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：计卓 1501

学　　号：U201514898

姓　　名：胡思勖

指导教师：辜希武

2018 年 **05** 月 **03** 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

对于整个容器类的设计而言，需要考虑包括效率、程序健壮性在内的多种因素。除了这些基本因素之外，考虑到总体的需求是使用栈实现队列，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。通过初步考虑，发现对于使用栈实现队列的限制这一方面需要加以关注，避免其带来的程序异常。

二、 系统设计

1. 概要设计

除了构造函数外，整个实验需要实现 6 个接口，而这 6 个接口两两一组完成了整个类提供的 3 种服务：如队列、出队列以及取队列第一个元素的值。由于所有的接口在同一个类中，所以在接方面没有层次结构。考虑到此队列类继承一个栈类并持有一个栈类，其类的 UML 图如图 1 所示。在本实验中，入队列时加入所持有的栈（图中左边的栈），而出队列时从继承而来的栈（图中右边的栈）中拿取。

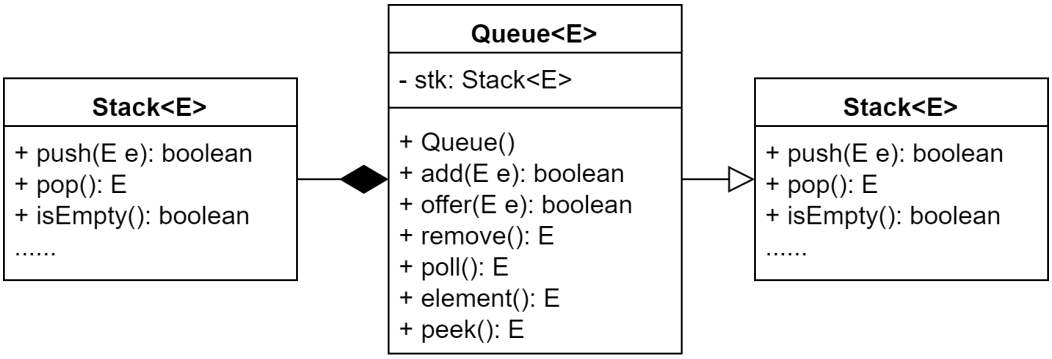

图 1 Queue 类 UML 图

2. 详细设计

1) add 方法

add 方法用于向队列尾部添加一个元素，而从两个栈的角度来看，其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。如果不为空则继续进行接下来的判断：若持有的栈中的元素个数已经达到了 dump 所指定的个数则将持有的栈中的内容逐一弹出并倒入继承而来的栈中，而若此时继承而来的栈不为空，则不能进行这一“倾倒”操作，此时抛出 IllegalStateException。反之，若持有的栈中的元素个数未达到 dump 所指定的个数，则不需要倾倒，直接将 e 加入持有的栈中即可，具体的流程图如图 2 所示。

2) offer 方法

offer 方法与 add 方法即为类似，只不过其不会抛出 IllegalStateException，取而代之的，它在队列处于这种非法的状态下返回 false。其流程图如图 3 所示。

为了能够更好的展示栈中元素的移动过程，图 4 展示了在 3 种不同状态下尝试插入元素时栈中元素的状态。其中，第①部分为持有栈不为空时尝试插入的情况，②是当持有栈满但继承栈为空时尝试插入的情况，③为持有栈满但继承栈不为空的情况。在最后一种情况中视调用的函数为 add 还是 offer 来决定抛出异常或返回 false。

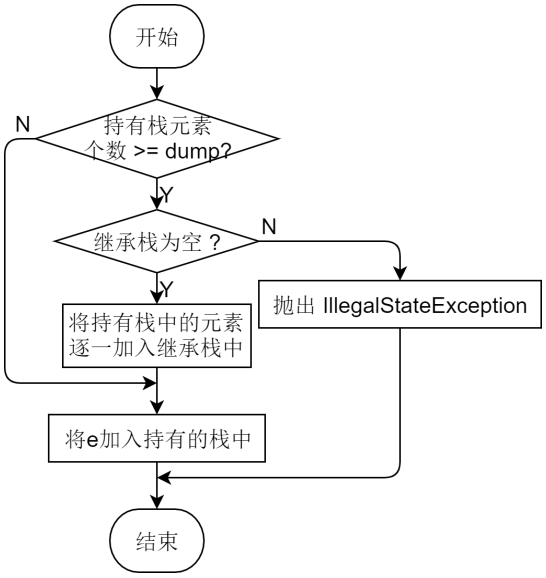

图 2 add 方法流程图

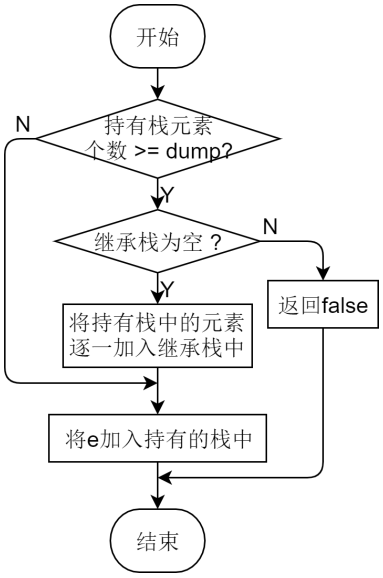

图 3 offer 方法流程图

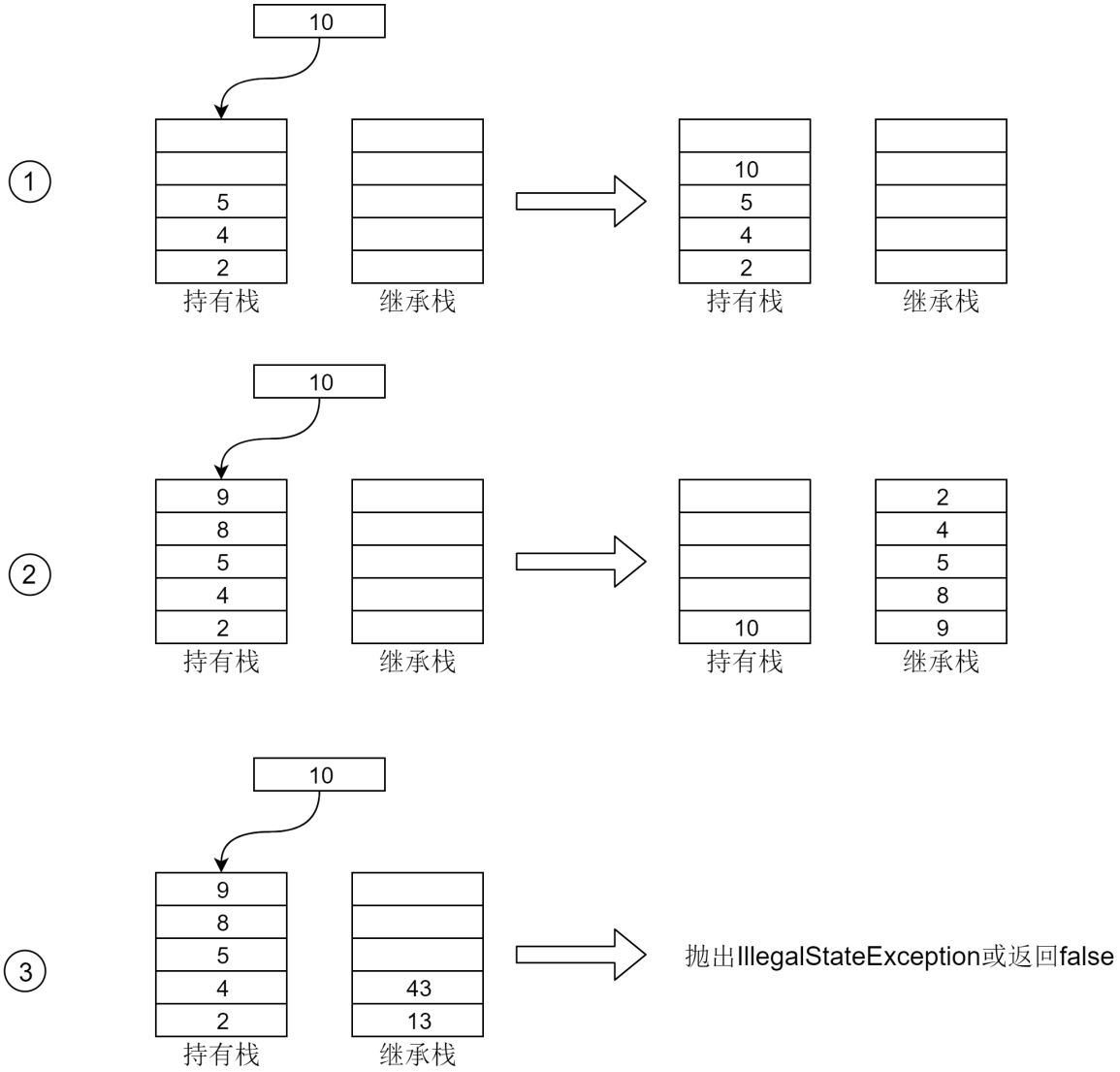

图 4 三种情况下尝试插入时两个栈的状态变化

3) remove 方法

remove 方法用于尝试从队列首部移除一个值并返回这个值。相较于 add/offer 方法，remove 需要考虑的异常要少很多：其只需要考虑在尝试移除时可能的栈为空的情况即可，对于其它可能由于使用了两个栈而导致的与正常队列不同的状况而言，均可以通过将持有的栈中的元素“倒入”继承的栈中解决。其流程图如图 5 所示。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，则表明持有的栈以及继承而来的栈中均没有元素，也即队列为空。此时抛出 NoSuchElementException。

4) poll 方法

与 remove 方法的作用与处理方式类似，这一方法也尝试移走并返回队列顶部的元素。不同的是，这一方法在队列为空（即持有的栈和继承的栈均为空时）返回 null 而不是抛出异常，其流程图如图 6 所示。

在进行 poll 或 remove 操作时，栈中的值的变化情况如图 7 所示。图中，①部分为继承的栈不为空时尝试移除的情况，此时直接中继承的栈顶移除元素，图中持有的栈为空，但在持有的栈不为空的情况下进行的也是相同的操作。②为继承的栈为空但持有的栈不为空的情况，这时需要先将持有的栈中的元素导入继承的栈中，这样才能接触到栈顶的元素，然后从继承栈栈顶弹出即可。这里有一个可能的优化：当持有的栈中有 n 个元素时，可以值倾倒持有栈中的顶部 n-1 个元素到继承的栈中，剩下的最后一个元素直接弹出并返回即可。但相较于全部倾倒实现起来更容易出错，且其对性能的提升可以忽略不计，因此此处未采用这种方法。最后，当两个栈均为空的情况图中并未显示，此时仅抛出异常或返回 null 即可（取决于调用者调用的是哪个函数）。

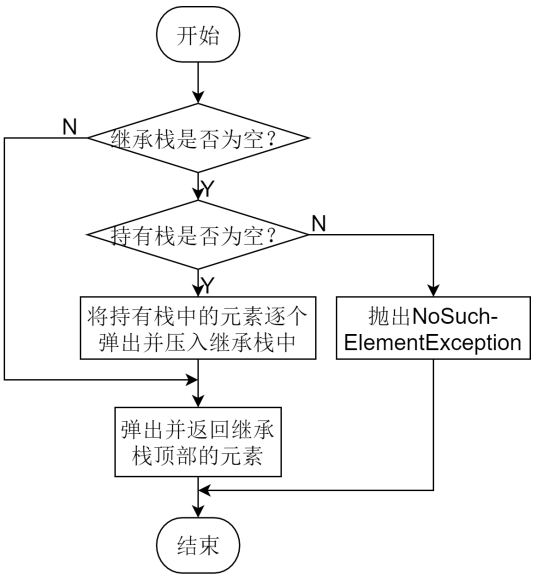

图 5 remove 方法流程图

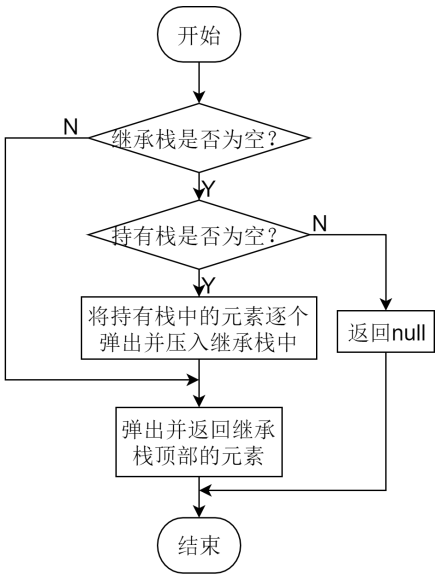

图 6 poll 方法流程图

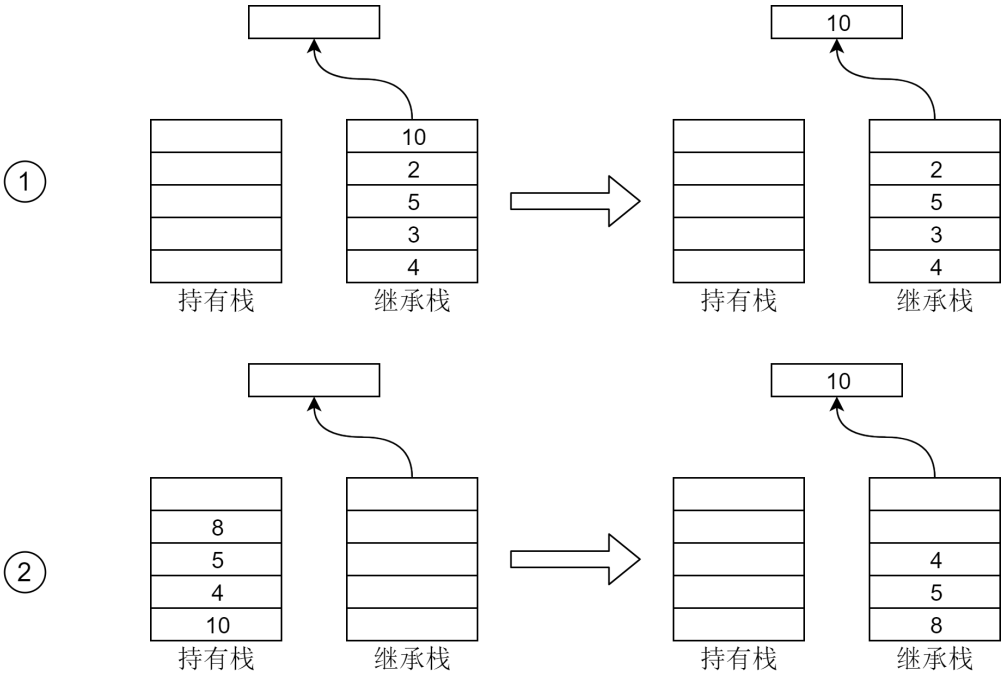

图 7 尝试移除顶端元素时两个栈的状态变化

5) element 方法

element 方法与 remove 方法十分相似，不同的是它不使队列首部的元素出队列。同样的，这一方法的实现需要分为 3 种情况考虑：继承的栈不为空、继承的栈为空但持有的栈不为空、两个栈均为空。第一种情况直接返回继承的栈的栈顶元素即可，第二种情况需要先把持有的栈中的元素逐个弹出并压入继承的栈中，然后返回继承的栈的栈顶元素，最后一种情况下直接抛出 NoSuchElementException 异常即可。其流程图与图 5 类似，只需将图 5 中“结束”前的“弹出并返回继承栈顶部的元素”更改为“返回继承栈顶部的元素”即可，此处不再给出其流程图。

6) peek 方法

除了在两个栈均为空的情况下，peek 方法返回 null 以外，其余的行为均与 element 方法相同，其流程图与图 6 相似，只需将图 6 中“结束”前的“弹出并返回继承栈顶部的元素”更改为“返回继承栈顶部的元素”即可此处不再赘述。

三、 软件开发

本程序以及测试程序的编写均在 Arch Linux x64 系统下完成，具体开发环境见表 1。

表 1 软件开发环境

项目	版本
操作系统	Arch Linux 2018 年 4 月更新
JRE	OpenJDK Runtime Environment 1.8.0_172-b11
JDK	OpenJDK 1.8.0_172, Java 8

四、 软件测试

由于题目未对可能的异常类型以及程序的相应行为做出明确规定，也未给出测试样例，因此处测试的目标为：在这一特定的系统设计下、在测试覆盖所有代码的情况下，程序能够给出与期望相同的结果；对于 java.util.Queue 文档中所提到的功能要求以及可能发生的异常本程序能够严格遵守规范，对于没有提到但可能发生的异常本程序能够依照这一特定的系统设计进行处理。

在这一测试中，将对于每一个方法进行单元测试，所使用的测试代码见源代码部分，此处仅给出测试结果。测试程序的运行结果如图 1 所示。

```
~/H/Java  lab1/src  ls                                  Sun Jun  3 00:38:14 2018
Main.class  Main.java  Queue.class  Queue.java
~/H/Java  lab1/src  java Main                          Sun Jun  3 00:38:18 2018
[x] insert 1
[x] insert 2
[x] offer 1
[x] offer 2
[x] offer 3
[x] remove 1
[x] remove 2
[x] remove 3
[x] poll 1
[x] poll 2
[x] poll 3
[x] element 1
[x] element 2
[x] element 3
[x] peek 1
[x] peek 2
[x] peek 3
~/H/Java  lab1/src                                     Sun Jun  3 00:38:19 2018
```

图 8 测试程序运行结果

从图中可以看出，对于 Queue 类的每个方法的测试均通过，程序没有抛出异常，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 erase 和 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，且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。

六、 过程和体会

1. 遇到的主要问题和解决方法

在早期的设计中，错误的对于 null 元素的插入抛出了 NullPointerException，经过仔细的检查以及对于文档的阅读，发现 Queue 类是允许插入 null 而不会抛出任何异常的，遂对程序进行修改，使之能够插入 null 元素。此外，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

搭建好开发环境后，在 bash 下使用如下命令编译和运行程序：

```
javac Queue.java
javac Main.java
java  Main
```

3. 源代码

Queue.java

```
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() {
        stk = new Stack<E>();
    }

    /// @brief add an element to the end of the queue
    /// @param e the element to add
    /// @throw IllegalStateException when the queue is full
    /// @throw NullPointerException when the element is null
    /// @return if element is successfully added
    public boolean add(E e) throws IllegalStateException,
ClassCastException, NullPointerException,
IllegalArgumentException {
        if(e == null)
            throw new NullPointerException();
        if (stk.size() >= dump){
            if(!this.isEmpty())
                throw new IllegalStateException();
            else {
                while(!stk.isEmpty())
                    this.push(stk.pop());
                stk.push(e);
            }
        } else {
            stk.push(e);
        }
        return true;
    }

    /// @brief add an element to the end of the queue
    /// @param e the element to add
    /// @throw NullPointerException when the element is null
    /// @return if element is successfully added
    public boolean offer(E e) throws ClassCastException,
NullPointerException, IllegalArgumentException {
        if(e == null)
            throw new NullPointerException();
        if (stk.size() >= dump){
            if(!this.isEmpty())
                return false;
            else{
                while(!stk.isEmpty())
                    this.push(stk.pop());
                stk.push(e);
```

```
            }
        } else {
            stk.push(e);
        }
        return true;
    }

    /// @brief remove an element from the head of the queue
    /// @throw NoSuchElementException when the queue is empty
    /// @return the element removed
    public E remove() throws NoSuchElementException {
        if(!this.isEmpty()){
            return this.pop();
        } else if(!stk.isEmpty()){
            while(!stk.isEmpty())
                this.push(stk.pop());
            return this.pop();
        } else {
            throw new NoSuchElementException();
        }
    }

    /// @brief remove an element from the head of the queue
    /// @return the element removed, null if no element in queue
    public E poll() {
        if(!this.isEmpty()){
            return this.pop();
        } else if(!stk.isEmpty()){
            while(!stk.isEmpty())
                this.push(stk.pop());
            return this.pop();
        } else {
            return null;
        }
    }

    /// @brief get the first element in the queue
    /// @throw NoSuchElementException if the queue is empty
    /// @return the first element in the queue
    public E element() throws NoSuchElementException {
        if(!this.isEmpty()){
            return super.peek();
        } else if(!stk.isEmpty()){
            while(!stk.isEmpty())
                this.push(stk.pop());
```

```
            return super.peek();
        } else {
            throw new NoSuchElementException();
        }
    }

    /// @brief get the first element in the queue
    /// @return the first element in the queue, or null if the queue
is empty
    public E peek() {
        if(!this.isEmpty()){
            return super.peek();
        } else if(!stk.isEmpty()){
            while(!stk.isEmpty())
                this.push(stk.pop());
            return super.peek();
        } else {
            return null;
        }
    }
}
```

Main.java

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

        // add test
        for(i = 0; i < que.dump; ++i)
            assert que.add(i);
        System.out.println("[x] insert 1");
        for(i = 0; i < que.dump; ++i)
            assert que.add(i);
        System.out.println("[x] insert 2");
        flag = false;
        try{
            for(i = 0; i < que.dump; ++i)
                que.add(i);
        } catch (IllegalStateException e) {
            System.out.println("[x] insert 3");
            flag = true;
```

```
        }
        assert flag;

        // offer test
        que = new Queue<Integer>();
        for(i = 0; i < que.dump; ++i)
            assert que.offer(i);
        System.out.println("[x] offer 1");
        for(i = 0; i < que.dump; ++i)
            assert que.offer(i);
        System.out.println("[x] offer 2");
        for(i = 0; i < que.dump; ++i)
            assert !que.offer(i);
        System.out.println("[x] offer 3");

        // remove test
        for(i = 0; i < que.dump; ++i)
            assert que.remove() == i;
        System.out.println("[x] remove 1");
        for(i = 0; i < que.dump; ++i)
            assert que.remove() == i;
        System.out.println("[x] remove 2");
        flag = false;
        try {
            for(i = 0; i < que.dump; ++i)
                que.remove();
        } catch (NoSuchElementException e) {
            System.out.println("[x] remove 3");
            flag = true;
        }
        assert flag;

        // poll test
        for(i = 0; i < 2 * que.dump; ++i)
            assert que.add(i);
        System.out.println("[x] poll 1");
        for(i = 0; i < 2 * que.dump; ++i)
            assert que.poll() == i;
        System.out.println("[x] poll 2");
        for(i = 0; i < que.dump; ++i)
            assert que.poll() == null;
        System.out.println("[x] poll 3");

        // element test
        que = new Queue<Integer>();
```

```
        for(i = 0; i < que.dump + 1; ++i){
            que.add(i);
            assert que.element() == 0;
        }
        System.out.println("[x] element 1");
        for(i = 0; i < que.dump + 1; ++i){
            assert que.element() == i;
            que.remove();
        }
        System.out.println("[x] element 2");
        flag = false;
        try{
            que.element();
        } catch (NoSuchElementException e) {
            System.out.println("[x] element 3");
            flag = true;
        }
        assert flag;

        // peek test
        que = new Queue<Integer>();
        for(i = 0; i < que.dump + 1; ++i){
            que.add(i);
            assert que.peek() == 0;
        }
        System.out.println("[x] peek 1");
        for(i = 0; i < que.dump + 1; ++i){
            assert que.peek() == i;
            que.remove();
        }
        System.out.println("[x] peek 2");
        assert que.peek() == null;
        System.out.println("[x] peek 3");
    }
}
```