

华中科技大学

课程实验报告

课程名称: Java 语言程序设计

实验名称: 泛型栈模拟泛型队列

院 系: 计算机科学与技术

专业班级: IOT1601

学 号: U201614897

姓 名: 潘 越

指导教师: 马光志

2019 年 4 月 21 日

目 录

一、需求分析.....	1
1. 题目要求.....	1
2. 需求分析.....	1
二、系统设计.....	2
1. 概要设计.....	2
2. 详细设计.....	2
(1) Insert.....	2
(2) Remove.....	3
(3) Examine.....	3
三、软件开发.....	3
四、软件测试.....	4
五、特点与不足.....	5
1. 技术特点.....	5
2. 不足和改进的建议.....	5
六、过程和体会.....	5
1. 遇到的问题和主要解决方法.....	5
2. 课程设计的体会.....	6
七、源码和说明.....	6
1. 文件清单及其功能说明.....	6
2. 用户使用说明书.....	6
3. 源代码.....	6
参考文献.....	11

一、需求分析

1. 题目要求

参见 <https://docs.oracle.com/javase/7/docs/api/java/util/Queue.html>，Java 提供的 `java.util.Queue` 是一个接口没有构造函数。试用 `java.util.Stack<E>` 泛型栈作为父类，用另一个泛型栈对象作为成员变量，模拟实现一个泛型子类 `Queue<E>`。除提供无参构造函数 `Queue()` 外，其它所有队列函数严格参照 `java.util.Queue` 的接口定义实现。

```
import java.util.Stack;
import java.util.NoSuchElementException;
public class Queue<E> extends Stack<E>{
    private Stack<E> stk;
    public Queue() { /* 在此插入代码*/ }
    public boolean add(E e) throws IllegalStateException, ClassCastException,
    NullPointerException, IllegalArgumentException { /* 在此插入代码*/ }
    public boolean offer(E e) throws ClassCastException, NullPointerException,
    IllegalArgumentException { /* 在此插入代码*/ }
    public E remove() throws NoSuchElementException { /* 在此插入代码*/ }
    public E poll() { /* 在此插入代码*/ }
    public E peek() { /* 在此插入代码*/ }
    public E element() throws NoSuchElementException { /* 在此插入代码*/ }
}
```

考虑到 `Stack<E>` 只能存储类型 `E` 的元素，以及 `Stack` 是一个存储能力(capacity, 参见有关说明)理论上无限的类型，这可能会影响到相关方法的异常处理，请适当处理上述异常（也许某些异常从来都不会发生）。

2. 需求分析

本次实验和上学期做过的 C++ 程序设计实验基本上一致，大部分的设计思想可以复用上一次做过的实验，不过在 Java 的处理上也和 C++ 有一些区别。

首先 Java 的 `Stack<E>` 是理论上存储能力无限的类型，利用它来构建的队列也是一个理论上存储能力无限的类型，因此一般情况下不会发生存储容量满的情况，构造函数中也无需添加容量参数。另一方面是在做 C++ 的这个实验时没有考虑异常处理的情况，而本实验中则根据文档，三组操作中都有一个是要抛出异常的，这也是我们要增加考虑的地方。

二、系统设计

1.概要设计

除了构造函数外，系统需要实现六个接口，他们分为三组，每一组都有两个函数，其中一个在发生错误时抛出一个异常，另一个在发生错误时返回一个确定的值，这三组操作分别提供入队列、出队列以及获取队列第一个元素的操作。具体如下表所示：

表 1.1 Queue<E>接口

	Throws exception	Returns special value
Insert	add(e)	offer(e)
Remove	remove()	poll()
Examine	element()	peek()

该队列继承一个栈，同时自身又有一个成员栈。这里将继承栈作为出队栈，将成员栈作为入队栈，整个系统的 UML 类图如下所示：

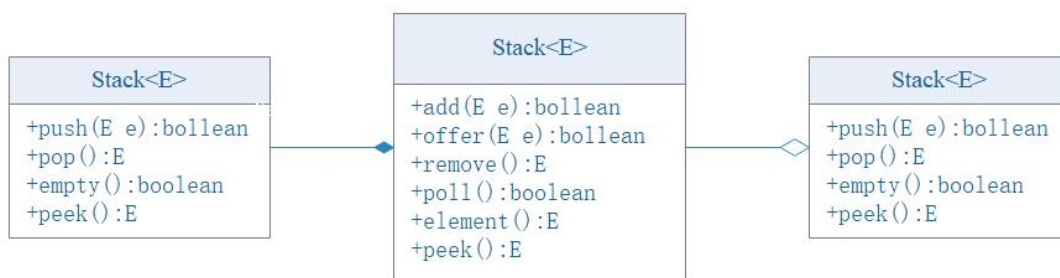


图 1.1 系统 UML 类图

2.详细设计

下面针对题目中需要完成的六个接口，逐一设计其算法流程。

(1) Insert

① add(e)方法

输入类型：E，返回类型：boolean，可抛出异常：NullPointerException.

add 方法用于向队列尾部增加一个元素。首先要判断该元素是否为空，若为空则抛出 NullPointerException 异常，若不为空，则直接插入入队栈中即可。这里不考虑调整元素操作，所有的栈内元素调整操作都在出队操作中执行。

② offer(e)方法

输入类型：E，返回类型：boolean.

offer 方法同样用于向队列尾部增加一个元素。和 add 的区别是，offer 方法在操作失败时应该返回一个值（例如 null），而不抛出异常。但是由于本实验的

设计，Stack 和 Queue 都是理论上容量无限的泛型，因此操作失败的异常永远不会发生，这里 offer 和 add 方法的实现是一致的。

(2) Remove

① remove()方法

输入类型：void，返回类型：E，可抛出异常：NoSuchElementException.

remove 方法用于从队列头部移除一个元素，并返回该元素的值。由于所有的元素都从出队栈中出队，因此这里需要考虑两种情况。若出队栈中有元素，则直接从出队栈中出栈一个元素，若出队栈为空，则将入队栈中的元素按顺序弹出一个，并进入出队栈，直到入队栈为空。经过这样一个调整，原本在栈中后进先出的顺序就变成了先进先出，满足栈的需求。而异常处理则是当两个栈都为空时，即队列内无元素，则抛出 NoSuchElementException 异常。

② poll()方法

输入类型：void，返回类型：E.

poll 方法和 remove 的算法实现一致，区别是 poll 不抛出异常，当出现队列中无元素的时候，返回 null 即可。

(3) Examine

① element()方法

输入类型：void，返回类型：E，可抛出异常：NoSuchElementException.

element 方法返回当前队列的第一个元素。和 remove 方法一样，同样有两种情况，若出队栈中有元素，则返回栈的第一个元素。若出队栈为空，则将入队栈中的元素按顺序弹出一个，并进入出队栈，直到入队栈为空，再返回出队栈的第一个元素即可。当两个栈均为空时，抛出 NoSuchElementException 异常。

② peek()方法

输入类型：void，返回类型：E.

peek 方法对应 element 方法，不抛出异常，其他功能一致，当出现队列中无元素的时候，返回 null.

三、软件开发

本实验的开发与测试的环境如下：

- (1) 操作系统：Arch Linux x86_64 kernel version 5.0.7
- (2) JRE: jre8-openjdk 8.u212-1
- (3) JDK: jdk8-openjdk 8.u212-1
- (4) IDE: IntelliJ IDEA 2019.1

实验程序的编译与调试均在 IDEA 中完成。

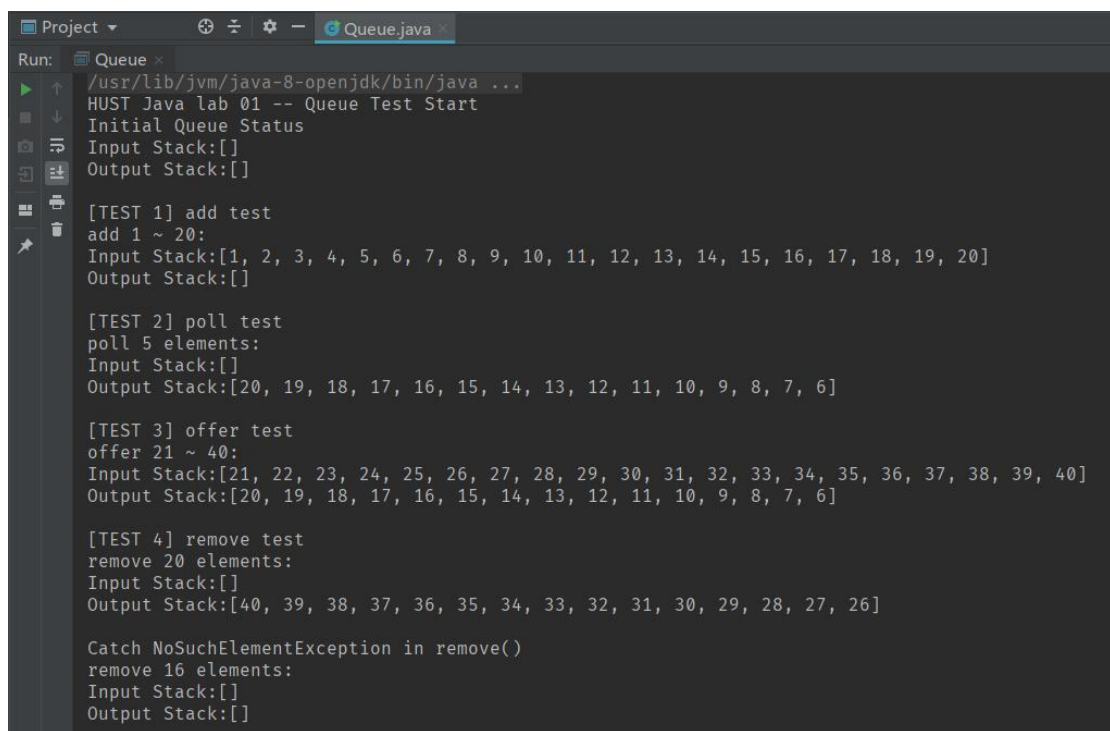
四、软件测试

根据题目中实现的功能，我们构造六个测试案例，分别测试六个接口，前面的接口在通过测试后，可以用于构建后面测试的测试程序。经过分析，设计如下表的测试用例。

表 4.1 测试用例

项目	测试目的	测试用例	测试结果
1	测试 add(e)	将 1~20 按顺序入队 打印队列状况	通过
2	测试 poll()	出队 5 个元素 打印队列状况	通过
3	测试 offer(e)	将 21~40 按顺序入队 打印队列状况	通过
4	测试 remove()	出队 20 个元素 打印队列状况 出队 16 个元素 捕获最后一次出队引发的异常	通过
5	测试 element()	将 1~20 按顺序入队 打印队列状况 按顺序出队 20 次，并打印队首元素 21 次 捕获最后一次获取队首元素引发的异常	通过
6	测试 peek()	将 1~20 按顺序入队 打印队列状况 按顺序出队 20 次，并打印队首元素 21 次 若返回 null，则打印错误信息	通过

测试程序直接作为 Queue<E>类的 main 函数即可，执行的结果如下面两图。



```

Project Queue.java
Run: Queue
/usr/lib/jvm/java-8-openjdk/bin/java ...
HUST Java lab 01 -- Queue Test Start
Initial Queue Status
Input Stack:[]
Output Stack:[]

[TEST 1] add test
add 1 ~ 20:
Input Stack:[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
Output Stack:[]

[TEST 2] poll test
poll 5 elements:
Input Stack:[]
Output Stack:[20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6]

[TEST 3] offer test
offer 21 ~ 40:
Input Stack:[21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40]
Output Stack:[20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6]

[TEST 4] remove test
remove 20 elements:
Input Stack:[]
Output Stack:[40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 27, 26]

Catch NoSuchElementException in remove()
remove 16 elements:
Input Stack:[]
Output Stack:[]
    
```

图 4.1 测试结果（1）

```
[TEST 5] element test
add 1 ~ 20:
Input Stack:[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
Output Stack:[]

remove 20 times and element 21 times
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Catch NoSuchElementException in element()
remove 20 elements:
Input Stack:[]
Output Stack:[]

[TEST 6] peek test
add 1 ~ 20:
Input Stack:[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
Output Stack:[]

remove 20 times and peek 21 times
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
peek returns null
remove 20 elements:
Input Stack:[]
Output Stack:[]

All Test passed!
```

图 4.1 测试结果（3）

从上面两图即可看到，程序正确的通过了所有的测试，在每次操作后，打印的状态也符合队列的特性，验证了程序的正确性。

五、特点与不足

1.技术特点

本实验中实现的 `Queue<E>` 泛型简洁明了，易于使用，对常见的错误都进行了处理。同时六项测试也验证了功能的完善性。

2.不足和改进的建议

由于这里的栈是直到 `Remove` 类方法或者 `Examine` 方法调用之前，都不执行两个栈之间的元素调整，所以在第一次执行这类方法时可以会很慢。

对这一点可以简单做一个改进，设计一个入队栈的阈值，当元素超过这个阈值并且出队栈为空时，就调整一部分元素过去，这样可以在队列使用的过程中平均运行的时间。

六、过程和体会

1.遇到的问题 and 主要解决方法

实验中遇到的主要问题就是在编写 `peek` 方法时，忘了在 `peek` 方法内调用 `peek` 方法时加上 `super` 调用超类的 `super` 方法，就导致了循环调用的问题，我编写的 `peek` 方法覆盖了原本 `Stack<E>` 的 `peek` 方法。这里使用 `super` 就可以保证调用超类中的 `peek` 方法。

2.课程设计的体会

通过本次实验，我熟悉了 Java 编程的基本环境，对 Java 的继承、覆盖和重写机制有了更深的理解。并通过和上学期的 C++实验作比较，了解了 Java 程序细节和 C++的不同之处，为之后的实验打好了基础。

七、源码和说明

1.文件清单及其功能说明

本实验的代码文件为 Queue.java，包含 Queue<E>类的实现细节以及 main 测试函数。

2.用户使用说明书

使用 IntelliJ IDEA 打开实验文件夹，直接点击编译运行即可看到实验的结果。

3.源代码

```
import java.util.Stack;
import java.util.NoSuchElementException;

public class Queue<E> extends Stack<E> {
    private Stack<E> stk;

    public Queue() {
        stk = new Stack<>();
    }

    public boolean add(E e) throws NullPointerException {
        if (e == null) {
            throw new NullPointerException();
        }
        stk.push(e);
        return true;
    }

    public boolean offer(E e) throws NullPointerException {
        if (e == null) {
            throw new NullPointerException();
        }
        stk.push(e);
    }
}
```



```

        return true;
    }

    public E remove() throws NoSuchElementException {
        if (!this.empty()) {
            return this.pop();
        } else if (!stk.empty()) {
            while (!stk.empty()) {
                this.push(stk.pop());
            }
            return this.pop();
        } else {
            throw new NoSuchElementException();
        }
    }

    public E poll() {
        if (!this.empty()) {
            return this.pop();
        } else if (!stk.empty()) {
            while (!stk.empty()) {
                this.push(stk.pop());
            }
            return this.pop();
        } else {
            return null;
        }
    }

    public E element() throws NoSuchElementException {
        if (!this.empty()) {
            return super.peek();
        } else if (!stk.empty()) {
            while (!stk.empty()) {
                this.push(stk.pop());
            }
            return super.peek();
        } else {
            throw new NoSuchElementException();
        }
    }
}

```

```

public E peek() {
    if (!this.empty()) {
        return super.peek();
    } else if (!stk.empty()) {
        while (!stk.empty()) {
            this.push(stk.pop());
        }
        return super.peek();
    } else {
        return null;
    }
}

public void show(String str) {
    System.out.println(str);
    System.out.printf("Input Stack:");
    System.out.println(stk.toString());
    System.out.printf("Output Stack:");
    System.out.println(this.toString());
    System.out.println();
}

public static void main(String[] args) {
    Queue<Integer> MyQueue = new Queue<>();

    // Start test
    System.out.println("HUST Java lab 01 -- Queue Test Start");
    MyQueue.show("Initial Queue Status");

    // Test add
    System.out.println("[TEST 1] add test");
    for (int i = 1; i <= 20; i++)
        MyQueue.add(i);
    MyQueue.show("add 1 ~ 20:");

    // Test poll
    System.out.println("[TEST 2] poll test");
    for (int i = 1; i <= 5; i++)
        MyQueue.poll();
    MyQueue.show("poll 5 elements:");

    // Test offer

```

```

        System.out.println("[TEST 3] offer test");
        for (int i = 21; i <= 40; i++)
            MyQueue.offer(i);
        MyQueue.show("offer 21 ~ 40:");

        // Test remove
        System.out.println("[TEST 4] remove test");
        for (int i = 1; i <= 20; i++) {
            try {
                MyQueue.remove();
            } catch (NoSuchElementException e) {
                System.out.println("Catch
NoSuchElementException");
            }
        }
        MyQueue.show("remove 20 elements:");
        for (int i = 1; i <= 16; i++) {
            try {
                MyQueue.remove();
            } catch (NoSuchElementException e) {
                System.out.println("Catch
NoSuchElementException in remove()");
            }
        }
        MyQueue.show("remove 16 elements:");

        // Test element
        System.out.println("[TEST 5] element test");
        for (int i = 1; i <= 20; i++)
            MyQueue.add(i);
        MyQueue.show("add 1 ~ 20:");
        System.out.println("remove 20 times and element 21 times");
        for (int i = 1; i <= 20; i++) {
            System.out.printf("%d ", MyQueue.element());
            MyQueue.remove();
        }
        System.out.println();
        try {
            System.out.printf("%d ", MyQueue.element());
        } catch (NoSuchElementException e) {
            System.out.println("Catch NoSuchElementException
in element()");
        }
    }
}

```

```

    }
    MyQueue.show("remove 20 elements:");

    // Test peek
    System.out.println("[TEST 6] peek test");
    for (int i = 1; i <= 20; i++)
        MyQueue.add(i);
    MyQueue.show("add 1 ~ 20:");
    System.out.println("remove 20 times and peek 21 times");
    for (int i = 1; i <= 20; i++) {
        System.out.printf("%d ", MyQueue.element());
        MyQueue.remove();
    }
    System.out.println();
    if (MyQueue.peek() == null) {
        System.out.println("peek returns null");
    }
    MyQueue.show("remove 20 elements:");

    System.out.println("All Test passed!");
}
}

```

参考文献

- 1 [美] Cay S. Horstmann 著, 周立新, 叶乃文, 邝劲筠, 杜永萍译. Java 核心技术 卷 I 基础知识 (原书第 10 版). 北京: 机械工业出版社, 2018.