# 華中科技大學

# 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 ：　校交 1601 班

学　　号 ：　U201610504

姓　　名 ：　刘逸帆

指导教师 ：　吕新桥


2019 年 5 月 10 日

# 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>，当存储元素的第 1 个栈的元素超过 dump 时，再有元素入队列就倒入第 2 栈。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>{
    public final int dump=10;
    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>只能存储类型为 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）。

思考：Queue<E>是否应该提供 clone 和 equals 函数，以及其它一些函数如 addAll 等？

## 2. 需求分析

在设计队列的过程中，严格参照 java.util.Queue 的接口实现来设计队列支持的方法以及每种方法中异常发生时的条件：

对于 add()方法，其需要实现的功能是：将指定的元素插入队列中。方法的返回值类型为 boolean，成功插入元素时返回 true；如果队列的空间不足，方法抛出 IllegalStateException 异常；如果插入的元素类型不符，方法抛出 ClassCastException 异常；如果插入的元素为 null，

方法抛出 NullPointerException 异常；如果插入的元素属性与预期不符，方法抛出 IllegalArgumentException 异常。

对于 offer()方法，其需要实现的功能是：将指定的元素插入队列中。方法的返回值类型为 boolean，成功插入元素时返回 true，插入失败则返回 false；如果插入的元素类型不符，方法抛出 ClassCastException 异常；如果插入的元素为 null，方法抛出 NullPointerException 异常；如果插入的元素属性与预期不符，方法抛出 IllegalArgumentException 异常。

对于 remove()方法，其需要实现的功能是：将队列头部的元素出队列。方法的返回值类型为类型 E，出队列成功时返回队列头部的元素；如果队列已经为空，方法抛出 NoSuchElementException 异常。

对于 poll()方法，其需要实现的功能是：将队列头部的元素出队列。方法的返回值类型为类型 E，出队列成功时返回队列头部的元素，若队列已经为空，返回 null。

对于 element()方法，其需要实现的功能是：获得队列头部的元素。方法的返回值类型为类型 E，若成功取得队列头部元素则将其返回；若队列已经为空，方法抛出 NoSuchElementException 异常。

对于 peek()方法，其需要实现的功能是：获得队列头部的元素。方法的返回值类型为类型 E，若成功取得队列头部元素则将其返回，若队列已经为空，返回 null。

此外在设计过程中由于是使用栈实现队列，还需要考虑使用栈实现队列与直接实现队列的不同，包括使用栈的实现带来的正面影响、负面影响、限制以及扩展性。

# 二、系统设计

## 1. 概要设计

除了构造函数外，基于堆栈构建的队列还需要实现六个方法，六个方法两两为一组完成了队列的 3 种服务：入队列、出队列、取队列头的值。由于所有的接口在同一个类中，所以没有层次结构。考虑到此队列类继承一个栈类并持有一个栈类，其 UML 图如 2.1 所示。在本实验中，元素入队列时将被加入所持有的栈 stk，而出队列时从继承而来的栈中取出元素。

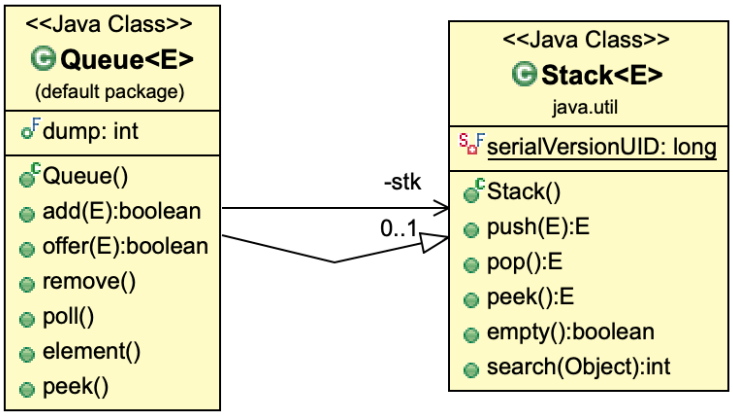


图 2.1 Queue 类 UML 图

## 2. 详细设计

根据 1.2 小节中的需求分析，对队列所支持的方法依次设计如下：

1) add()方法

功能：add 方法用于向队列尾部添加一个元素。首先判断要插入的元素是否为空，如果为空则拒绝插入并抛出 NullPointerException 异常；如果不为空则进行判断：若持有的 stk 栈中的元素个数已经达到了 dump 所指定的最大个数，则尝试将持有的栈中的内容逐一弹出并倒入继承而来的栈中，若此时继承而来的栈不为空则不能进行这一“倾倒”操作，抛出 IllegalStateException；若持有的 stk 栈中的元素个数未达到 dump 所指定的最大个数，则直接将 e 加入持有的栈中即可。

入口参数：E e，待插入队列的元素 e。

出口参数：boolean 类型，插入成功则返回 true。

流程图：如图 2.2 所示。

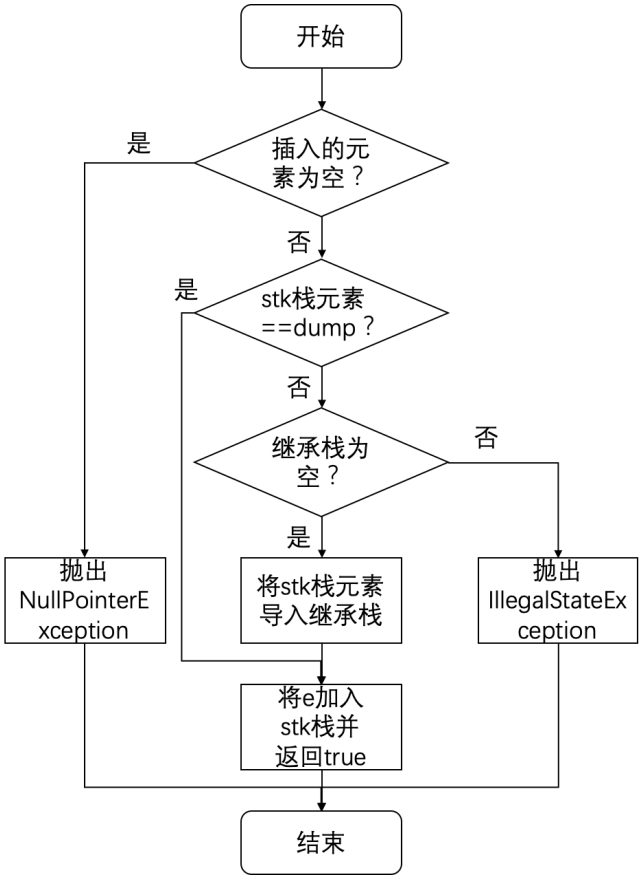


图 2.2 add 方法流程图

2) offer()方法

功能：offer 方法与 add 方法类似，但是在插入失败时其不会抛出 IllegalStateException，取而代之的，此时返回 false。

入口参数：E e，待插入队列的元素 e。

出口参数：boolean 类型，插入成功则返回 true，否则插入失败则返回 false。

流程图：如图 2.3 所示。

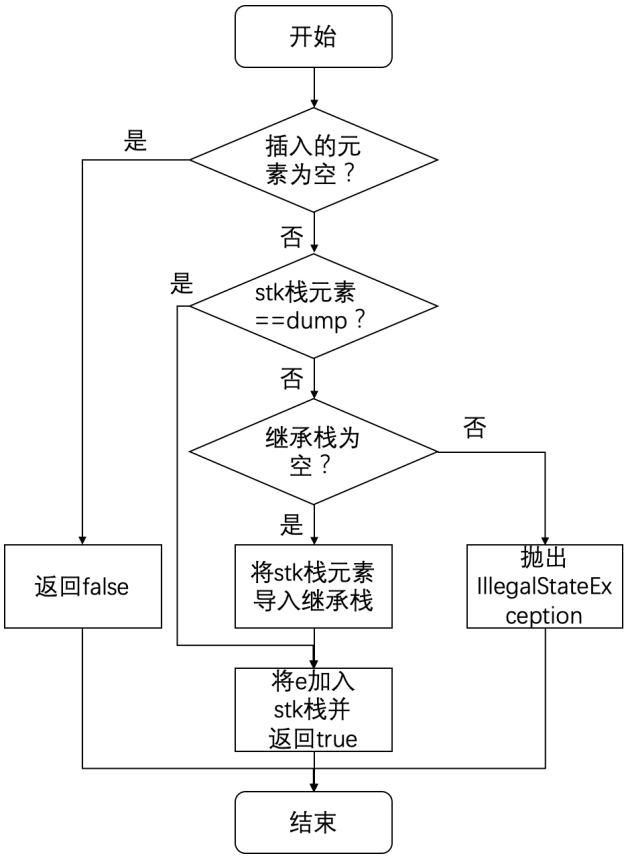


图 2.3 offer 方法流程图

3) remove 方法

功能：remove 方法用于尝试从队列首部移除一个值并返回这个值。首先，判断继承而来的栈是否为空，如果不为空则直接弹出并返回其栈顶的元素即可。如果为空则进一步判断持有的 stk 栈是否为空；如果不为空则首先将其中的元素逐一弹出并压入继承而来的栈中，然后从继承而来的栈中弹出顶端元素并返回即可；若非上述两种情况，说明两个栈中均没有元素，即队列为空，此时抛出 NoSuchElementException 异常。

入口参数：无。

出口参数：队列头元素 E e。

流程图：如图 2.4 所示。

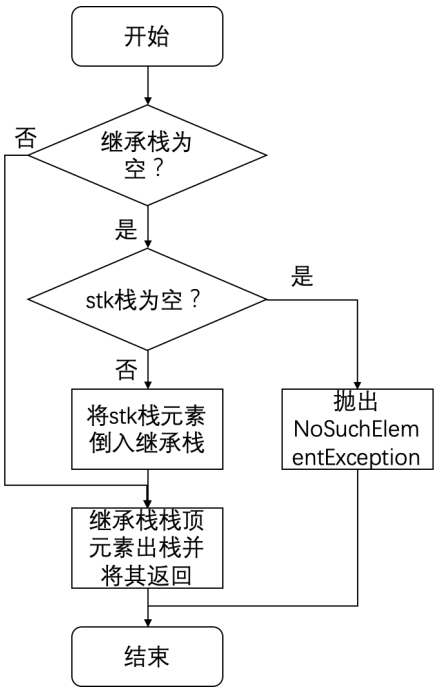


图 2.4 remove 方法流程图

4) poll 方法

功能：poll 方法与 remove 方法类似，但在队列为空（即持有的栈和继承的栈均为空）时返回 null 而不是抛出异常。

入口参数：无。

出口参数：队列头元素 E e。

流程图：如图 2.5 所示

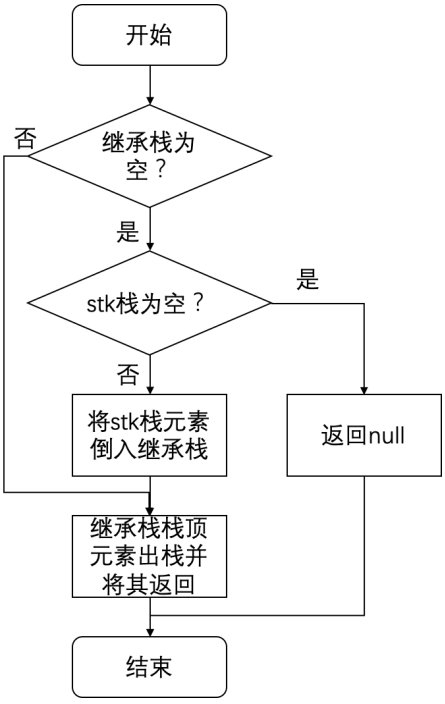


图 2.5 poll 方法流程图

5) element 方法

功能：element 方法与 remove 方法相似，不同的是它不使队列首部的元素出队列。

入口参数：无。

出口参数：队列头元素 E e。

6) peek 方法

功能：peek 方法与 poll 方法相似，不同的是它不使队列首部的元素出队列。

入口参数：无。

出口参数：队列头元素 E e。

# 三、软件开发

开发环境：具体开发环境如表 3.1 所示。

表 3.1 软件开发环境

| 项目 | 版本 |
|---|---|
| 操作系统 | MAC OS X |
| JRE | OpenJDK Runtime Environment 1.8.0_172-b11 |
| JDK | OpenJDK 1.8.0_172, Java 8 |

编译、连接生成可执行文件：将项目导入 Eclipse 并编译运行即可。

调试工具：Eclipse。

# 四、软件测试

1) add()方法测试

先入队列至队列满，再通过 try 语句尝试向队列中插入元素、插入 null 值，程序输出如图 3.1 所示，输出结果说明队列能正确处理元素上限和错误值插入的异常。

```
insert dump done
insert overflow
insert error
--- insert test done. ---
```

图 3.1 add 方法测试结果

2) remove()方法测试

将队列中的元素依次出队列并输出其值，在队列为空后再通过 try 语句尝试出队列，程序输出如图 3.2 所示，输出结果说明队列能正确处理 remove 空队列的异常。

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
remove dump done
remove error
--- remove test done. ---
```

图 3.2 remove 方法测试结果

3) poll()方法测试

分别对空队列和内含元素的队列进行 poll 操作，输出结果如图 3.3 所示，说明程序在 poll 空队列时能够正常返回 null 值。

```
null
100
--- poll test done. ---
```

图 3.3 poll 方法测试结果

4) offer()方法测试

将队列排空后，尝试通过 offer 对空队列进行操作，程序输出如图 3.4 所示，说明程序在 offer 空队列时能够正常返回 null 而不抛出异常。

```
offer
--- offer test done. ---
```

图 3.4 offer 方法测试结果

5) peek()和 element()方法测试

分别在队列中有无元素时对队列进行 peek 操作和 element 操作，程序输出如图 3.5 所示，说明程序在对有元素的队列进行 peek 和 element 操作时，都能取出队首元素的值但不将其排出；对于空队列，element 会抛出异常而 peek 操作能正常返回 null 值。

```
0
0
0
0
element error
--- peek and element test done. ---
```

图 3.5 peek 和 element 方法测试结果

6) 栈满测试

尝试在 stk 栈为满（10 个元素）、继承栈中有两个元素的情况下尝试向队列中 add 元素，程序在最后一次 add 的过程中抛出了 IllegalStateException 异常；再尝试通过 remove 操作将队列中的 12 个元素排空，随后尝试对空队列进行 remove 操作，引发 NoSuchElementException 异常。程序输出如图 3.6 所示，可以判断通过堆栈实现的队列具有正确的特性，验证了代码的正确性。

```
insert overflow
remove overflow
```

图 3.6 测试堆栈实现队列的正确性

综上所述，对于 Queue 类的每个方法的测试均通过，程序依照预期抛出了异常，说明所有的功能正常，并与预期的结果一致。

# 五、特点与不足

## 1. 技术特点

本 Queue 类实现简洁易懂，并且效率较高，对于可能出现的错误都进行了处理，并且符合 Queue 类接口定义的规范。测试程序的代码覆盖范围广，几乎覆盖了所有的代码。

## 2. 不足和改进的建议

部分代码还可以进行进一步的优化，如 poll 方法在倾倒时可以留下一个元素以减少一次 pop 和一次 push 操作；每组中两个相似的方法可以将其相同的部分提取成为私有方法以提高代码的复用程度；测试用例仍有更完善的余地，从而能更确切得体现出程序的正确性。

# 六、过程和体会

## 1. 遇到的主要问题和解决方法

在最初测试程序的设计中缺少对于部分代码的覆盖，导致一些问题没有检出，在使用了同学的测试样例后发现程序出错，进而发现此问题。在对于测试程序以及源程序进行修改后能够完整的覆盖所有的代码，从而保证了没有意料之外的错误发生。

## 2. 课程设计的体会

通过此次课程设计我对于 Java 的继承机制、覆盖与重写机制有了更进一步的了解。由于

本实验的内容与 C++实验的内容相似，我得以更为深入的认识到了 Java 的各个实现细节相较于 C++的不同之处，并能够更为熟练的使用 Java 进行开发。更为重要的是，我体会到了严谨的编码以及测试对于程序健壮性的重要之处，并会将这种态度应用于今后的学习生活过程中。

# 七、源码和说明

## 1. 文件清单及其功能说明

本实验的最终实现由 2 个文件构成：Queue.java 以及 Main.java。Queue.java 为 Queue 类的实现（使用 Stack 类），而 Main.java 则是对于 Queue.java 的测试程序。

## 2. 用户使用说明书

使用 Eclipse 导入项目并编译执行即可。

## 3. 源代码

1) Queue.java 程序代码

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

public class Queue<E> extends Stack<E>{
    public final int dump = 10;
    private   Stack<E> stk;

    // first stk: stk, second: super
    public Queue( ){
        stk = new Stack<E>();
    }

    public boolean add(E e) throws IllegalStateException, ClassCastException, NullPointerException,
IllegalArgumentException{
        if (e==null) {// if the specified element is null and this queue does not permit null elements
            throw new NullPointerException();
        }
        if (stk.size() == dump) {
            if (this.isEmpty()) {
                while (!stk.isEmpty()) {
                    this.push(stk.pop());
                }
                stk.push(e);
            }
            else {
                throw new IllegalStateException();
            }
        }
        else {
            stk.push(e);
        }
        return true;
    }

    public boolean offer(E e) throws ClassCastException, NullPointerException,
IllegalArgumentException{
        if (e==null) {// if the specified element is null and this queue does not permit null elements
```

```
            throw new NullPointerException();
        }
        if (stk.size() == dump) {
            if (this.isEmpty()) {
                while (!stk.isEmpty()) {
                    this.push(stk.pop());
                }
                stk.push(e);
            }
            else {
                return false;
            }
        }
        else {
            stk.push(e);
        }
        return true;
    }

    public E remove( ) throws NoSuchElementException {
        if(this.isEmpty()) {
            if(stk.isEmpty()) {// if this queue is empty
                throw new NoSuchElementException();
            }
            else {// move items from 'stk' to 'this'
                while(!stk.isEmpty()) {
                    this.push(stk.pop());
                }
                return this.pop();
            }
        }
        else {
            return this.pop();
        }
    }

    public E poll( ) {
        if(this.isEmpty()) {
            if(stk.isEmpty()) {// if this queue is empty
                return null;
            }
            else {// move items in 'stk' to 'this'
                while(!stk.isEmpty()) {
                    this.push(stk.pop());
                }
                return this.pop();
            }
        }
        else {
            return this.pop();
        }
    }

    public E element( ) throws NoSuchElementException {   /* 在此插入代码*/
        if(this.isEmpty()) {
            if(stk.isEmpty()) {// if this queue is empty
                throw new NoSuchElementException();
            }
            else {
```

```
                    while(!stk.isEmpty()) {
                        this.push(stk.pop());
                    }
                    return super.lastElement();
                }
            }
            else {// move items in 'stk' to 'this'
                return super.lastElement();
            }
        }

        public E peek ( ) {   /* 在此插入代码*/
            if(this.isEmpty()) {
                if(stk.isEmpty()) {// if this queue is empty
                    return null;
                }
                else {// move items in 'stk' to 'this'
                    while(!stk.isEmpty()) {
                        this.push(stk.pop());
                    }
                    return super.lastElement();
                }
            }
            else {
                return super.lastElement();
            }
        }
    }
```

2) Main.java 程序代码

```
    import java.util.NoSuchElementException;
    public class Main{
        public static void main(String[] argvs){
            Queue<Integer> que = new Queue<Integer>();
            int i;
            int dump = 20;

            // add test
            for(i = 0; i < dump; ++i)
                que.add(i);
            System.out.println("insert dump done");
            // add more data
            try{
                que.add(123);
            }
            catch (IllegalStateException e) {
                System.out.println("insert overflow");
            }
            // add exception test
            try{
                que.add(null);
            }
            catch (NullPointerException e) {
                System.out.println("insert error");
            }
            System.out.println("--- insert test done. ---");


            // remove test
```

```
for(i = 0; i < dump; ++i)
	System.out.println(que.remove());
System.out.println("remove dump done");
// remove exception test
try {
	que.remove();
} catch (NoSuchElementException e) {
	System.out.println("remove error");
}
System.out.println("--- remove test done. ---");


// poll null test
System.out.println(que.poll());
// poll test
que.add(100);
System.out.println(que.poll());
System.out.println("--- poll test done. ---");


// offer test
que = new Queue<Integer>();
for(i = 0; i < dump; ++i)
	que.offer(i);
System.out.println("offer");
System.out.println("--- offer test done. ---");


// peek and element test
System.out.println(que.peek());
System.out.println(que.peek());
System.out.println(que.element());
System.out.println(que.element());
// element error test
for(i = 0; i < dump; ++i){
	que.remove();
}
try {
	que.element();
}
catch (NoSuchElementException e){
	System.out.println("element error");
}
que.peek();
System.out.println("--- peek and element test done. ---");

// two stack test
for(i = 0; i < dump; ++i) // clear the que
	que.poll();
que.add(1);
que.add(2);
que.add(3);
que.remove();
for(i = 0; i < dump/2; ++i)
	que.add(i);
try {
	que.add(1);
}
catch (IllegalStateException e) {
```

```
                System.out.println("insert overflow");
            }
            que.remove();
            que.remove();
            for(i = 0; i < dump/2; ++i)
                que.remove();
            try {
                que.remove();
            }
            catch (NoSuchElementException e) {
                System.out.println("remove overflow");
            }

        }
}
```