王昊

Anhui Normal University

实验四 多线程与异常处理

[第一章 实验目的 3](#_Toc439106186)

[实验 1 4](#_Toc439106187)

[1.0实验内容与要求 4](#_Toc439106188)

[1.1 编程分析 4](#_Toc439106189)

[1.2 源代码 4](#_Toc439106190)

[1.3 运行结果 7](#_Toc439106191)

[1.3.1 还没开始 7](#_Toc439106192)

[1.3.2 红色移动 8](#_Toc439106193)

[1.3.3 绿色移动 8](#_Toc439106194)

[2.3.4 蓝色三倍速 8](#_Toc439106195)

[1.4 调试情况分析 9](#_Toc439106196)

[实验 2 10](#_Toc439106197)

[2.0实验内容与要求 10](#_Toc439106198)

[2.1 编程分析 10](#_Toc439106199)

[2.2 源代码 10](#_Toc439106200)

[2.3 运行结果 10](#_Toc439106201)

[2.4 调试情况分析 12](#_Toc439106202)

[实验 3 13](#_Toc439106203)

[3.0实验内容与要求 13](#_Toc439106204)

[3.1 编程分析 13](#_Toc439106205)

[3.2 源代码 13](#_Toc439106206)

[3.3 运行结果 15](#_Toc439106207)

[3.4 调试情况分析 17](#_Toc439106208)

[实验 4 18](#_Toc439106209)

[4.0实验内容与要求 18](#_Toc439106210)

[4.1 编程分析 18](#_Toc439106211)

[4.2 源代码 18](#_Toc439106212)

[4.3 运行结果 20](#_Toc439106213)

[4.4 调试情况分析 21](#_Toc439106214)

[实验 5 思考题 22](#_Toc439106215)

[5. 1异常分为哪两类？请简单介绍。 22](#_Toc439106216)

[5.2 多线程的同步处理用于什么情况? 22](#_Toc439106217)

# 第一章 实验目的

* 掌握使用Thread的子类创建线程
* 学习使用Thread类创建线程
* 学习处理线程同步问题
* 理解什么是异常
* 掌握java的异常处理机制和方法

# 实验 1

## 1.0实验内容与要求

* 双线程接力。编写一个应用程序，除了主线程外，还有两个线程：first和second。first负责模拟一个红色的按钮从坐标(10，60)运动到(100，60)；second负责模拟一个绿色的按钮从坐标(100，60)运动到(200，60)。

## 1.1 编程分析

moveComponent方法由synchronized关键字修饰，说明在同一时间只有一个线程可以调用这个方法。然后moveComponent里面判断是哪个线程调用自己的，然后根据distance的值移动相应的按钮。

## 1.2 源代码

package exp4\_1;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

public class MoveButton extends JFrame implements Runnable, ActionListener {

Thread first, second, third;//用Thread类声明first,second两个线程对象

JButton redButton, greenButton, blueButton, startButton;

int distance = 10;

MoveButton() {

first = new Thread(this); //创建first线程，当前窗口做为该线程的目标对象

second = new Thread(this); //创建second线程，当前窗口做为该线程的目标对象

third = new Thread(this);

redButton = new JButton();

greenButton = new JButton();

blueButton = new JButton();

redButton.setBackground(Color.red);

greenButton.setBackground(Color.green);

blueButton.setBackground(Color.blue);

startButton = new JButton("start");

startButton.addActionListener(this);

setLayout(null);

add(redButton);

redButton.setBounds(10, 60, 15, 15);

add(greenButton);

greenButton.setBounds(100, 60, 15, 15);

add(blueButton);

blueButton.setBounds(200, 60, 15, 15);

add(startButton);

startButton.setBounds(10, 100, 100, 30);

setBounds(0, 0, 300, 200);

setVisible(true);

validate();

addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent e) {

System.exit(0);

}

}

);

}

public void actionPerformed(ActionEvent e) {

try {

first.start();

second.start();

third.start();

} catch (Exception exp) {

}

}

public void run() {

while (true) {

if (Thread.currentThread() == first) //判断当前占有CPU资源的线程是否是first

{

moveComponent(redButton);

try {

Thread.sleep(20);

} catch (Exception exp) {

}

}

if (Thread.currentThread() == second) //判断当前占有CPU资源的线程是否是second

{

moveComponent(greenButton);

try {

Thread.sleep(10);

} catch (Exception exp) {

}

}

if (Thread.currentThread() == third) //判断当前占有CPU资源的线程是否是third

{

moveComponent(blueButton);

try {

Thread.sleep(5);

} catch (Exception exp) {

}

}

}

}

public synchronized void moveComponent(Component b) {

if (Thread.currentThread() == first) {

while (distance > 100 && distance <= 300) {

try {

wait();

} catch (Exception exp) {

}

}

distance = distance + 1;

b.setLocation(distance, 60);

if (distance >= 100) {

b.setLocation(10, 60);

notifyAll();

}

}

if (Thread.currentThread() == second) {

while ((distance >= 10 && distance < 100) || (distance >= 200 && distance < 300)) {

try {

wait();

} catch (Exception exp) {

}

}

distance = distance + 1;

b.setLocation(distance, 60);

if (distance >= 200) {

b.setLocation(100, 60);

notifyAll();

}

}

if (Thread.currentThread() == third) {

while (distance >= 10 && distance < 200) {

try {

wait();

} catch (Exception exp) {

}

}

distance = distance + 1;

b.setLocation(distance, 60);

if (distance > 300) {

distance = 10;

b.setLocation(200, 60);

notifyAll();

}

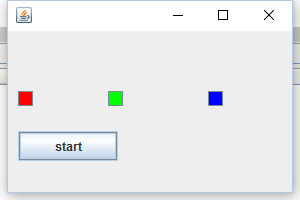
}

}

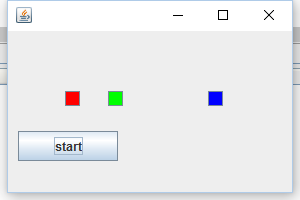
}

## 1.3 运行结果

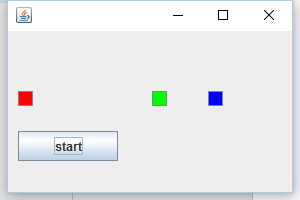
### 1.3.1 还没开始



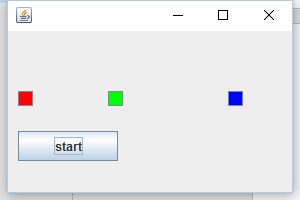
### 1.3.2 红色移动



### 1.3.3 绿色移动



### 2.3.4 蓝色三倍速



## 1.4 调试情况分析

moveComponent的写法想了一会才想通……

# 实验 2

## 2.0实验内容与要求

仔细阅读下面的JAVA语言源程序，请对程序进行分析。

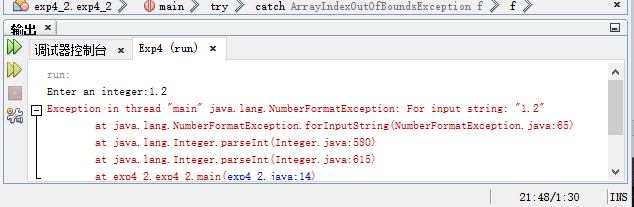
## 2.1 编程分析

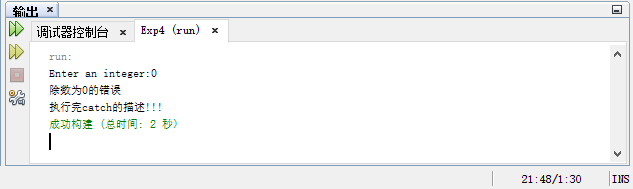
这个程序接受一个输入(整数)，如果输入的是0，那么会引发ArithmeticException，被catch然后println相应语句，如果是其他整数，因为试图向数组外部写入，会引发ArrayIndexOutOfBoundsException，被catch然后println相应提示。其他输入将会导致程序异常中止。

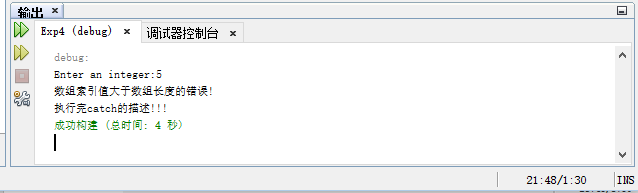
## 2.2 源代码

略

## 2.3 运行结果







## 2.4 调试情况分析

无

# 实验 3

## 3.0实验内容与要求

利用多线程求解某范围素数，每个线程负责10000的范围。

线程1找1至10000的素数

线程2找10001至20000的素数

线程3找20001至30000的素数

要求将找到的素数写入到当前项目文件夹的文件prime.dat中。

## 3.1 编程分析

没有说写入的素数需不需要排序欸……那就当作需要按照升序来写入。

1.建立一个FindPrime类，实现runnable接口。构造函数接受三个参数，一个是起始数字，一个是结束数字，还有一个是Vector对象。当线程运行的时候，从start到end判断是不是素数，是的话将其插入到Vector对象。

2.在主类中建立三个线程，运行。当三个线程都结束了，对Vector排序（三个线程运行的顺序不知道，Vector的内容一定是乱的）。然后将Vector的内容写入文件。

## 3.2 源代码

**//FindPrime.java**

package exp4\_3;

import java.util.Vector;

public class FindPrime implements Runnable {

int start, end;

Vector primes;

public static boolean isPrime(int num) {

for (int i = 2; i <= (int) Math.sqrt(num); i++) {

if (num % i == 0) {

return false;

}

}

return true;

}

FindPrime(int s, int e, Vector p) {

start = s;

end = e;

primes = p;

}

@Override

public void run() {

for (int i = start; i <= end; i++) {

if (isPrime(i)) {

primes.add(i);

}

}

}

}

**Exp4\_3.java**

package exp4\_3;

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

import java.io.Writer;

import java.util.Vector;

import java.util.Collections;

import java.util.Comparator;

public class exp4\_3 {

public static void main(String[] args) {

File file = new File("prime.dat");

Thread t1, t2, t3;

Vector pl = new Vector();

t1 = new Thread(new FindPrime(2, 10000, pl));

t2 = new Thread(new FindPrime(10001, 20000, pl));

t3 = new Thread(new FindPrime(20001, 30000, pl));

try {

t1.start();

t2.start();

t3.start();

} catch (Exception e) {

System.out.println("Can't start threads");

return;

}

try {

Writer writer = new FileWriter(file);

while (true) {

if (!t1.isAlive() && !t2.isAlive() && !t3.isAlive()) {

break;

}

}

Collections.sort(pl, new SortNumber());

for(int i = 0;i<pl.size();i++){

writer.write((pl.get(i)).toString());

writer.write("\n");

}

writer.close();

} catch (IOException e) {

System.out.println("Error:" + e);

}

}

}

class SortNumber implements Comparator {

@Override

public int compare(Object o1, Object o2) {

int a = (int) o1;

int b = (int) o2;

if (a > b) {

return 1;

} else if (a < b) {

return -1;

} else {

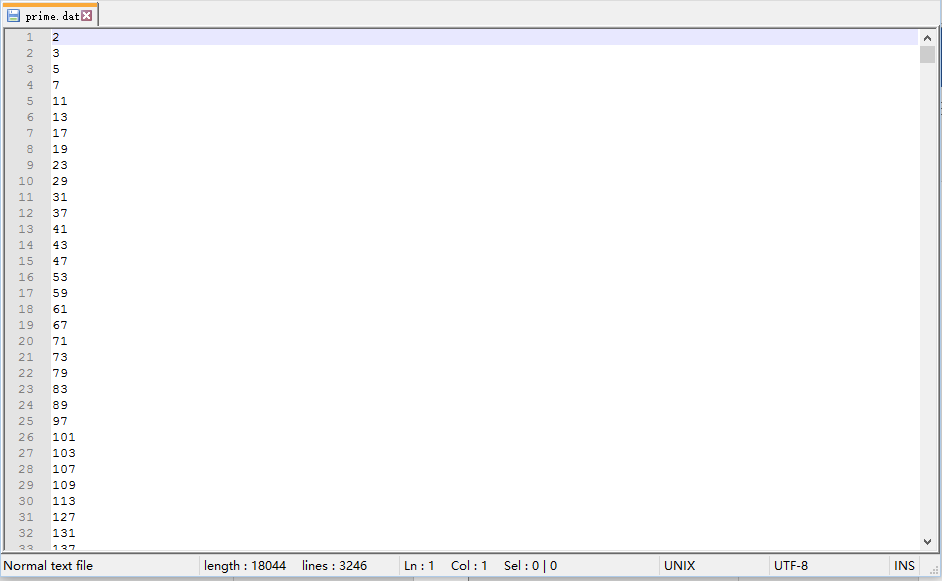
return 0;

}

}

}

## 3.3 运行结果



## 3.4 调试情况分析

一开始用的ArrayList，但是报错，一查说是ArrayList不是线程安全的……多线程操作的时候会出错，用Vector就好了，Vector的操作是Synchronized（同步的），但是他比ArrayList慢。

# 实验 4

## 4.0实验内容与要求

自定义异常类。请为下面的Date2类定义一个异常类DateException，当set方法的3个整型参数不能构成一个合法的日期时候，抛出异常，Date2类中的构造方法调用set方法时将捕获异常并显示异常信息。并设计主类。

## 4.1 编程分析

主要就是要给DateException重写下toString方法，因为给定的代码有这么一句：

catch (DateException e) {

System.out.println(e);

}

如果没有重写toString方法将不会显示错误信息而是写出DateException的类型名称。

其他的比如判断日期合法与否，生成错误信息之类的都很简单……

## 4.2 源代码

package exp4\_4;

import java.util.Scanner;

class DateException extends Exception {

String message;

public DateException(int y, int m, int d) {

message = y + "." + m + "." + d + " is not a valid date";

}

@Override

public String toString(){

return message;

}

}

public class Date2 {

private int year, month, day; //私有的成员变量

public Date2(int y, int m, int d) //指定参数的构造方法

{

try {

set(y, m, d);

System.out.println(year + "年" + month + "月" + day + "日，日期合法！");

} catch (DateException e) {

System.out.println(e);

}

}

public void set(int y, int m, int d) throws DateException //公有的成员方法，设置日期值，请补充代码

{

boolean flag = false;

if (d < 0) {

flag = true;

}

if (m < 0 && m > 12) {

flag = true;

} else if (m == 2) {

if (y % 400 == 0 || (y % 4 == 0 && y % 100 != 0)) {

if (d > 29) {

flag = true;

}

} else {

if (d > 28) {

flag = true;

}

}

} else if (m == 1 || m == 3 || m == 5 || m == 7 || m == 8 || m == 10 || m == 12) {

if (d > 31) {

flag = true;

}

} else {

if (d > 30) {

flag = true;

}

}

if (flag) {

throw new DateException(y, m, d);

} else {

year = y;

month = m;

day = d;

}

}

public static void main(String[] args){

int y,m,d;

Scanner input = new Scanner(System.in);

y = input.nextInt();

m=input.nextInt();

d= input.nextInt();

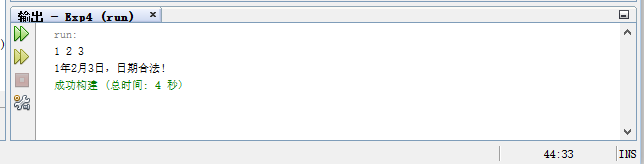
Date2 test = new Date2(y,m,d);

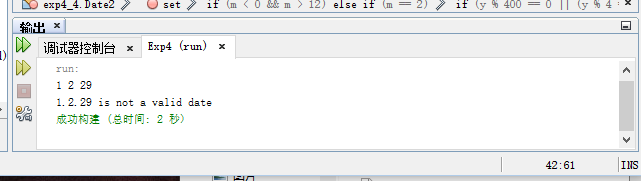
input.close();

}

}

## 4.3 运行结果





## 4.4 调试情况分析

一开始没有重写toString方法，没有得到想要的结果。

# 实验 5 思考题

## 5. 1异常分为哪两类？请简单介绍。

检查型异常(Checked Exception)和运行时异常(Runtime Exception)

Checked Exception在编译期间需要检查,如果该异常被throw,那么在该异常所在的method后必须显示的throws,调用该method的地方也必须捕获该异常,否则编译器会抛出异常.

Runtime Exception就是在运行期间系统出现的异常,该类异常继承于RuntimeException,该类异常在编译时系统不进行检查,如NullPointerExcetpion,NumberFormatException.

## 5.2 多线程的同步处理用于什么情况?

（1）共享变量

当在线程对象（Runnable）中定义了全局变量，run方法会修改该变量时，如果有多个线程同时使用该线程对象，那么就会造成全局变量的值被同时修改，造成错误。

（2）执行步骤

在多个线程运行时，可能需要某些操作合在一起作为“原子操作”(atomic operation)，即在这些操作可以看做是“单线程”的。