Project 2 Data Structure & Algs

project2 -A

Question 1

requirement: implement stack using array and list

1. Stack Abstract Data Type (Stack ADT)

Be familiar with Stack ADT and know the basic principles before writing code to solve the problem

Stack is the easiest ADT type

- A stack is a container for items
- insertion and remove the data from this type is based on **last-in-first-out (LIFO)** principle which means that the data that put in the last will put out in the first
- some special terminology for operation in a stack
    - Item are **pushed onto** the stack (insertion)
    - Item are **popped off** of the stack (removal)
    - the top item is the last item that we pushed onto the stack

2. Implement stack using array

Array based implementation

1. data store in an array
2. need remember which index store the top item
3. **disadvantage**: because of the array size, finite capacity

Implement stack using array we need implement some method, for using array is need the following methods:

1. `public void push(int data)` - push integer *data* onto the stack (actually all objects are available, here integers are used)
2. `public int pop()` - pop the integer off the stack, follow the LIFO principle
3. `public size()` - return the number of items that store in the stack
4. `public boolean isEmpty()` - check if the stack is empty

The most important part is **push** method and **pop** method.

2.1 push method

Put the data into the array, when can't push more item into the array, it will print an error to show that can't put anymore item in this array.

```
// because of the array limitation, need set the max size of array
public void push(int data){
    if (top < maxSize){
        values[top] = data;
        top++;
    } else {
        // when the array is full, can't ues push method
        System.err.println("Error to push");
        System.exit(1);
    }
    System.out.println(data + " is pushed ont the stack");
}
```

2.2 pop method

Take out the last item put into the array, if the array is empty, it will print an error to show that can't take out anymore item from this array.

```
public int pop(){
    // when the stack is empty, can't use the pop method
    if (isEmpty()){
        System.err.println("Error to pop");
        System.exit(1);
    }
    int i = values[top - 1];
    top--;
    System.out.println(i + " is popped off of the stack");
    return i;
}
```

2.3 print method

```
public void printArray(int number){
    if (number == size()){
        return;
    }
    printArray(number + 1);
    System.out.print(values[number] + " ");
}
```

---

3. Node class

use Node object to store the data, the Node class is following:

Node object can store the input data, and the next Node object, which **make all Nodes linkable**.

```
// vavriable in Node class
// store the data
private int data;
// a node to store a reference to the next node in the stack
private Node next;
// the Node class
public class Node {
    int data;
    Node next;
    public Node(int data){
        this.data = data;
    }
}
```

4. Implement stack using linked list

Link based implementations

1. data are stored in custom object called **nodes**
2. object references are used to keep track of the order of the item
3. **advantage**: infinite capacity

The method use linked list is same as using array, is also need the following methods:

but something **different** is that store the data into Node class not array.

1. `public void push(int data)` - create a new Node class to store the data, and **associate the newly joined Node with the last joined Node.**
2. `public int pop()` - pop the data that store in the last joined off , and follow the LIFO principle
3. `public size()` - return the number of items that store in the stack
4. `public boolean isEmpty()` - check if the stack is empty

Different to the using array is that using push and pop method, first need to create a new Node object.

2.1 push method

```
public void push(int data){
    // create a new Node object to store the data
    Node n = new Node(data);
    // associate the newly joined Node with the last joined Node
    n.next = top;
    top = n;
    size++;
    System.out.println(data + " is pushed onto the stack");
}
```

2.2 pop method

```
public int pop(){
    // check if the stack is empty, if is empty can't use pop method and print an error to tell that the stack is empty
    if (isEmpty()){
        System.err.println("Error to pop because this stack is empty");
        System.exit(1);
    }
    // use data to store the data that store in the Node object
    int data = top.data;
    top = top.next;
    size--;
    System.out.println(data + " is popped off of the stack");
    return data;
}
```

2.3 print method

```
public void printLinkList(Node node) {
    if (node == null) {
        return;
    }
    printLinkList(node.next);
    System.out.print(node.data + " ");
}
```

Question 2

requirement: Implement queue using linked list

1. The Queue Abstract Data Type (Queue ADT)

- similar to the Stack abstract data type
- follow the first-in-first-out (**FIFO**) principle
    - which means that when you remove one item, this item is the item that stay longest time in the queue
- some special terminology
    - **enqueued** (insertion)
    - **dequeued** (removal)
    - the **front** of the queue is the next item to be dequeued
    - the **rear** of the queue is the last item that was dequeued

2. Implement queue using linked list

2.1 Queue interface

```
public interface Queue {
    public void enqueue (int element);
    public int dequeue();
    public int front();
    public int size();
    public boolean isEmpty();
}
```

the queue ADT have the following methods:

1. `enqueue(i)` : same as push
2. `dequeue`: same as pop (but in this way, removed the front and return it. which different with pop)
3. `front()`: same as top (return the front item but don't removal it)
4. `size()`: return the number of items
5. `isEmpty`: same as `isEmpty` in the top

the **enqueue** and **dequeue** method is the main parts

2.2 enqueue method

insert the data in the queue

```
public void enqueue(int data){
    Node n = new Node(data);
    if(size == 0){
        front = n;
    }else{
        rear.next = n;
    }
    rear = n;
    size++;
    System.out.println(data + " is enqueue in the linkQueue");
}
```

2.3 dequeue method

take out off the rear of the queue

```
public int dequeue(){
    if (front == null){
        System.out.println("Queue reach the end");
        System.exit(1);
    }
    int data = front.data;
    // remove the front item
    front = front.next;
    size--;
    System.out.println(data + " is dequeue in the linkQueue");
    return data;
}
```

2.4 print linked list method

Use iteration to print all items in the queue

```
public void printLinkList(Node front) {
    if (front == null) {
        return;
    }
    printLinkList(front.next);
    System.out.print(front.data + " ");
}
```

Discussion & Challenges

1. For array and linked list

For use array or linked list these two methods to implement stack and queue, I think **using linked list** is **more better** but need think more. The advantage of using linked list is have **infinite capacity** to store the data, but also have some disadvantage, which is not very easy to get the nodes you want. You just can take the first and the last Node of the stack (or queue). The only way you can get the node you want is by using the `Node.next` to get the next Node object until reach the Node that you want.

Linked list by using the Node object to store the data, I was very impressed and found it amazing (also the double linked list is amazing too). It doesn't even need to create many variables to store Node objects, just need remember the first Node that can get all Nodes.

2. For Queue ADT and stack ADT

This two type of data structure is similar in many respects, both have two way to implement and the different is the following principle, which Queue ADT is FIFO and the stack ADT is FILO.

3. Some challenge for myself

maybe is to figure out which principle to follow and to figure out the connection of nodes.

project2 - B

complete the methods in the Double Linked List (DLL)

Question 1

requirement: complete the `insertFirst` and `insertLast` methods

1. Double Linked List Abstract Data Structure (DLL ADT)

Difference between DLL and single linked list is the Node in DLL is connect with the node before it and after it.

Node class in DLL:

```
public class Node implements Position {
    // element store the data
    private Object element;
    Node next;
    Node previous;
    public Node (int d){
        this.element = d;
    }
    public Object element() {
        return element;
    }
}
```

So in DLL can insert some data in the list.

2. Methods need be implemented

2.1 insertFirst method

this method is same as the pop in using linked list implement stack, insert the data in the beginning of the list.

The important thing is must to check if the first is null (which represent that the list is empty), if the first is null that the new node will be the first and the last of the list. **Also don't forget to increase the size number**.

(the Node class has implemented the Position interface)

```
public Position insertFirst(int data){
    Node node = new Node(data);
    node.next = first;
    if (first != null){
        first.previous = node;
    } else {
        last = node;
    }
    first = node;
    size++;
    System.out.println("The number " + data + " has been inserted at the beginning");
    return node;
}
```

2.2 insertLast method

This method is the opposite of the previous one, is insert the data in the end of the list.

and also need to pay attention to check if the last is null.

```
public Position insertLast(int data){
    Node node = new Node(data);
    node.previous = last;
    if (last != null){
        last.next = node;
    }
    last = node;
    size++;
    System.out.println("The number " + data + " has been inserted at the end");
    return node;
}
```

Question 2

requirement: complete the `insertAfter` and `insertBefore` methods.

1. insertAfter method

This method can insert one Node after the node that you input.

when you insert one Node into the DLL, remember to change the **Node before** the insert position and the **Node after** the insert position.

```
public Position insertAfter(Node position,int data){
    Node n = new Node(data);
    if (position.next == null){
        insertLast(data);
    }else{
        n.next = position.next;
        n.previous = position;
        position.next.previous = n;
        // or can use after(p) method
        position.next = n;
    }
    System.out.println("The number " + data + " has been inserted after " +
position.element);
    size++;
    return n;
}
```

2. insertBefore method

This method is similar to the insertAfter method

```
public Position insertBefore(Node position,int data){
    Node n = new Node(data);
    if (position.previous == null){
        insertFirst(data);
    }else{
        n.next = position;
        n.previous = position.previous;
        position.previous.next = n;
        position.previous = n;
    }
    System.out.println("The number " + data + " has been inserted before " +
position.element);
    size++;
    return n;
}
```

Question 3

requirement: complete the `remove` method

1. remove method

Have some special cases, for example the position is the first (or last) element of the list.

```
public int remove(Node position){
    int data = position.element();
    // assume that the data is int, is also can be an Object
    // if the position is the first and the last
    if (position.previous == null){
        position.next.previous = null;
    } else if (position.next == null){
        position.previous.next = null;
    } else {
        position.next.previous = position.previous;
        position.previous.next = position.next;
    }
    size--;
    System.out.println("The number " + data + " has been removed");
    return data;
}
```

Discussion & Challenges

1. DLL and SLL

DLL is more complex than SLL, in DLL the Position object connect to **next and previous** position, which make insert data into the list is possible. DLL is more useable than SLL, is more easy to take out the data in the list. Actually, if the size of the list is huge, same like to take out or insert data before & after is difficult in now code. I think can add some method in the Position interface, just like the index to make the node know how many nodes it is, so we can know use how many `Node.next` can get this Node (use some loop).

2. some challenges for myself

The challenge for myself is to make my code considering all special cases, also forget some special cases in somewhere.

project2 - C

Complete the methods in double-ended queue data strcture

Question 1

requirement: implement the `addFirst` and `addLast` methods

1. Double-ended queue Abstract Data Structure (DEQ ADT)

- A double-ended queue is a data structure that supports insertion and deletion at **both the front and the rear** of the queue
- more complicated than the queue
- It is like the **combination** of a **Stack** and a **Queue** (so is follow the FIFO and LIFO principle)

2. Methods that need be implemented

2.1 addFirst method

add the data in the beginning of queue

important things check if the first is null

```
public void addFirst(int data){
    Node n = new Node(data);
    if(isEmpty()){
        first = n;
        last = n;
    } else {
        n.next = first;
        first.previous = n;
        first = n;
    }
    System.out.println(data + " is been added in the beginning of the queue");
    size++;
}
```

2.2 addLast method

add the data in the end of queue

```
public void addLast(int data){
    Node n = new Node(data);
    if(isEmpty()){
        first = n;
    } else {
        n.previous = last;
        last.next = n;
    }
    last = n;
    System.out.println(data + " is been added in the end of the queue");
    size++;
}
```

Question 2

requirement: implement the `removeFirst` method

if the queue is empty, it will throws an EmptyDequeException.

have a special case for only have one data in the queue.

1. removeFirst method

```
public int removeFirst(){
    if(isEmpty()) {
        throw new EmptyDequeException();
    }
    int data = first.element;
    // remove first element
    first = first.next;
    if(size == 1){
        first = null;
        last = null;
    } else {
        first.previous = null;
    }
    System.out.println(data + " is been removed from the front of queue");
    size--;
    return data;
}
```

Question 3

requirement: implement the `removeLast` method

1. removeLast method

is same as the `removeFirst` method

```
public int removeLast(){
    if(isEmpty()){
        throw new EmptyDequeException();
    }
    int data = last.element;
    last = last.previous;
    if (size == 1){
        first = null;
        last = null;
    } else {
        last.next = null;
    }
    size--;
    System.out.println(data + " is been removed from the rear of queue");
    return data;
}
```

Discussion & Challenges

1. DEQ & DLL (implement by linked list)

I think this two ADT is similar in can access the data from beginning and end , is both can add data to the beginning (and end). But for myself, I think the DLL is more useful.

2. some challenges for myself

Actually, the challenges for me maybe is I always confusing different abstract data structures, especially when the two structures have many similarities,

and also I'm trying to iterate through some of the problems instead of using for loops (my code isn't efficient that way).