Chapter 12：异常处理和文本IO

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) 。

异常产生有以下原因：

- 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：
  - 表达式违反了正常语义，例如整数除0
  - 通过空引用访问实例变量或方法
  - 访问数组越界

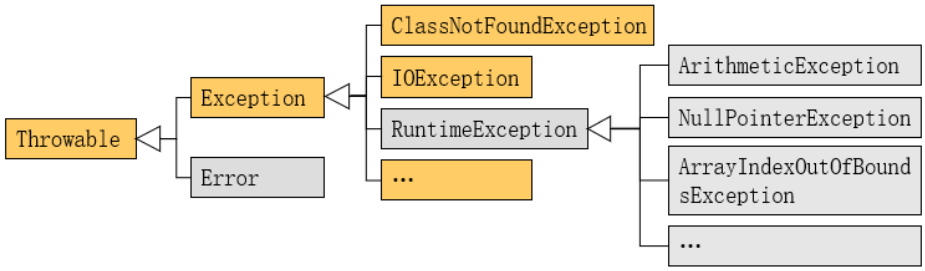
```
public class A {
    public void m1(){ }
    public static void main(String[] args){
       A o = null;
       /*
    通过空引用访问实例方法，会间接地抛出异常NullPointerException
    */
       o.m1();
   }
}
```

- 通过执行throw语句**显示抛出**异常
  - 程序在某个条件下，用throw直接抛出异常

```
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
       }
   }
}
```

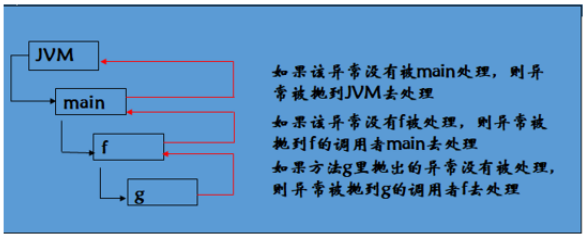
异常特点：

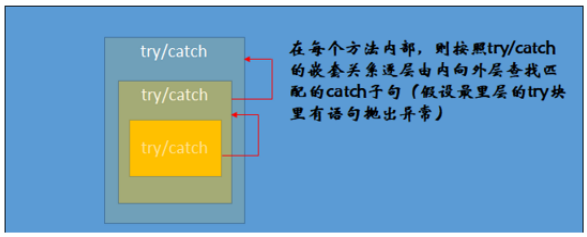
- 异常**都必须继承**Throwable的直接或间接子类。用户通过继承自定义异常
- 异常分为两类：从Exception派生的程序级错误，可由程序本身处理；从Error派生的系统及错误，程序可不用处理
- Exception的子类中又分为**必检异常**和**非必检异常**，属于RuntimeException这个分支的异常都是非必检异常。除了RuntimeException这个分支外，其他都是必检异常，即要么在函数中用catch捕获，要么在函数上加上异常声明。



运行时异常处理过程：

- 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






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

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

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

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

```
// 输出:
// in Main
// in A
// in B
// in C
// java.lang.ArithmeticException: / by zero
// end A
// end Main
```

12.2 异常声明、抛出和捕获

12.2.1 异常种类

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。

必检异常(Checked Exception)，编译器确保必检异常被捕获（通过try/catch）或声明（通过在方法头上是同throws子句声明可能抛出异常），即要不在方法里捕获异常，要不在方法头声明异常。

12.2.2 异常声明

1. 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

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

2. 抛出异常

- 情况一：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
- 情况二：显示直接抛出

```
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");
}
```

12.2.3 异常捕获

语法：

```
    try {
        statements
    } catch (ExceptionType1 id1) {
        statements1
    } catch (ExceptionType2 id2) {
        statements2
    } finally {
        statements3
    }
// 包含catch子句，finally子句是非必要的
// 包含finally子句，catch子句是非必要的
```

- 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。

```
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

12.3 异常捕获的顺序

- 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。

```
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。

```
import  java.lang.Exception;
public class ValueBeyondRangeException extends Exception{
    int value, range;
    public ValueBeyondRangeException(int v, int r){ value=v; range=r; }
    public 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IO

- 文本：非二进制文件(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。

```
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);
        };
        File tF=new File(args[1]);      //args[1]:目标文件
        if(tF.exists( )){
            System.out.println("Target File "+args[0]+ "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( ));
        }
    }
}
```