Lab 2: Stack & Queue

Deadline

2017/9/27 24:00

Please compress your code and upload into the FTP server with filename in format `162120100xx_name.zip`

Specification

Your task is to implement stack and queue class with double-ended queue. Before that, you need to implement the double-ended queue with doubly linked list. Each node of the doubly linked list holds the element, the previous and the next node.

Implement the following data structures.

1. **Doubly linked list node data structure**

   To model a single node in a doubly linked list, create a data type `DLNode` with the following API:

   ```
   public class DLNode {
       // create a DLNode
       DLNode(int element)
       // get the value of the element
       public int getElement()
       // set the value of the element
         public void setElement(int element)
       // get the previous DLNode
         public DLNode getPrev()
       // set the previous DLNode
         public void setPrev(DLNode prev)
       // get the next DLNode
         public DLNode getNext()
       // set the next DLNode
         public void setNext(DLNode next)
   }
   ```

2. **Double-ended queue data structure**

```
public class MyDeque {
    //create a deque
    MyDeque(DLNode node)
    //insert a node at the beginning of deque
    public void insertFirst(DLNode node)
    //remove and return the first node
      public DLNode removeFirst()
    //insert a node at the end of deque
      public void insertLast(DLNode node)
    //remove and return the last node
      public DLNode removeLast()
    //return first node
      public DLNode first()
    //return last node
      public DLNode last()
    //return number of nodes
      public int size()
    //judge whether the deque is empty
      public boolean isEmpty()
    //display content of the deque
      public String toString()
}
```

3. **Stack data structure**

```
public abstract class MyStack {
    //push a node into stack
      public abstract void push(DLNode node)
    //pop a node from stack
      public abstract DLNode pop()
    //return top node
      public abstract DLNode top()
     //return number of nodes
       public abstract int size()
    //judge whether the stack is empty
      public abstract boolean isEmpty()
    //display the content of the stack
       public String toString()
}
```

4. **Queue data structure**

```
public abstract class MyQueue {
    //enqueue a node
    public abstract void enqueue(DLNode node)
    //dequeue a node
      public abstract DLNode dequeue()
    //return front node of queue
      public abstract DLNode front()
    //return number of nodes
      public abstract int size()
    //judge whether the queue is empty
      public abstract boolean isEmpty()
    //display the content of the stack
       public String toString()
}
```

The data type `MyStack` and `MyQueue` are abstract, you need to implement them with `MyDeque` by using adaptor pattern. Your task is to design the `StackAdaptor` and `QueueAdaptor` which inherit `MyStack` and `MyQueue` respectively. You should implement your own data structure and may not call any library function other than those in `java.lang` in this lab.

Namely, create a class called `StackAdaptor` to extend the abstract class `MyStack` using your `MyDeque` , and so is `QueueAdaptor`.

Test case

The following method is an example to test the data type you implement. After each operation, there is an annotation showing the state of the stack or the queue.

```java
public class Main {
    public static void main(String[] args){
        DLNode temp = null;

        System.out.println("Stack");
        MyStack stack = new StackAdaptor(new MyDeque(new DLNode(1))); // 1
        System.out.println("1: " + stack.toString());
        stack.push(new DLNode(2));
        System.out.println("2: " + stack.toString());//12
        temp = stack.pop();
        System.out.println("pop " + temp.getElement());
        System.out.println("3: " + stack.toString());//1
        temp = stack.pop();
        System.out.println("pop " + temp.getElement());
        System.out.println("4: " + stack.toString());//
        stack.push(new DLNode(3));
        System.out.println("5: " + stack.toString());//3

        System.out.println("Queue");
        MyQueue queue = new QueueAdaptor(new MyDeque(new DLNode(1)));
        System.out.println("1: " + queue.toString());//1
        temp = queue.dequeue();
        System.out.println("dequeue " + temp.getElement());
        System.out.println("2: " + queue.toString());//
        queue.enqueue(new DLNode(2));
        System.out.println("3: " + queue.toString());//2
        queue.enqueue(new DLNode(3));
        System.out.println("4: " + queue.toString());//23
        temp = queue.dequeue();
        System.out.println("dequeue " + temp.getElement());
        System.out.println("5: " + queue.toString());//3
    }
}
```