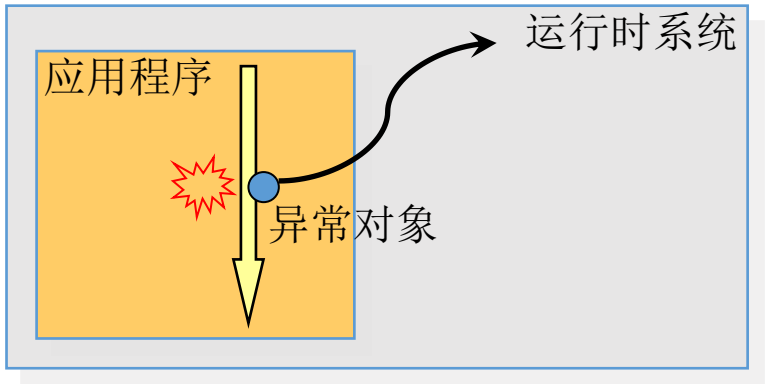
第12章 异常处理和文本I/O

目 录

contents

12.1 异常处理概述

- 异常(Exception)：又称为例外，是程序在运行过程中发生的非正常事件，其发生会影响程序的正常执行。
- 当一个方法中发生错误时，将创建一个对象并将它交给运行时系统，此对象被称为异常对象(exception object)
- 创建异常对象并将它交给运行时系统被称为抛出一个异常(throw an exception) 。



12.1 异常处理概述

- 异常产生的原因
  - Java虚拟机同步检测到一个异常的执行条件，间接抛出异常，例如：
    - 表达式违反了正常的语义，例如整数除零。
    - 通过空引用访问实例变量或方法。
    - 访问数组超界。
    - 资源超出了某些限制，例如使用了过多的内存。
    - …
  - 显式地执行throw语句抛出异常

12.1 异常处理概述

异常的抛出都是由throw语句直接或间接抛出：

1：程序运行时的逻辑错误导致异常间接抛出，例如通过空引用访问实例变量和方法

```
public class A {
    public void m1(){ }
    public static void main(String[] args){
        A o = null;
        /*
         通过空引用访问实例方法，会间接地抛出异常NullPointerException
         */
        o.m1();
    }
}
```

```
D:\jdk1.8.0_231_64bit\bin\java.exe ...
Exception in thread "main" java.lang.NullPointerException
    at hust.cs.javacourse.ch12.ImplicitAndExplicitThrow.main(ImplicitAndExplicitThrow.java:11)

Process finished with exit code 1
```

12.1 异常处理概述

异常的抛出都是由throw语句直接或间接抛出：

2：程序在满足某条件时，用throw语句直接抛出异常，如

```
if(满足某条件){
    throw new Exception("异常描述信息");
```

为什么这里的main函数必须加异常声明而前一个PPT例子不需要？一个是必检异常，一个不是

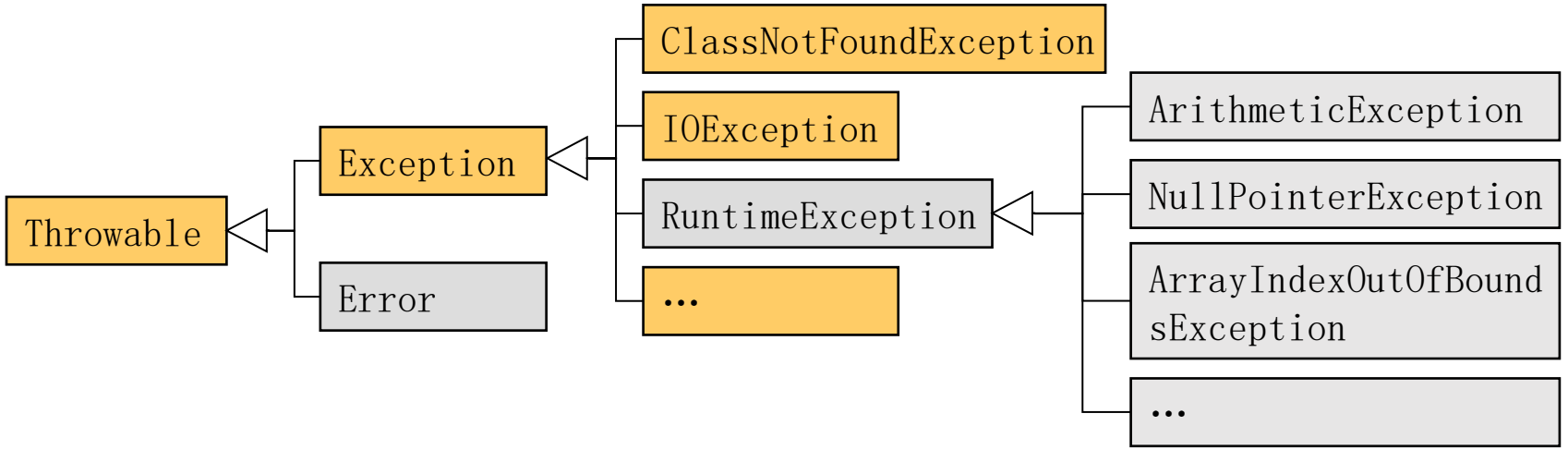
```
public class A {
    //由于main方法里抛出的异常没有被处理,因此在main方法必须加上异常声明throws Exception
    public static void main(String[] args) throws Exception{
        int i = new Scanner(System.in).nextInt();
        if(i > 10){ //假设应用逻辑要求用户输入整数不能大于10
            throw new Exception("Input value is too big"); //显式地用throw抛出异常
        }
    }
}
```

```
D:\jdk1.8.0_231_64bit\bin\java.exe ...
100
Exception in thread "main" java.lang.Exception: Input value is too big
    at hust.cs.javacourse.ch12.ImplicitAndExplicitThrow.main(ImplicitAndExplicitThrow.java:15)

Process finished with exit code 1
```

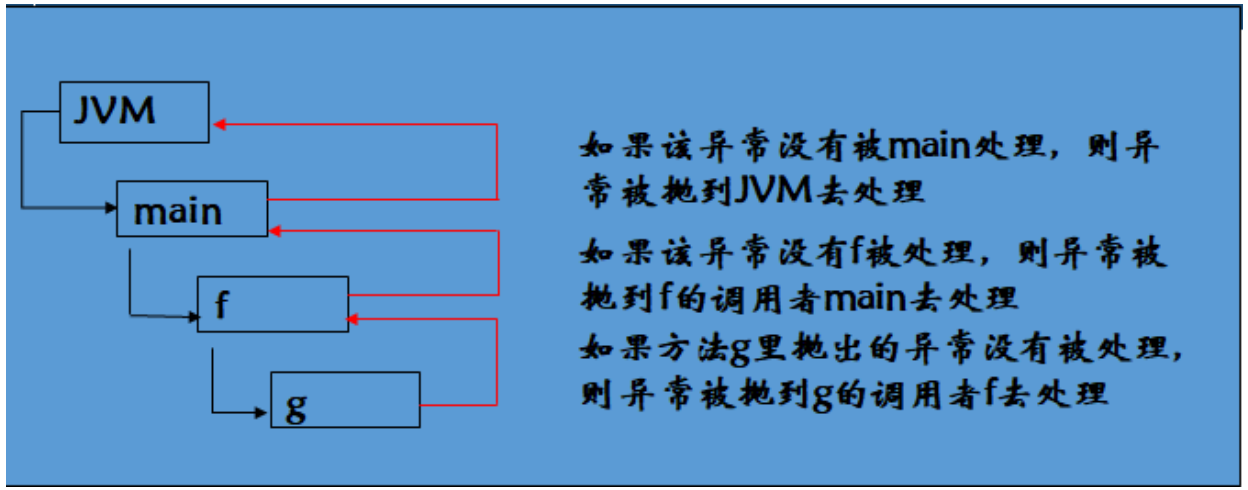
12.1 异常处理概述

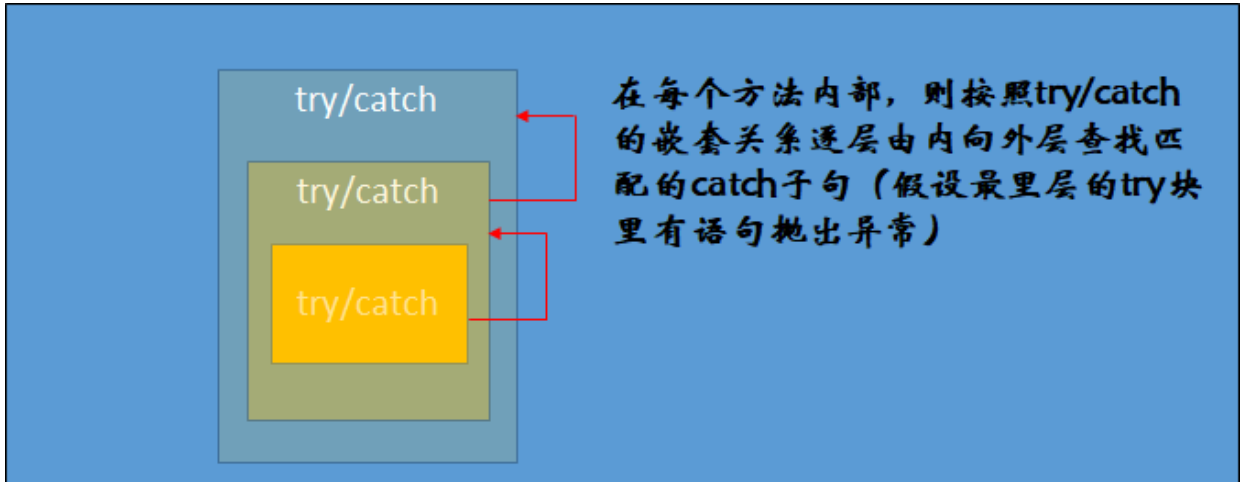
- Java异常都必须继承Throwable的直接或间接子类。用户通过继承自定义异常。
- Java的异常分为二大类：从Exception派生的是程序级错误，可由程序本身处理；从Error派生是系统级错误，程序可不用处理（也基本上处理不了，例如JVM内存空间不够）。
- Exception的子类里，除了RuntimeException这个分支外，其他的都是必检异常（即：要么在函数里用catch子句捕获并处理，要么在所在函数加上异常声明，PPT第5页例子）。RuntimeException的子类是非必检异常（PPT第4页例子）



12.1 异常处理概述

- 运行时异常系统处理异常的过程如下：
- 当发生异常时，运行时系统按与方法调用次序相反的次序搜索调用堆栈，寻找一个包含可处理异常的代码块的方法，这个代码块称为异常处理器(exception handler)，即try/catch语句
- 如果被抛出的异常对象与try/catch块可以处理的类型匹配，运行时系统将异常对象传递给它，这称为捕获异常(catch the exception)
- 如果运行时系统彻底搜索了调用堆栈中的所有方法，但没有找到合适的异常处理器，程序则终止





12.1 异常处理概述

```
public class CallStack {

    public static void methodA() {
        System.out.println("in methodA");
        try {
                methodB();
        }
        catch (Exception e) {
                System.out.println(e);
        }
        System.out.println("end methodA");
    }

    public static void methodB() {
        System.out.println("in methodB");
        methodC();
        System.out.println("end methodB");
    }

}
```

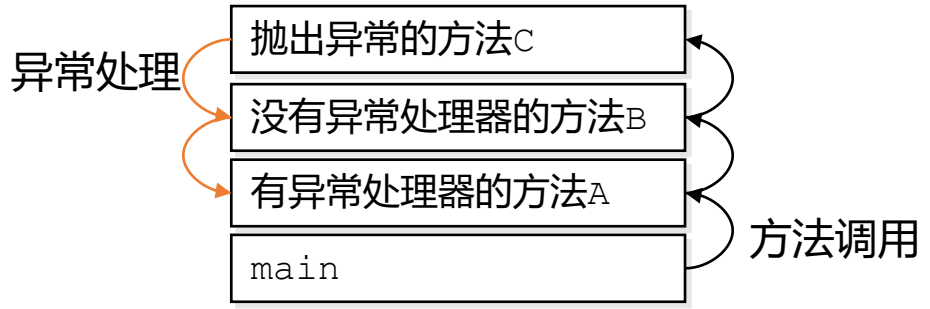
方法A捕获异常并处理，这里只是简单打印出异常对象e

方法B没有处理，顺着调用栈向上抛到方法A

```
public class CallStack {
    public static void methodC() {
        System.out.println("in methodC");
        int i = 10 / 0;
        System.out.println(i);
        System.out.println("end methodC");
    }

    public static void main(String[] args) {
        System.out.println("in methodMain");
        methodA();
        System.out.println("end methodMain");
    }
}
```

方法C抛出异常，但没有处理，顺着调用栈向上抛到方法B



12.2 异常声明、抛出和捕获

- 非必检异常(Unchecked Exception)是运行时异常(RuntimeException)和错误(Error)类及它们的子类，非必检异常在方法里可不捕获异常同时方法头可不声明异常，编译器不会报错。**但该发生的异常还是要发生。**
- 其它的异常称为必检异常(Checked Exception)，编译器确保必检异常被捕获或声明（即要么在方法里捕获异常，要么在方法头声明异常）
  - 捕获：方法可以通过try/catch语句来捕获异常
  - 声明：方法可以在方法头使用throws子句声明可能抛出异常
- 方法可以抛出的异常
  - 方法里调用throw语句直接抛出的任何异常
  - 调用另一个方法**f**时，由方法**f**间接抛出的异常

12.2 异常声明、抛出和捕获

- 异常声明：由方法声明可能抛出的异常
- 如果方法不捕获其中发生的必检异常，那么方法必须声明它可能抛出的这些异常
- 通过throws子句声明方法可能抛出的异常。throws子句由throws关键字和一个以逗号分隔的列表组成，列表列出此方法抛出的所有异常，即一个方法可以声明多个可能抛出的异常
- 例如

```
public void myMethod() throws IOException {
   InputStream in =
            new FileInputStream(new File("C:\\1.txt"));
}
```

12.2 异常声明、抛出和捕获

◆抛出异常

◆抛出异常有二种情况

◆间接抛出：执行语句（如new FileInputStream(new File("C:\\1.txt")); ）或调用方法时由被调用方法抛出的异常

◆显式直接抛出

例如

```
int i = new Scanner(System.in).nextInt();
if(i > 10){ //假设应用逻辑要求用户输入整数不能大于10
    throw new Exception("Input value is too big");
}
```

显式地用throw抛出异常

12.2 异常声明、抛出和捕获

◆捕获异常

- 语法

```
try {
        statements
} catch (ExceptionType1 id1) {
        statements1
} catch (ExceptionType2 id2) {
        statements2
} finally {
        statements3
}
```

当包含catch子句时，finally子句是可选的。
当包含finally子句时，catch子句是可选的。

- 将可能抛出异常的语句放在try块中。当try块中的语句发生异常时，异常由后面的catch块捕获处理。
- 一个try块后面可以有多个catch块。每个catch块可以处理的异常类型由异常类型参数指定。异常参数类型必须是从Throwable派生的类。
- 当try块中的语句抛出异常对象时，运行时系统将调用第一个异常对象类型与参数类型匹配的catch子句。如果被抛出的异常对象可以被合法地赋值给catch子句的参数，那么系统就认为它是匹配的（和方法调用传参一样，子类异常对象匹配父类型异常参数类型）。
- 无论try块中是否发生异常，都会执行finally块中的代码。通常用于关闭文件或释放其它系统资源。
- 处理异常时，也可以抛出新异常，或者处理完异常后继续向上（本方法调用者）抛出异常以让上层调用者知道发生什么事情：链式异常。

12.2 异常声明、抛出和捕获

```
public static String read(String filePath){
    String s = null;
    BufferedReader reader = null; //BufferedReader一次读文本文件一行
    try{
        StringBuffer buf = new StringBuffer();
        reader = new BufferedReader(new InputStreamReader(new FileInputStream(new
                                    File(filePath))));
        while( (s = reader.readLine()) != null){//readLine方法读取到文件末尾返回null
            buf.append(s).append("\n");
        }
        s = buf.toString().trim();
    }
    catch (FileNotFoundException e) { e.printStackTrace();}
    catch (IOException e) { e.printStackTrace()}
    finally {
        if(reader != null) {
            try { reader.close()}
            catch (IOException e) { e.printStackTrace();}
        }
    }
    return s;
}
```

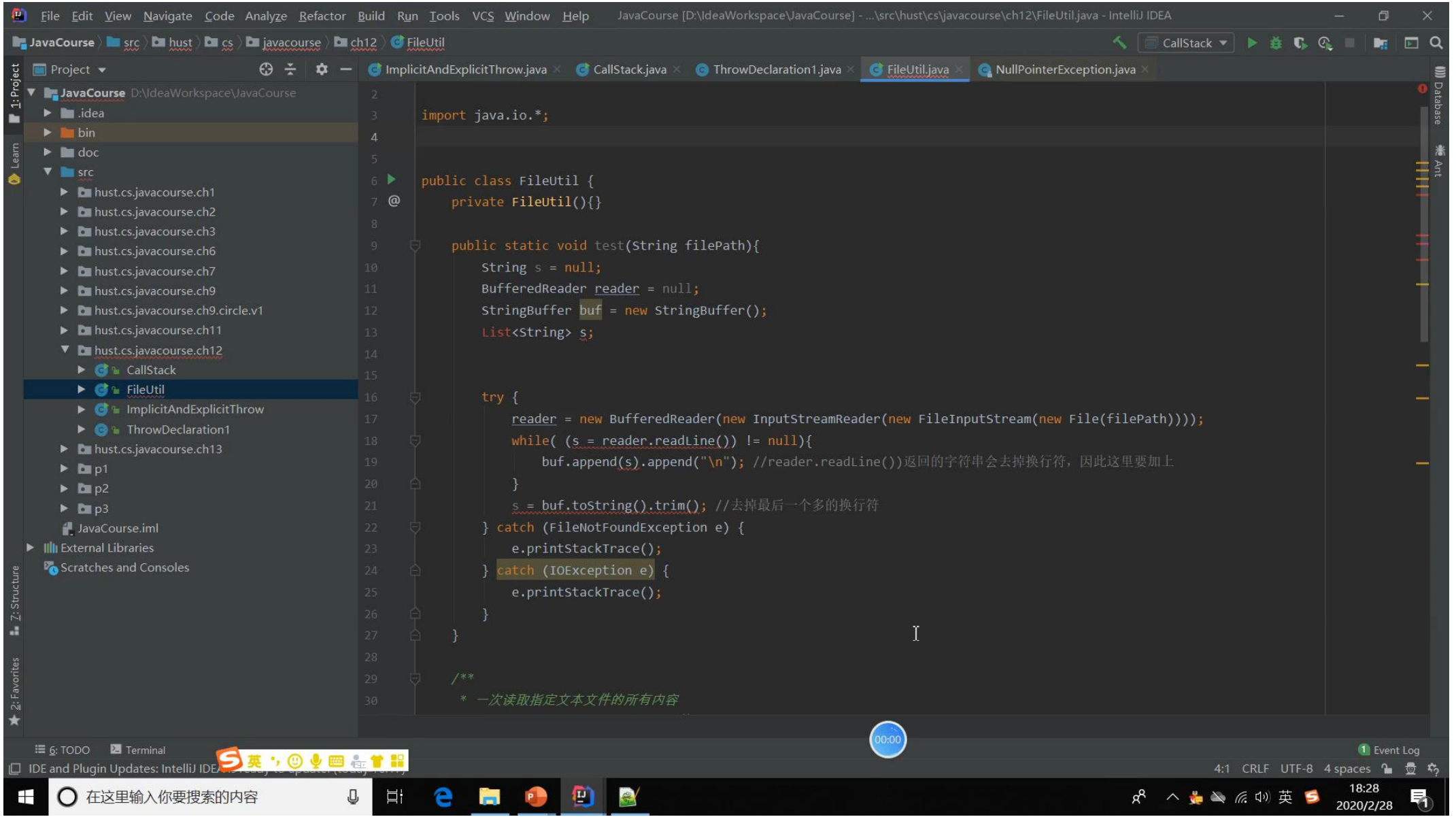
new FileInputStream可能抛出FileNotFoundException。怎么知道的？通过FileInputStream构造函数方法头

readLine方法可能抛出IOException 。怎么知道的？通过readLine的方法头的throws声明

try块里可能抛出的二个异常分别被二个catch块处理

由于reader打开后，执行readLine时可能抛出异常，因此在finally块里关闭流是最合适的地方。注意close也可能抛出异常，因此还得用try/catch处理

方法read内部已经处理了所有可能发生的异常，因此方法首部不需要加throws声明。同时read方法的调用代码不需要try/catch


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
JavaCourse [D:\IdeaWorkspace\JavaCourse] - ...\src\hust\cs\javacourse\ch12\FileUtil.java - IntelliJ IDEA
JavaCourse src hust cs javacourse ch12 FileUtil
CallStack
Project
ImplicitAndExplicitThrow.java
CallStack.java
ThrowDeclaration1.java
FileUtil.java
NullPointerException.java
JavaCourse D:\IdeaWorkspace\JavaCourse
.idea
bin
doc
src
hust.cs.javacourse.ch1
hust.cs.javacourse.ch2
hust.cs.javacourse.ch3
hust.cs.javacourse.ch6
hust.cs.javacourse.ch7
hust.cs.javacourse.ch9
hust.cs.javacourse.ch9.circle.v1
hust.cs.javacourse.ch11
hust.cs.javacourse.ch12
CallStack
FileUtil
ImplicitAndExplicitThrow
ThrowDeclaration1
hust.cs.javacourse.ch13
p1
p2
p3
JavaCourse.iml
External Libraries
Scratches and Consoles
import java.io.*;
public class FileUtil {
private FileUtil(){}
public static void test(String filePath){
String s = null;
BufferedReader reader = null;
StringBuffer buf = new StringBuffer();
List<String> s;
try {
reader = new BufferedReader(new InputStreamReader(new FileInputStream(new File(filePath))));
while((s = reader.readLine()) != null){
buf.append(s).append("\n"); //reader.readLine()返回的字符串会去掉换行符，因此这里要加上
}
s = buf.toString().trim(); //去掉最后一个多的换行符
} catch (FileNotFoundException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}
}
/**
* 一次读取指定文本文件的所有内容
00:00
6: TODO
Terminal
Event Log
IDE and Plugin Updates: IntelliJ IDEA
4:1 CRLF UTF-8 4 spaces
在这里输入你要搜索的内容
18:28
2020/2/28

12.2 异常声明、抛出和捕获

◆方法异常声明与方法内捕获处理异常的关系

```
public class ThrowDeclaration1 {
    //由于m1内部处理了所有异常，因此不用加throws声明
    public void m1(){
        try{
            //执行可能抛出异常的语句
        }
        catch(Throwable e){ //由于Throwable是所有异常的父类，因此这里可以捕获所有异常
            //处理异常
        }
    }

    public void m2(){
        m1(); //由于m1没有异常声明，因此m1的调用者不需要try/catch
    }
}
```

12.2 异常声明、抛出和捕获

◆方法异常声明与方法内捕获处理异常的关系

```
public class ThrowDeclaration2 {
    //m1内部可能抛出的异常没有处理，因此必须加throws声明
    //throws声明就是告诉方法的调用者，调用本方法可能抛出什么异常
    public void m1() throws IOException {

        //执行可能抛出异常IOException的语句,但没有try/catch

    }

    public void m2(){
        //由于m1有异常声明，因此m2调用m1时有第一个选择：1 用try/catch捕获和处理异常
        //这时m2就不用加throws异常声明
        try {
            m1();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- 非必检异常(Unchecked Exception)是运行时异常(RuntimeException)和错误(Error)类及它们的子类，方法可以不捕获同时不声明非必检异常（注意只是编译器不检查了，但如果真的有异常该抛出还是会抛出）
- 10/0这种异常
- 方法如果声明或捕获非必检异常也没问题

```
public void m2() throws IOException {
    //由于m1有异常声明，因此m2调用m1时有第2个选择:
    //2 也在方法头声明异常，方法体里不捕获异常。
    //这时如果有方法m3调用m2， m3也就面临二个选择：声明异常或者在m3里捕获异常
    m1();
}
```

12.3 异常的捕获顺序

◆每个catch根据自己的参数类型捕获相应的类型匹配的异常。

◆由于父类引用参数可接受子类对象，因此，若把Throwable作为第1个catch子句的参数，它将捕获任何类型的异常，导致后续catch没有捕获机会。

◆通常将继承链最底层的异常类型作为第1个catch子句参数，次底层异常类型作为第2个catch子句参数，以此类推。越在前面的catch子句其异常参数类型应该越具体。以便所有catch都有机会捕捉相应异常。

◆无论何时，throw以后的语句都不会执行。

◆无论同层catch子句是否**捕获、处理**本层的异常（即使在catch块里抛出或转发异常），同层的finally总是都会执行。

◆一个catch捕获到异常后，同层其他catch都不会执行，然后执行同层finally。

12.3 异常的捕获顺序-实例

```
import java.lang.System;
import java.lang.ArithmeticException;
public  class  Main {
    static int div(int x, int y) {   //各种Exception都被捕获，函数无须声明异常
        int   r=0;
        try{
                //自己抛出异常对象
                if(y==0) throw  new ArithmeticException( );
                r=x/y; }
        catch(ArithmeticException  ae) {   System.out.print(ae.toString( )); throw ae;   }
        catch(Exception  ae){//捕获各种Exception：若是第1个catch，则后续的catch子句无机会捕获
                System.out.print(ae.toString( ));
        }
        finally{  r=-1; }          //无论是否有异常，r=-1
        return  r;
    }
    public  static  void  main(String[ ] args) {
        try{   div(5, 0);    } //调用div(5, 0)后，div函数的执行轨迹已用红色标出
        catch(Throwable  ae) { //任何异常都被捕获，包括Error类型异常
                System.out.print(ae.toString( ));
        }
    }
}
```

处理完异常后可以继续抛出异常，交给上层调用者继续处理。注意即使这里抛出异常，同层的finally仍会执行

catch子句里抛出异常，这个异常在div方法里没有处理，但是div可以不声明异常？为什么？因为ae是非必检

因此虽然div没有异常声明，在main里调用div也用了try/catch

12.4 自定义异常类

◆自定义异常类必须继承Throwable或其子类。

◆自定义异常类通常继承Exception及其子类，因为Exception是程序可处理的类。

◆如果自定义异常类在父类的基础上增加了成员变量，通常需要覆盖toString函数。

◆自定义异常类通常不必定义clone：捕获和处理异常时通常只是引用异常对象而已。

12.4 自定义异常类-实例

```
import  java.lang.Exception;
public class ValueBeyondRangeException extends Exception{
    int value, range;
    public ValueBeyondRangeException(int v, int r){ value=v; range=r; }
    public String toString( ){
        return value + ” beyonds “ + range;
    }
}
//使用例子
int v = 1000, range = 100;
try{
      if(v > range)
            throw new ValueBeyondRangeException  (v,range) ;
}
catch(ValueBeyondRangeException e){ System.out.println(e.toString( )); }
```

12.5 文本I/O

◆文本：非二进制文件(参见FileInputStream、FileOutputStream)。

◆类库：java.io.File、java.util.Scanner、java.io.PrinterWriter。

◆类File： 对文件和目录的抽象，包括：路径管理，文件读写状态、修改日期获取等。

◆类Scanner：从File或InputStream的读入。可按串、字节、整数、双精度、或整行等不同要求读入。

◆类PrinterWriter ： 输出到File或OutputStream： 可按串、字节、整数、双精度、或整行等不同要求输出。

12.5 文本I/O-实例

```
package  filecopy;
import java.lang.System;
import java.io.File;
import java.io.PrintWriter;
import java.io.IOException;
import java.util.Scanner;
public class Copy {
    public static void main(String[ ] args) {  //参数不含程序名
        if(args.length!=2){
            System.out.println("Usage: Java  Copy  <sourceFile>   <tagetFile>");
            System.exit(1);
        };
        File sF=new File(args[0]);                          //args[0]:源文件路径
        if(!sF.exists( )){
            System.out.println("Source Fiel "+args[0]+ "does not exist!");
            System.exit(2);
        };
```

12.5 文本I/O-实例

```
        File tF=new File(args[1]);        //args[1]:目标文件
        if(tF.exists( )){
            System.out.println("Target File "+args[1]+ "already exist");
            System.exit(3);
        };
        try{
            Scanner input=new Scanner(sF);
            PrintWriter output=new PrintWriter(tF);
            while(input.hasNext( )){
                String s=input.nextLine(); //读取下一行
                output.println(s);                  //打印这一行
            }
            input.close( );
            output.close( );
        }
        catch(IOException ioe){
              System.out.println(ioe.toString( ));
        }
    }
}
```