栈和队列

一 目的

1、掌握栈和队列的顺序存储结构和链式存储结构及实现。

2、掌握栈和队列的入栈与出栈，入队与出队等基本运算的实现。

3、掌握栈和队列在实际问题中的应用。

二 内容

1、实现顺序栈或链栈的基本操作，并设计测试类使其实际运行测试各操作的正确性。

2、实现顺序存储基础上的循环队列的基本操作，并设计测试类使其实际运行测试各操作的正确性。

3*、假设以带头结点的循环链表表示队列，并且只设一个指针指向队尾元素结点（注意不设头指针），试编写相应的队列初始化、入队列和出队列的算法。

4*、假设将循环队列定义为：以域变量 rear 和 length 分别指示循环队列中队尾元素的位置和内含元素的个数。试给出此循环队列的队满条件，并写出相应的入队列和出队列的算法（在出队列的算法中要返回队头元素）。

5、前海理发店由于服务质量好而受到客户的特别好评，为此前来理发的客人越来越多，显然原来的纯人工管理模式不能适应客流量日益增大的情况了，请选择合适的数据结构为该店设计理发服务模型并实现理发服务管理功能。

三 设计说明

1．首先设计顺序栈和链栈的定义接口 IStack.java，链栈结点定义 StackNode.java，然后通过 SeqStack.java 实现顺序栈定义及基本操作、通:LinkStack.java 实现链栈定义及基本操作。最后设计测试类 TestStack 测试基本操作。

2.首先定义 Queue 接口，接着书写实现类 CircleSequenceQueue.java，最后通过 Test.Java 实现对顺序循环队列的测试。

四 功能说明

栈的接口 IStack.java 中定义了如下方法：E push(E item)入栈，E pop()出栈，E peek()取栈顶元素，int size()返回栈中元素的个数， boolean empty()判断栈是否为空 ，void clear()清空栈

接口 Queue.Java 中 append(Object obj)入队 delete()出队 getFront()获取头元素 isEmpty()判断是否为空。

五 调试分析

循环顺序队列满足先进先出的实现，同时注意到顺序循环队列跟普通顺序队列的差别在于预防了假溢出的状态

六 测试结果

```
*******入栈操作*******
23入栈
45入栈
3入栈
7入栈
6入栈
945入栈
*******出栈操作*******
945出栈
6出栈
7出栈
3出栈
45出栈
```

```
0 ,1 ,2 ,
3 ,4 ,5 ,6 ,7 ,8 ,9 ,10 ,11 ,12 ,12 ,12 ,
```

七 带注释的源代码

栈接口定义

```
package sj;
public interface IStack<E> {

    E push(E item); //入栈

    E pop(); //出栈

    E peek(); //取栈顶元素

    int size();//返回栈中元素的个数

    boolean empty();//判断栈是否为空

    void clear();  //清空栈

    }
```

顺序栈定义及基本操作实现: SeqStack.java

```
package sj;
import java.lang.reflect.Array;
import java.lang.reflect.Array;
```

```java
public class SeqStack<E> implements IStack<E> {

    private int maxsize; // 顺序栈的容量

    private E[]data; // 数组，用于存储顺序栈中的数据元素

    private int top; // 指示顺序栈的栈顶

    // 初始化栈

    @SuppressWarnings("unchecked")
    public SeqStack(Class<E> type,int size) {
       data = (E[]) Array.newInstance(type, size);
       this.maxsize = size;
       top = 0;
    }

    // 入栈操作

    public E push(E item) {
       if (isFull()) {
           data[top] = item;
           top++;
           return item;
       } else
           return null;
    }

    // 出栈操作

    public E pop() {
       if (!empty()) {
           E temp = data[top - 1];
           top--;
           return temp;
       } else {
           return null;
       }
    }

    // 获取栈顶数据元素

    public E peek() {
       if (!empty()) {
           E temp = data[top - 1];
           return temp;
       } else {
           return null;
```

```
        }
    }

    // 求栈的长度

    public int size() {
        return top;
    }

    // 判断顺序栈是否为空

    public boolean empty() {
        if (this.top == 0)
            return true;
        return false;
    }

    // 判断顺序栈是否为满

    public boolean isFull() {
        if (this.maxsize ==top)
            return false;
        return true;
    }

    // 清空栈

    public void clear() {
        this.top = 0;
    }
}
```

链栈结点的定义:StackNode.java

```
package sj;
public class StackNode<E> {

    private E data; // 数据域

    private StackNode<E>next; // 引用域

     //构造函数

    public StackNode(){
          }
    public StackNode(E data) {
        this.data = data;
    }
    public StackNode(E data, StackNode<E> next) {
        super();
        this.data = data;
```

```
        this.next = next;
    }

    //数据域 get 属性

    public E getData() {
        return data;
    }

    //数据域 set 属性

    public void setData(E data) {
        this.data = data;
    }

    //引用域 get 属性

    public StackNode<E> getNext() {
        return next;
    }

    //引用域 get 属性

    public void setNext(StackNode<E> next) {
        this.next = next;
    }
}
```

链栈定义及基本操作实现:LinkStack.java

```
package sj;
public class LinkStack<E> implements IStack<E> {

    private StackNode<E>top; // 栈顶指示器

    private int size; // 栈中结点的个数


    public StackNode<E> getTop() {
        return top;
    }

    public void setTop(StackNode<E> top) {
        this.top = top;
    }

    public int getSize() {
        return size;
    }

    public void setSize(int size) {
```

```
    this.size = size;
}

// 初始化链栈
public LinkStack() {
    top =null;
    size = 0;
}

// 入栈操作
public E push(E item) {
    StackNode<E> i = new StackNode<E>(item);
    i.setNext(top);
    top = i;
    size++;
    return item;
}

// 出栈操作
public E pop() {
    E i = null;
    if (!empty()) {
        i = top.getData();
        top =top.getNext();
        size--;
    }
    return i;
}

// 获取栈顶数据元素
public E peek() {
    if (!empty())
        return top.getData();
    return null;
}

// 求栈的长度
public int size() {
    return size;
```

```
    }

    // 判断顺序栈是否为空
    public boolean empty() {
       if (top ==null)
           return true;
       return false;
    }
    // 清空栈
    public void clear() {
       top =null;
    }
}
```

栈的测试

```
package sj;
public class TestStack {
    public static void main(String[] args) {
       int[] data={23,45,3,7,6,945};
       //IStack<Integer> stack=new
SeqStack<Integer>(Integer.class,data.length);
       IStack<Integer> stack=new LinkStack<Integer>();
       //入栈操作

       System.out.println("*******入栈操作*******");
        for(int i=0; i<data.length;i++){
            stack.push(data[i]);
             System.out.println(data[i]+"入栈");
            }
        int size=stack.size();
       //出栈操作

        System.out.println("*******出栈操作*******");
        for(int i=1; i<size;i++){
            System.out.println(stack.pop()+"出栈 ");
        }
    }
}
```

定义 Queue 接口

```
package sj;
public interface Queue {

    //入队

    public void append(Object obj) throws Exception;

    //出队

    public Object delete() throws Exception;

    //获取头元素

    public Object getFront() throws Exception;

    //队列是否为空

    public boolean isEmpty();

}
```

实现循环顺序队列

```
package sj;
public class CircleSequenceQueue implements Queue {

    private Object[] queueData; //队列数组

    private int maxSize; //队列最大长度

    private int size; //队列大小

    private int rearIndex; //队尾位置

    private int frontIndex; //对首位置


    public CircleSequenceQueue() {

        this.maxSize = 10; //设置默认大小

        this.size = 0;
        this.rearIndex = 0;
        this.frontIndex = -1;
        this.queueData = new Object[maxSize];
    }

    public CircleSequenceQueue(int maxSize) {
        this.maxSize = maxSize;
        this.size = 0;
        this.rearIndex = 0;
        this.frontIndex = -1;
```

```
        this.queueData = new Object[maxSize];
    }

    public void append(Object obj) throws Exception {
        if (size == maxSize)
            throw new Exception("队列已满!!!");

        this.queueData[rearIndex] = obj;
        this.rearIndex = rearIndex+1==this.maxSize?0:++rearIndex; //如果队列
数组游标到了数组尾部，转至 0 位置
        this.size++;

    }
    public Object delete() throws Exception {
        if (0 == size)
            throw new Exception("队列为空!!!");

        this.frontIndex = frontIndex+1==this.maxSize?0:++frontIndex; //如果
队列数组游标到了数组尾部，转至 0 位置
        this.size--;
        return this.queueData[frontIndex];
    }
    public Object getFront() throws Exception {
        if (0 == size)
            throw new Exception("队列为空!!!");
        int index = frontIndex+1==this.maxSize?0:frontIndex+1;
        return this.queueData[index];
    }
    public boolean isEmpty() {
        return size>0?false:true;
    }
}
```