

class Queue {
        private int[] arr;
        private int front;
        private int rear;
        private int size;

        public Queue(int capacity) {
                arr = new int[capacity];
                front = 0;
                rear = -1;
                size = 0;
        }

        public void enqueue(int item) {
                if (isFull()) {
                        System.out.println("Queue is full. Cannot enqueue item.");
                        return;
                }

                rear = (rear + 1) % arr.length;
                arr[rear] = item;
                size++;
        }

        public int dequeue() {
                if (isEmpty()) {
                        System.out.println("Queue is empty. Cannot dequeue item.");
                        return -1;
                }

                int item = arr[front];
                front = (front + 1) % arr.length;
                size--;
                return item;
```

```
    }

    public int peek() {
        if (isEmpty()) {
            System.out.println("Queue is empty. Cannot peek item.");
            return -1;
        }

        return arr[front];
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public boolean isFull() {
        return size == arr.length;
    }

    public int size() {
        return size;
    }
}

public class duilie {
    public static void main(String[] args) {
        Queue queue = new Queue(5);

        queue.enqueue(10);
        queue.enqueue(20);
        queue.enqueue(30);

        System.out.println("Size of the queue: " + queue.size());
        System.out.println("Front item: " + queue.peek());

        System.out.println("Dequeued item: " + queue.dequeue());
        System.out.println("Dequeued item: " + queue.dequeue());

        System.out.println("Size of the queue: " + queue.size());
        System.out.println("Front item: " + queue.peek());

        queue.enqueue(40);
        queue.enqueue(50);
```

```
        System.out.println("Size of the queue: " + queue.size());
        System.out.println("Front item: " + queue.peek());

        System.out.println("Dequeued item: " + queue.dequeue());
        System.out.println("Dequeued item: " + queue.dequeue());
        System.out.println("Dequeued item: " + queue.dequeue());

        System.out.println("Size of the queue: " + queue.size());
        System.out.println("Front item: " + queue.peek());
    }
}           3.package h3;

public class DynamicStack {
    private int capacity;
    private int[] stack;
    private int top;

    public DynamicStack() {
        capacity = 10;
        stack = new int[capacity];
        top = -1;
    }

    public void push(int item) {
        if (top == capacity - 1) {
            expandStack();
        }
        stack[++top] = item;
    }

    public int pop() {
        if (top == -1) {
            throw new IllegalStateException("Stack is empty");
        }
        return stack[top--];
    }

    public int peek() {
        if (top == -1) {
            throw new IllegalStateException("Stack is empty");
        }
        return stack[top];
    }
```

```
    public boolean isEmpty() {
        return top == -1;
    }

    public int size() {
        return top + 1;
    }

    private void expandStack() {
        int newCapacity = capacity * 2;
        int[] newStack = new int[newCapacity];
        System.arraycopy(stack, 0, newStack, 0, capacity);
        stack = newStack;
        capacity = newCapacity;
    }
    public static void main(String[] args) {
        DynamicStack stack = new DynamicStack();

        stack.push(10);
        stack.push(20);
        stack.push(30);

        System.out.println("Size of stack: " + stack.size());
        System.out.println("Top element: " + stack.peek());

        while (!stack.isEmpty()) {
            System.out.println("Popped element: " + stack.pop());
        }

        System.out.println("Size of stack: " + stack.size());
    }
}
```

4.

```
package h3;


public class StaticStack {
    private int maxSize;
    private int[] stackArray;
    private int top;

    public StaticStack(int size) {
        maxSize = size;
        stackArray = new int[maxSize];
        top = -1;
```

```
}

public void push(int value) {
    if (top == maxSize - 1) {
        System.out.println("Stack is full. Cannot push element.");
        return;
    }
    stackArray[++top] = value;
}

public int pop() {
    if (top == -1) {
        System.out.println("Stack is empty. Cannot pop element.");
        return -1;
    }
    return stackArray[top--];
}

public int peek() {
    if (top == -1) {
        System.out.println("Stack is empty. Cannot peek element.");
        return -1;
    }
    return stackArray[top];
}

public boolean isEmpty() {
    return (top == -1);
}

public boolean isFull() {
    return (top == maxSize - 1);
}

public static void main(String[] args) {
    StaticStack stack = new StaticStack(5);

    stack.push(10);
    stack.push(20);
    stack.push(30);
    stack.push(40);
    stack.push(50);

    System.out.println(stack.isFull());
```

```
            System.out.println(stack.peek());

            while (!stack.isEmpty()) {
                System.out.println(stack.pop());
            }

            System.out.println(stack.isEmpty());
        }
}
```

Conclusion:

Through these experiments, we gained practical insights into implementing stacks and queues in Java. Understanding custom stack implementations, circular queues, dynamic stack resizing, and static-sized stacks is crucial for mastering data structures and algorithms in Java programming. These experiments provide a foundational understanding of these essential data structures, preparing us for more advanced programming challenges.