Data Structure Project Report

Name: Lu Jingyu

UCD: 20205767

BJUT: 20372216

Class:SE-2

Project_2_A_1

Stack using array

```
public class Project_2_A_1{
    public static class ArrayStack{
        private int maxSize;
        private  int[] values;
        private int top;

        public ArrayStack(int size){
            maxSize=size;
            values=new int[size];
            top=0;
        }
        public int size() { return top; }
        public boolean isEmpty() { return top==0; }
        public int top() { return values[top-1]; }
        public void push(int i){
            if (top<maxSize){
                values[top]=i;
                top=top+1;
            }
        }
        public int pop(){
            if (isEmpty()){
                throw new RuntimeException("Stack empty");
            }
            int i=values[top-1];
            top=top-1;
            return i;
        }
        public static void main(String[] args){
            ArrayStack stack=new ArrayStack( size: 7);
            int i=0;
            int j=0;
            int k=0;
            int[] arr={4,5,6,7,12,23,45};
            while (i<7){
                stack.push(arr[i]);
                System.out.println(stack.top());
                i++;
            }
            while (j<3){
                System.out.println(stack.size());
                stack.pop();
                j++;
            }
            stack.push( i: 65);
            while (k<7){
                stack.pop();
                k++;
            }
        }
    }
```

Method Explanation:

push(i): Insert i at the top of the stack

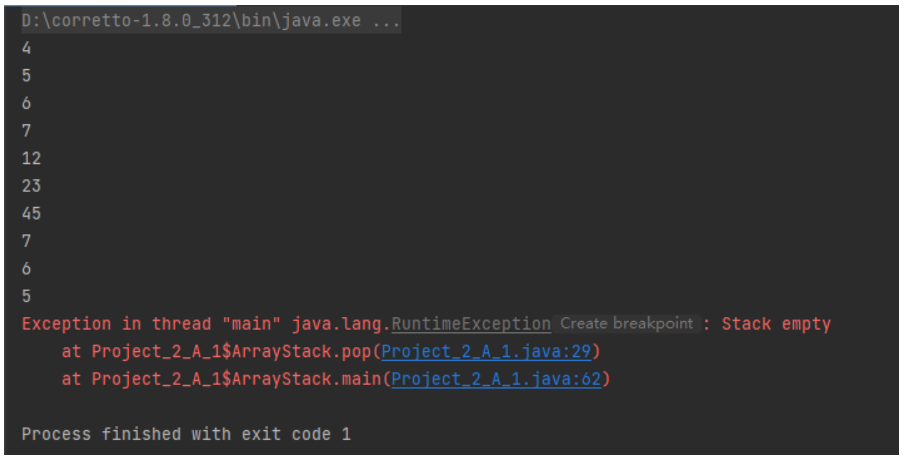
pop(): Removes the object at the top of the stack and returns it

top(): Returns the object at the top of the stack

size(): Returns the number of elements currently stored on the stack

isEmpty(): Determine if there are elements in the stack

Output



If pop exceeds the stack length, an error is thrown

Challenge:

The stack is characterized by first in and last out，it was a challenge for me to understand how stacks worked, not like the old arrays, which could be deleted and added at any time, but strictly first in and last out. Once fully understood, the code implementation is not difficult

Stack using linked list

```java
public static class LinkStack {
    Node top;
    int size;
    class Node {
        private int data;
        private Node next;
        public Node(int s) { data=s; }
    }


    public void push(int s){
        Node node = new Node(s);
        node.next = top;
        top = node;
        size = size + 1;
    }

    public int pop(){
        if (top==null){
            throw new RuntimeException("Stack empty");
        }
        int temp = top.data;
        top = top.next;
        size = size-1;
        return temp;
    }
```

Method Explanation:

push(i): Push elements onto the stack

1. Make the top reference point to the new node, and the next of the new node point to the old top
2. Size + 1 for the number of elements in the stack

pop(): Pop stack element

1. The top reference refers to the next element (top.next) on the top of the original stack and frees the reference to the original top element
2. Size-1 records the number of elements in the stack

```
    public static void main(String[] args) {
        LinkStack linkedStack = new LinkStack();
        int[] arr={4,5,6,7,12,23,45};
        int i=0;
        while (i<7){
            linkedStack.push(arr[i]);
            System.out.println("linkedStack1 "+linkedStack.top.data);
            i++;
        }
        int j=0;
        while (j<3){
            linkedStack.pop();
            j++;
        }
        linkedStack.push( s: 65);
        int k=0;
        while (k<7){
            linkedStack.pop();
            System.out.println("linkedStack2 "+linkedStack.top.data);
            k++;
        }
    }
}
```

Output

```
D:\corretto-1.8.0_312\bin\java.exe ...
linkedStack1 4
linkedStack1 5
linkedStack1 6
linkedStack1 7
linkedStack1 12
linkedStack1 23
linkedStack1 45
Exception in thread "main" java.lang.RuntimeException Create breakpoint : Stack empty
    at Project_2_A_1$LinkStack.pop(Project_2_A_1.java:79)
    at Project_2_A_1$LinkStack.main(Project_2_A_1.java:103)

Process finished with exit code 1
```

If pop exceeds the stack length, an error is thrown

Challenge:

Understanding the role of nodes in a stack is difficult because elements are not added and removed directly in a stack, but by changing the direction of the node

Project_2_A_2

queue

```java
public class LinkQueue {
    static class Node {
        public int data;
        public Node next;
        public Node(int data) {
            this.data = data;
        }
    }
    static class SingleLinkedList{
        private Node front;
        private Node rear;
        private int size;

        public void enqueue(int i){
            Node n = new Node(i);
            if(size==0){
                front = n;
            }
            else{
                rear.next = n;
            }
            rear = n;
            size = size + 1;
        }

        public int dequeue(){
            if (front==null){
                throw new RuntimeException("Queue empty");
            }
            int i = front.data;
            front = front.next;
            size = size - 1;
            return i;
        }

    }
```

Method Explanation:

enqueue(i): adding element
When joining a team, the rear moves back one place

dequeue(): remove the element
When joining a team, the front moves back one place

```java
public static void main(String[] args) {
    SingleLinkedList q=new SingleLinkedList();
    int[] arr={4,5,6,7,12,23,45};
    int i=0;
    while (i<7){
        q.enqueue(arr[i]);
        System.out.println("LinkedList1 "+q.rear.data);
        i++;
    }
    int j=0;
    while (j<3){
        q.dequeue();
        j++;
    }
    q.enqueue( i: 65);
    int k=0;
    while (k<7){
        q.dequeue();
        k++;
    }
```

Output:

```
D:\corretto-1.8.0_312\bin\java.exe ...
LinkedList1 4
LinkedList1 5
LinkedList1 6
LinkedList1 7
LinkedList1 12
LinkedList1 23
LinkedList1 45
Exception in thread "main" java.lang.RuntimeException Create breakpoint : Queue empty
    at LinkQueue$SingleLinkedList.dequeue(LinkQueue.java:28)
    at LinkQueue.main(LinkQueue.java:54)
```

Double Linked List

```java
public class doubleList {
    private Node first;
    private Node last = first;
    class Node {
        private Object element;
        Node next;
        Node pre;
        public Node(Object e) { this.element=e; }
        public Object element() { return element; }
    }

    public Node getFirst() { return first; }


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

    public void insertFirst(Object d) {
        Node node = new Node(d);
        if (isEmpty()) {
            last=node;
        } else {
            first.pre = node;
        }
        node.next = first;
        first = node;
    }
```

Method Explanation:

getfirst(): Return the reference that is stored in the variable first

isEmpty(): Check if there are any elements in the list

insertFirst(d): Insert the value into the first position

1. Change the next reference in n to point to the first
2. Change the previous reference in first to point to n
3. Change the first reference to point to n

```java
public void insertLast(Object d) {
    Node node = new Node(d);
    if (isEmpty()) {
        first = node;
    } else {
        node.pre = last;
        last.next = node;
    }
    last = node;
}

public void insertAfter(Object p, Object d) {
    Node node = new Node(d);
    Node temp = first;
    while ((temp != null) && (temp.element() != p)) {
        temp = temp.next;
    }
    if (temp == null) {
        if (isEmpty()) {
            first = node;
            last = node;
        } else {
            last.next = node;
            node.pre = last;
            last = node;
        }
    } else {
        if (temp == last) {
            node.next = null;
            last = node;
        } else {
            node.next = temp.next;
            temp.next.pre = node;
        }
        temp.next = node;
        node.pre = temp;
```

insertLast(d): Insert the value into the last position

1. change the previous reference to n to point to last
2. change the next reference of last to point to n
3. change the last reference to point to n

insertAfter(p, d): Insert the value into the position after

1. convert p to node
2. change the next reference in n to point to the next reference of P
3. change the previous reference in n to point to P
4. change the next reference of P to point to n
5. change the next point of P to point to n

```java
public void insertBefore(Object p, Object d) {
    Node node = new Node(d);
    Node temp = last;
    while ((temp != null) && (temp.element() != p)) {
        temp = temp.next;
    }
    if (temp == null) {
        if (isEmpty()) {
            last = node;
            first = node;
        } else {
            first.pre = node;
            node.next = first;
            first = node;
        }
    } else {
        if (temp == first) {
            node.pre = null;
            first = node;
        } else {
            node.pre = temp.pre;
            temp.pre.next = node;
        }
        temp.pre = node;
        node.next = temp;
    }
}
```

insertBefore(p, d): Insert the value into the position before

1. convert p to node
2. change the previous reference in n to point to the previous reference of P
3. change the next reference in n to point to P
4. change the next reference before p to point to n
5. change the previous value of P to point to n
6.

```java
public Node remove(Object p) {
    Node temp = first;
    while ((temp != null) && (temp.element() != p)) {
        temp = temp.next;
    }
    if (temp == null) {
        return null;
    } else {
        if (temp == first) {
            first = temp.next;
            temp.next.pre = null;
        } else if (temp == last) {
            last = temp.next;
            temp.next.pre = null;
        } else {
            temp.pre.next = temp.next;
            temp.next.pre = temp.pre;
        }
    }
    return temp;
}
```

Remove (P): removes the object P from the list

1. copy the data from position P to variable D
2. convert p to node
3. change the previous position of P to point to the next position of P
4. change the previous one of P to point to the previous one of P
5.

```
    public void print(){
        Node temp = first;
        while(temp != null){
            System.out.print(temp.element+",");
            temp = temp.next;
        }
        System.out.println();
    }
    public static void main(String[] args) {

        doubleList doubledList= new doubleList();
        doubledList.insertFirst( d: 111);
        doubledList.insertLast( d: 1);
        doubledList.insertBefore( p: 1, d: "aa");
        doubledList.insertAfter( p: 1, d: "aa");
        doubledList.remove( p: 1);
        doubledList.print();

    }
}
```

Output

```
D:\corretto-1.8.0_312\bin\java.exe ...
111,aa,aa,

Process finished with exit code 0
```

Challenge:

The challenge lies in the realization of insertBefore(p, d). I realized this method only after I sorted out the relationship between pre and next of the two adjacent nodes by drawing a picture on the paper. InsertAfter (p, d) is the same as insertBefore(p, d) but in reverse order

Project 2_C

double-ended queue

```
public class linkDeque implements Deque{
    private Node first;
    private Node last;
    private int size;


    class Node {
        private Object element;
        Node next;
        Node pre;
        public Node(Object e){
            this.element=e;
        }
        public Object element(){
            return element;
        }
    }

    @Override
    public void addFirst(Object o) {
        Node n= new Node(o);
        if (isEmpty()){
            first=n;
            last=n;
        }else{
            first.pre=n;
            n.next=first;
            first=n;

        }
        size=size+1;
    }
```

Method Explanation:

addFirst (o): Inserts a value into the first of the queue

1. Change the previous reference to first to n
2. Change the next reference of n to first
3. Change the first reference to point to n

```
public void addLast(Object o) {
    Node n= new Node(o);
    if (isEmpty()){
        last=n;
        first=n;
    }else{
        last.next=n;
        n.pre=last;
        last=n;
    }
    size=size+1;
}

@Override
public Object removeFirst() throws EmptyDequeException {
    if (isEmpty()){
        throw new EmptyDequeException();
    }
    Object o=first.element();
    first=first.next;
    if (size==1){
        first=null;
        last=null;
    }else{
        first.pre=null;
    }
    size=size-1;
    return o;
}
```

addLast(o): Inserts the value at the end of the queue

1. Change the next reference of last to n
2. Change the prev reference of n to last
3. Change the last reference to point to n

removeFirst (): Removes the next object from the front of the queue

1. Copy element reference from first as o
2. Change the first reference to the next reference of first
3. Change the previous reference to first to null

```java
public Object removeLast() throws EmptyDequeException {
    if (isEmpty()){
        throw new EmptyDequeException();
    }
    Object o=last.element();
    last=last.pre;
    if (size==1){
        first=null;
        last=null;
    }else{
        last.next=null;
    }
    size=size-1;
    return o;
}
```

removeLast (): Deletes the next object from behind the queue

1. Copy the previous element reference to o
2. Changes the last reference to the previous reference to the last
3. Change the next reference to last to null

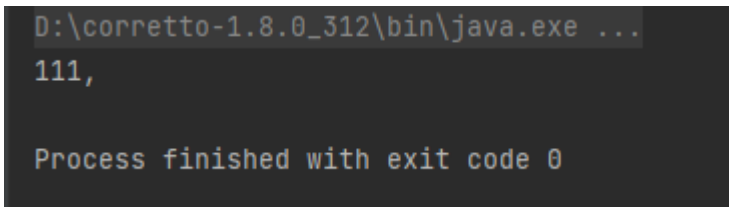
```java
public void print(){
    Node temp = first;
    while(temp != null){
        System.out.print(temp.element+",");
        temp = temp.next;
    }
    System.out.println();
}
public static void main(String[] args) {

    linkDeque d=new linkDeque();

    d.addFirst( o: "111");
    d.addLast( o: 1);
    d.addFirst( o: 2);
    d.removeFirst();
    d.removeLast();
    d.print();

}
```

Output:



Challenge:

The challenge is to understand what a two terminal queue is. A two terminal queue supports inserting and removing elements at both ends. When I understand a two terminal queue, I regard it as a one-way queue with head and tail connected.
Another challenge is how to implement removeFirst () and removeLast (), which takes into account special cases such as shih with only or no elements in the queue