

**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 ： 计算机1508**

**学 号 ： U201514699**

**姓 名 ： 张 鑫**

**指导教师 ： 纪俊文**

**2018 年 4 月 21 日**

目录

[一 需求分析 - 1 -](#_Toc512092091)

[1 题目要求 - 1 -](#_Toc512092092)

[2 需求分析 - 1 -](#_Toc512092093)

[二 系统设计 - 2 -](#_Toc512092094)

[1 概要设计 - 2 -](#_Toc512092095)

[2 详细设计 - 3 -](#_Toc512092096)

[三 软件开发 - 4 -](#_Toc512092097)

[四 软件测试 - 4 -](#_Toc512092098)

[五 特点与不足 - 7 -](#_Toc512092099)

[1 技术特点 - 7 -](#_Toc512092100)

[2 不足和改进的建议 - 8 -](#_Toc512092101)

[六 过程和体会 - 8 -](#_Toc512092102)

[1 遇到的主要问题和解决方法 - 8 -](#_Toc512092103)

[2 课程设计的体会 - 8 -](#_Toc512092104)

[七 源码和说明 - 8 -](#_Toc512092105)

[1 文件清单及其功能说明 - 8 -](#_Toc512092106)

[2 用户使用说明书 - 9 -](#_Toc512092107)

[3 源代码 - 9 -](#_Toc512092108)

# 

# 需求分析

## 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=1024;

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？

## 需求分析

栈的特点是后进先出，要求用两个栈协同实现一个队列，队列的特点是先进先出，要提供入队方法、出队方法、返回队尾元素但不出队的方法。当队列已满，还要继续入队时，需要抛出IllegalStateException异常；当队列为空，还需要出队时，要抛出NoSuchElementException异常

# 系统设计

## 概要设计

队列继承了一个栈，是它的父类，还实例化了一个栈，是它的一个元素，由于栈的后进先出特性与队列的先进先出特性不符合，所以要将一个栈的元素通过另一个栈倒置一下才能实现队列，这也就是实现的机理，一个栈主要是用来存放元素的，另一个栈是辅助倒置元素的。我们用实例化的栈stk作为存放元素的栈，父类super栈作为辅助栈。

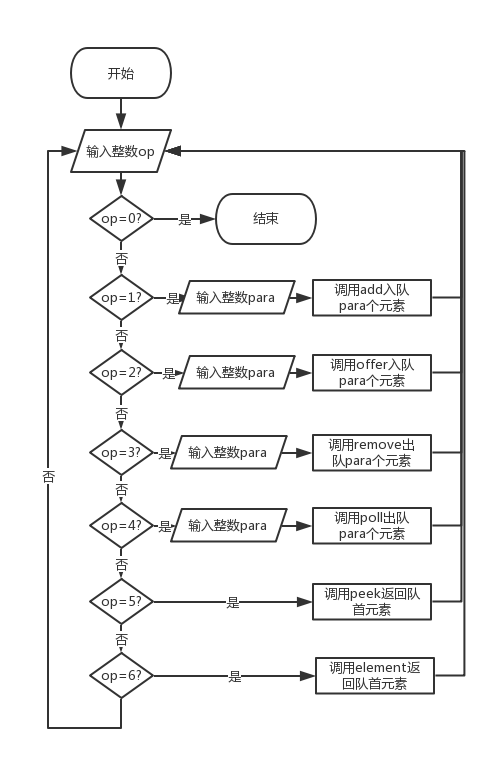


图 二.1 总体结构图

首先是入队，当stk的元素个数没有达到dump时，一直向stk入栈；否则，将stk的元素全部出栈，依次入栈到super，然后继续向stk入栈。出队时，如果super有元素，则将站定元素出栈，否则，将stk的元素全部导入super，再将super栈顶元素出栈。

队列的方法具体包括：add、offer、remove、poll、peek、element，前2个为入队方法，参数为入队元素，后4个为出队方法，不需要参数，返回值为出队元素。

完成功能方法后还需要进行测试，使用简单的字符菜单界面，用户每次输入数字以测试不同的功能。该菜单采用死循环结构，当输入0时退出循环，其他选项对应功能测试，总体流程图如图 二.1。

## 详细设计

1. **功能函数**
2. add

功能：入队一个元素；

入口参数：E（泛型）类型的元素e；

出口参数：Boolean型，true表示入队成功；

实现：先判断stk栈是否已经有dump个元素，若没有，则将元素压到stk栈，返回true；否则，判断super栈是否为空，若是，则将stk的元素依次出栈，并压栈到super栈中，这样，stk栈就为空，又可以重新压栈，将新的元素压入stk，返回true；若super不为空，则不可以继续压栈，此时需要抛出异常。

1. offer

功能：入队一个元素；

入口参数：E（泛型）类型的元素e；

出口参数：Boolean型，true表示入队成功，false表示入队失败；

实现：实现逻辑与add函数完全相同，唯一不同的是，当队列已满，不可以继续入队时，返回false，而不是抛出异常。

1. remove

功能：出队一个元素；

入口参数：无；

出口参数：E（泛型）类型，为出队元素的值；

实现：所有的出队都要在super栈中完成，先判断super栈是否为空，若不为空，则出栈一个元素，返回其值；否则，再判断stk栈是否为空，若不为空，则将stk栈的元素依次出栈，压栈到super中，此时super又有元素，又可以出栈了，将其栈顶元素出栈并返回；若stk和super都为空，则没有元素可以出栈，抛出一个异常。

1. poll

功能：出队一个元素；

入口参数：无；

出口参数：E（泛型）类型，为出队元素的值；

实现：实现逻辑与remove完全相同，唯一不同的是，当两个栈都为空时，不抛出异常，而返回null。

1. peek

功能：返回队尾元素的值；

入口参数：无；

出口参数：E（泛型）类型，为出队元素的值；

实现：实现的基本逻辑同remove，不过这里只返回元素值，而不真正删除元素，所以在队堆栈进行操作时，不能用出栈操作pop，而要用堆栈中的peek方法取值。当没有元素可以出队时，返回null。

1. element

功能：返回队尾元素的值；

入口参数：无；

出口参数：E（泛型）类型，为出队元素的值；

实现：实现逻辑与peek完全相同，唯一不同的是，当队列为空时，抛出异常，而不返回值。

1. **用户界面**

用一个while死循环实现，先打印提示信息，引导用户输入功能编号，获取输入后，用switch-case语句判断编号，分别执行不同的功能，有些功能，比如入队和出队，还需要输入入队和出队的元素个数。入队元素不需要手动输入，用随机函数生成。用try-catch语句块处理异常，当队列空、满等异常情况发生时，均能打印出相关的提示信息。

# 软件开发

在Windows10系统下进行开发，编译环境为eclipse\_win32\_v4.7.0，编译器选项为Java1.8版本。

在eclipse中打开该工程，点击工具栏的运行按钮即可运行，在底部或者右侧的控制台能够看到提示信息，根据提示进行操作。

调试工具也是eclipse，由于Java的println函数基本上可以输出所有的类型，而且还有很明确的报错信息，当遇到异常时能够定位错误，还可以将错误打印出来，比较容易调试。

# 软件测试

1. 测试思路：

先用add方法向队列添加元素，再用remove删除元素；用offer方法添加元素，再用poll删除元素；再前两组测试中，顺带测试peek和element方法，因为这两个方法只取元素值，不改变队列元素，所以可以一起测试。测试用例的设计要包含正常情况和异常情况。

1. 编译运行测试程序，输出菜单信息如图 四.1所示，输入0-6分别对应7个功能。

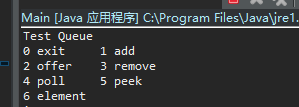


图 四.1 菜单

1. 选择功能1，用add方法入队，参数设为5，入队5个随机数，如图 四.2所示。

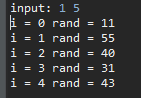


图 四.2 入队5个元素

由于dump的值为10，所以最多可以一次性入队20个元素，为了测试异常情况，再次入队20个元素，加上已经存在的5个元素，一共25个，理应会发生异常，测试结果如图 四.3所示。可见入队15个元素以后，就提示队列已满，后面没有继续入队。

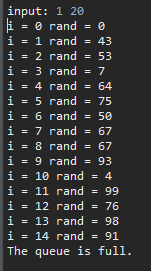


图 四.3 入队20个元素

1. 选择功能3，用remove进行出队，先出队10个元素，如图 四.4，可见出队的元素与最初入队的10个元素相同。

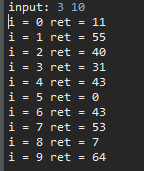


图 四.4 出队10个元素

出队10个元素以后，队列中还有10个元素，为了测试异常情况，此时出队15个元素，就可以触发队列为空时继续出队的异常，如图 四.5，在出队了10个元素以后，打出提示信息，表示队列已经为空，异常捕获成功。

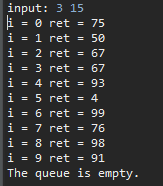


图 四.5 出队15个元素

1. 选择功能2，用offer方法进行入队，先入队3个元素，如图 四.6所示。

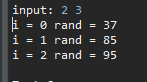


图 四.6 用offer入队

1. 选择功能5，用peek返回队头元素，如图 四.7所示，元素值37是前面第一个入队的元素；



图 四.7 用peek返回元素

选择功能6，用element返回队头元素，如图 四.8所示，返回的结果与peek相同，因为两者在正常情况下的功能相同。



图 四.8 用element返回元素

1. 继续选择功能2，在原有的3个元素基础上入队20个元素，此时触发入队异常，如图 四.9所示，在入队17个元素以后打印出了异常信息。

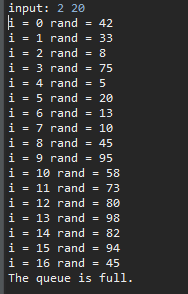


图 四.9 用offer入队

1. 选择功能4，用poll出队，先出队8个元素，如图 四.10所示，出队的元素与最初入队的8个元素相同。

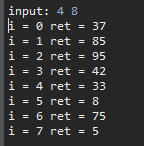


图 四.10 用poll出队

再次用功能4出队20个元素，此时队列中只有12个元素，所以会触发异常，如图 四.11所示，出队12个元素以后，打印出了异常信息。

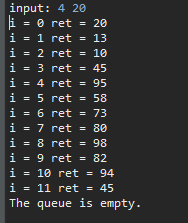


图 四.11 用poll出队异常

1. 此时队列已经为空，peek和element的异常情况还没有测试。选择功能5，用peek和element分别获取队首元素，结果如图 四.12所示，两者均提示队列为空。

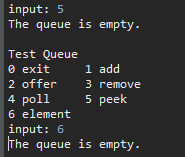


图 四.12 peek和element异常

至此，基本功能测试完毕，每个方法的测试都涵盖了正常情况和异常情况。

# 特点与不足

## 技术特点

实现了题目要求的基本功能，对于异常的处理比较合理，任何异常都能够抛出并处理，没有因为异常而导致程序崩溃。测试用例的设计也算合理，涵盖了大部分情况，测试时可以任意操作，程序容错性较强。

## 不足和改进的建议

没有实现附加的功能，没有实现clone、equals等方法。

# 过程和体会

## 遇到的主要问题和解决方法

最主要的问题是编译的过程中的一个比较奇怪的报错，错误定位在stack的pop方法处，而且该信息是死循环，一开始以为是语法错误，后来发现语法上没有问题。经过查找资料才明白，是因为stack类的pop方法会调用stack的peek方法，而stack的peek方法在queue中被覆盖了，所以实际调用的是queue中的peek方法，然而queue中的peek方法又调用了stack的peek方法，这样就造成了循环调用。

该问题的解决方法比较巧妙，那就是用stk栈作为入队的栈，super作为倒置元素的栈，这样，即使peek方法被覆盖，它的功能对于stack来说也是正确的，不会出现死循环。

## 课程设计的体会

实验的难度不算大，但还是花费了较多时间。

首先，这是第一次写Java代码，原来没有接触过，所以很多东西都需要现学现用，甚至包括编译环境的配置都没有接触过。

其次，在前面提到的问题上浪费了较多时间，对代码修改了很多次都排除不了那个错误，原因是两个栈的使用顺序错误了，最后重写了代码才解决问题。

现在可以说对Java基本入门了，熟悉了eclipse环境，习惯了Java语言的设计思想，但还有很大的提升控件。

# 源码和说明

## 文件清单及其功能说明

实验项目的文件目录如图 七.1所示，源文件在src目录下，其中Queue.java是队列类的实现，Main.java是测试程序所在的类。

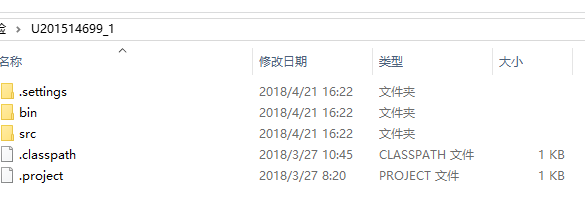


图 七.1 文件清单

## 用户使用说明书

图 七.1中的工程可以直接在eclipse中打开，如图 七.2所示，选择：文件-Open Projects form File System，找到工程文件夹的位置，点击确定，既可打开。打开以后，在工具栏中找到绿色的运行按钮，即可按键运行。

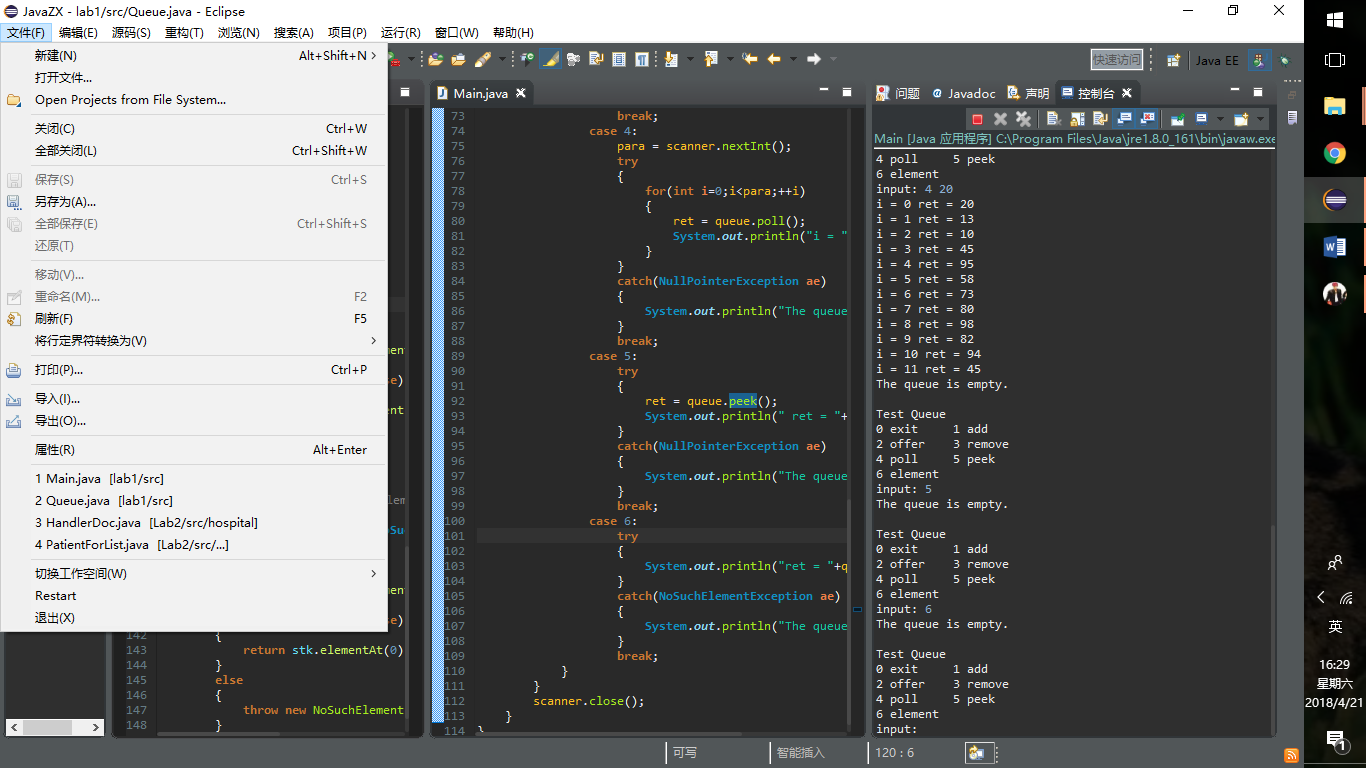


图 七.2 使用说明

## 源代码

Queue.java：

import java.util.Stack;

import java.util.NoSuchElementException;

public class Queue<E> extends Stack<E>

{

/\*\*

\* 自定义的泛型队列

\*/

private static final long serialVersionUID = -535350814358605791L;

public final int dump = 10;

private Stack<E> stk;

public Queue( )

{

stk = new Stack<E>();

}

/\*\*

\*增加一个元索

\*如果队列已满，则抛出一个IllegalStateException异常

\*/

public boolean add(E e) throws IllegalStateException

{

//stk未满

if(stk.size()<dump)

{

stk.push(e);

return true;

}

//super为空

else if(super.empty())

{

//将stk的元素倒入super中

for(;stk.empty()==false;)

{

super.push(stk.pop());

}

stk.push(e);

return true;

}

//队列已满，抛出异常

else

{

throw new IllegalStateException();

}

}

/\*\*

\* 添加一个元素并返回true

\* 如果队列已满，则返回false

\*/

public boolean offer(E e)

{

if(stk.size()<dump)

{

stk.push(e);

return true;

}

else if(super.empty())

{

for(;stk.empty()==false;)

{

super.push(stk.pop());

}

stk.push(e);

return true;

}

else

{

return false;

}

}

/\*\*

\* 移除并返回队列头部的元素

\* 如果队列为空，则抛出一个NoSuchElementException异常

\*/

public E remove( ) throws NoSuchElementException

{

E ret;

//super不为空，则从中出栈

if(super.empty()==false)

{

ret = super.pop();

}

//stk不为空

else if(stk.empty()==false)

{

//将stk的元素倒入super

for(;stk.empty()==false;)

{

super.push(stk.pop());

}

ret = super.pop();

}

//两者皆空，抛出异常

else

{

throw new NoSuchElementException();

}

return ret;

}

/\*\*

\* 移除并返问队列头部的元素

\* 如果队列为空，则返回null

\*/

public E poll( )

{

E ret;

if(super.empty()==false)

{

ret = super.pop();

}

else if(stk.empty()==false)

{

for(;stk.empty()==false;)

{

super.push(stk.pop());

}

ret = super.pop();

}

else

{

return null;

}

return ret;

}

/\*\*

\* 返回队列头部的元素

\* 如果队列为空，则返回null

\*/

public E peek ( )

{

//super不为空，取栈顶元素

if(super.empty()==false)

{

return super.lastElement();

}

//stk不为空，取栈底元素

else if(stk.empty()==false)

{

return stk.firstElement();

}

return null;

}

/\*\*

\* 返回队列头部的元素

\* 如果队列为空，则抛出一个NoSuchElementException异常

\*/

public E element( ) throws NoSuchElementException

{

if(super.empty()==false)

{

return super.lastElement();

}

else if(stk.empty()==false)

{

return stk.elementAt(0);

}

else

{

throw new NoSuchElementException();

}

}

}

Main.java：

import java.util.NoSuchElementException;

import java.util.Scanner;

public class Main

{

public static void main (String[] args)throws

IllegalStateException,NullPointerException,NoSuchElementException

{

Queue<Integer> queue = new Queue<Integer>();

Scanner scanner = new Scanner(System.in);

int op,para,ret;

boolean exit = true;

while(exit)

{

System.out.print(

"\nTest Queue\n"+

"0 exit\t 1 add\n"+

"2 offer\t 3 remove\n"+

"4 poll\t 5 peek\n"+

"6 element\n"+

"input: "

);

op = scanner.nextInt();

switch(op)

{

case 0:

exit = false;

break;

case 1:

para = scanner.nextInt();

try

{

for(int i=0;i<para;++i)

{

int rand = (int)(Math.random()\*100);

queue.add(rand);

System.out.println("i = "+i+" rand = "+rand);

}

}

catch(IllegalStateException ae)

{

System.out.println("The queue is full.");

}

break;

case 2:

para = scanner.nextInt();

for(int i=0;i<para;++i)

{

int rand = (int)(Math.random()\*100);

boolean retb = queue.offer(rand);

if(retb==false)

{

System.out.println("The queue is full.");

break;

}

System.out.println("i = "+i+" rand = "+rand);

}

break;

case 3:

para = scanner.nextInt();

try

{

for(int i=0;i<para;++i)

{

ret = queue.remove();

System.out.println("i = "+i+" ret = "+ret);

}

}

catch(NoSuchElementException ae)

{

System.out.println("The queue is empty.");

}

break;

case 4:

para = scanner.nextInt();

try

{

for(int i=0;i<para;++i)

{

ret = queue.poll();

System.out.println("i = "+i+" ret = "+ret);

}

}

catch(NullPointerException ae)

{

System.out.println("The queue is empty.");

}

break;

case 5:

try

{

ret = queue.peek();

System.out.println(" ret = "+ret);

}

catch(NullPointerException ae)

{

System.out.println("The queue is empty.");

}

break;

case 6:

try

{

System.out.println("ret = "+queue.element());

}

catch(NoSuchElementException ae)

{

System.out.println("The queue is empty.");

}

break;

}

}

scanner.close();

}

}