内容提要

- 异常处理简介
- 错误类和异常类
- 异常处理方法
- 生成异常对象
- 声明自己的异常类

异常处理简介

- 在进行程序设计时，错误的产生是不可避免的
  - 语法错误：编译器编译时就能检查出来
  - 运行时错误：如除零溢出、数组越界、内存溢出等
- 所谓（运行时）错误，是在程序运行过程中发生的异常事件，这些事件的发生将阻止程序的正常运行
- 编程程序时，应事先当充分考虑到各种可能出现的异常，并作出相应的处理，以保证程序的正确运行
- 如何处理错误？把错误交给谁去处理？程序又该如何从错误中恢复？
- 为了加强程序的鲁棒性，Java语言提供了一种面向对象的运行错误处理机制

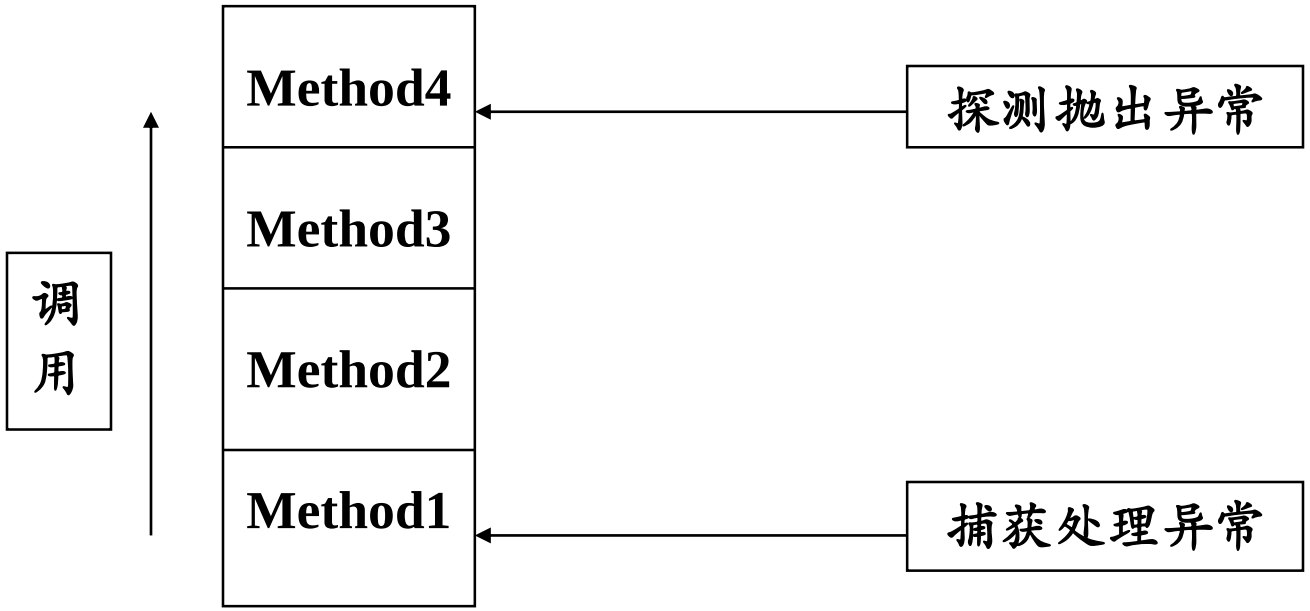
异常处理简介

- 异常的基本概念
  - 广义的异常：一切运行时错误
  - 狭义的异常（也叫例外）：特殊的运行时错误
  - Java中声明了很多异常类，每个异常类都代表了一种运行错误，类中包含了该运行错误的信息及其获取方法
  - 每当Java程序运行过程中发生一个可识别的运行错误时，即该错误有一个异常类与之相对应时，系统都会产生一个相应的该异常类的对象，即产生一个异常

异常处理简介

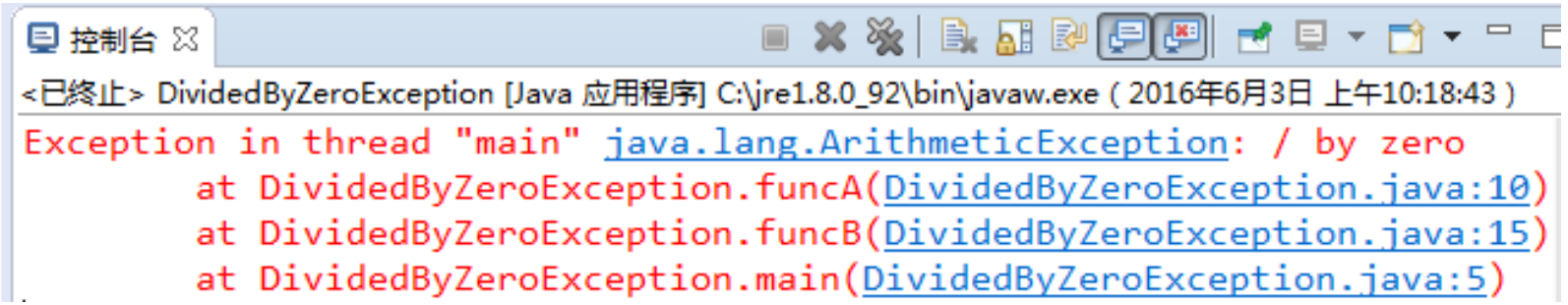
- Java处理异常的机制
  - 抛出(throw)异常
    - 在方法的运行过程中，如果发生了异常，则该方法生成一个代表该异常的对象并把它交给运行时系统，运行时系统便寻找相应的代码来处理这一异常
  - 捕获(catch)异常
    - 运行时系统在方法的调用栈中查找，从生成异常的方法开始进行回溯，直到找到包含相应异常处理的方法为止

异常处理简介



异常处理简介

```
public class DividedByZeroException {
   public static void main(String[] args) {
          int z=0;
          z=funcB(0);  //请尝试在此添加try...catch包围块
          System.out.println(z);
   }
   static int funcA(int d){
          int x=0;
          x=100/d;
          return x;
   }
   static int funcB(int c){
          int y=0;
          y=5+funcA(c);
          return y;
   }
}
```



异常处理简介

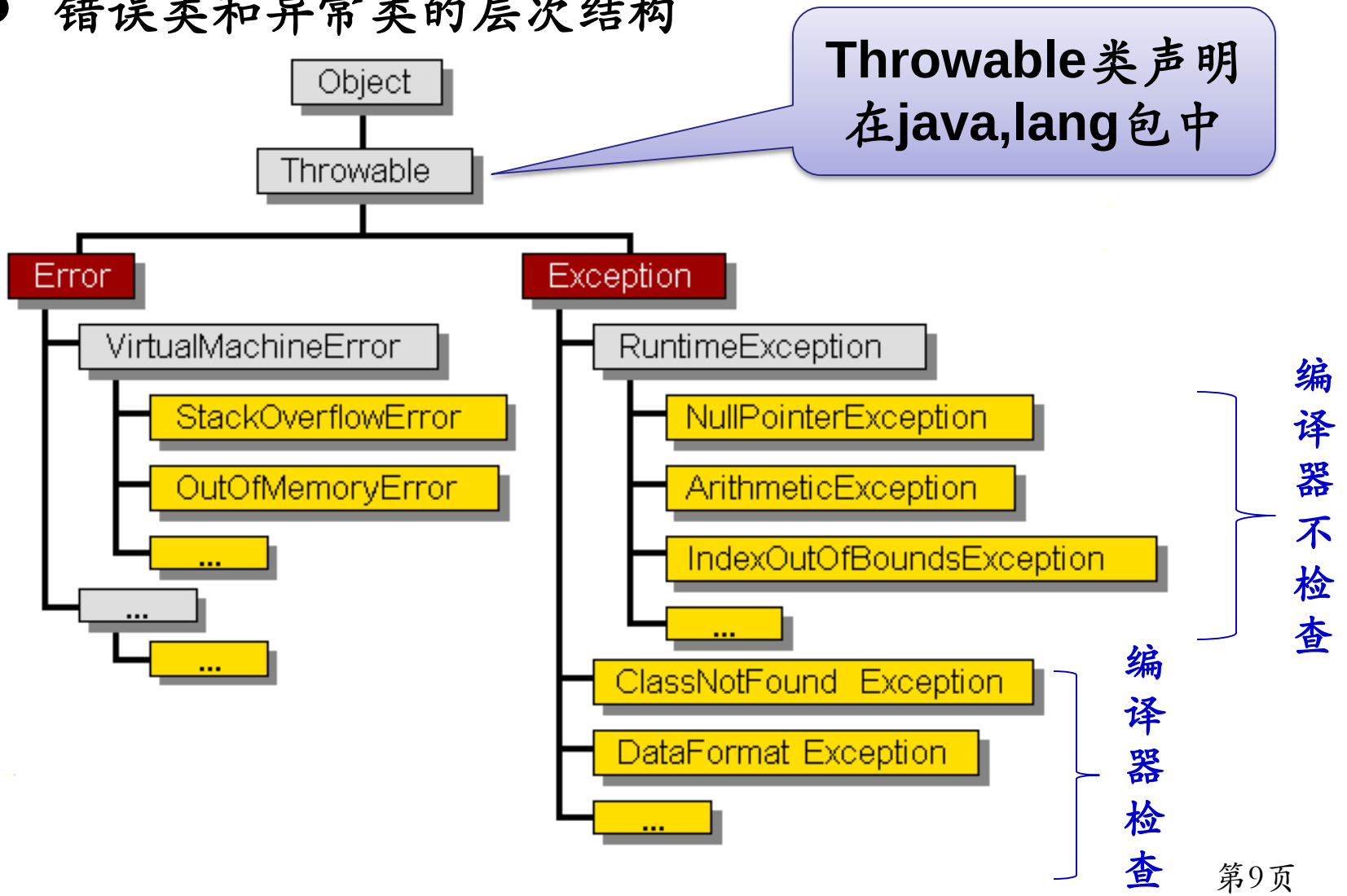
- Java运行时错误的分类（根据错误的严重程度不同）
  - 错误
    - 致命性的，用户程序无法处理
    - Error类是所有错误类的父类
  - 异常
    - 非致命性的，可编制程序捕获和处理
    - Exception类是所有异常类的父类

异常处理简介

- 异常又可分为两大类
  - 非检查型异常
    - 不能期望程序捕获的异常(例如数组越界，除零等)
    - 继承自RuntimeException类
    - 在方法中不需要声明，编译器也不进行检查
  - 检查型异常
    - 其他类型的异常
    - 如果被调用的方法抛出一个类型为E的检查型异常，那么调用者必须捕获E或者也声明抛出E（或者E的一个父类），对此编译器要进行检查

异常处理简介

- 错误类和异常类的层次结构



异常处理简介

- Java预定义的一些常见异常
  - **ArithmeticException**：整数除法中除数为0
  - **NullPointerException**：访问的对象还没有实例化
  - **NegativeArraySizeException**
    创建数组时元素个数是负数
  - **ArrayIndexOutOfBoundsException**
    访问数组元素时，数组下标越界
  - **ArrayStoreException**
    程序试图向数组中存取错误类型的数据
  - **FileNotFoundException**
    试图存取一个并不存在的文件
  - **IOException**：通常的I/O错误检查型异常

编译器不检查

编译器检查

异常处理简介

【例】测试系统定义的运行异常—数组越界出现的异常

```
public class HelloWorld {
   public static void main (String args[ ]) {
         int i = 0;
         String[ ]  greetings = {"Hello world!", "No, I mean it!",
                                 "HELLO WORLD!!"};
         while (i < 4) {
               System.out.println (greetings[i]);
               i++;
         }

         System.out.println ("Hi");
   }
}
```

异常处理简介

– 运行结果

– 说明

- 访问数组下标越界，导致ArrayIndexOutOfBoundsException异常
- 该异常是系统定义好的类，对应系统可识别的错误，所以Java虚拟机会自动中止程序的执行流程，并新建一个该异常类的对象，即抛出数组出界异常

异常的处理

- 对于检查型异常，Java强迫程序必须进行处理
- 处理方法有两种
  - 声明抛出异常
    - 不在当前方法内处理异常，而是把异常抛出到调用方法中
  - 捕获异常
    - 使用try{}catch(){}finally{}块，捕获到所发生的异常，并进行相应的处理
- 对于非检查型异常，也可以这么做，但没有强制要求

异常的处理：声明抛出异常

- 如果程序员不想在当前方法内处理异常，可以使用throws子句声明将异常抛出到调用方法中
- 声明语法形如：

  void 方法名(形参表) **throws** 异常类型1, 异常类型2, …

  {方法体}
- 如果所有的方法都选择了抛出此异常，最后 JVM将捕获它，输出相关的错误信息，并终止程序的运行。在异常被抛出的过程中，任何方法都可以捕获它并进行相应的处理

异常的处理：声明抛出异常

- 一个例子：抛出FileNotFoundException异常

（1）在文件输入流的构造方法中声明了抛出该异常

```
public java.io.FileInputStream.FileInputStream(String name) throws FileNotFoundException
```

（2）在自定义的方法中我们也声明了要抛出该异常

```
static void func(String fn) throws FileNotFoundException {
   //创建文件输入流
   FileInputStream fis=new FileInputStream(fn);
   //  do something
}
public static void main(String[] args)  {
        func("readme.txt");
}
```

如果没有声明抛出该异常，编译器要求该方法中捕获并处理它

编译错误，要求main方法中捕获并处理上述异常，或或main()也声明抛出它

异常的处理：捕获异常

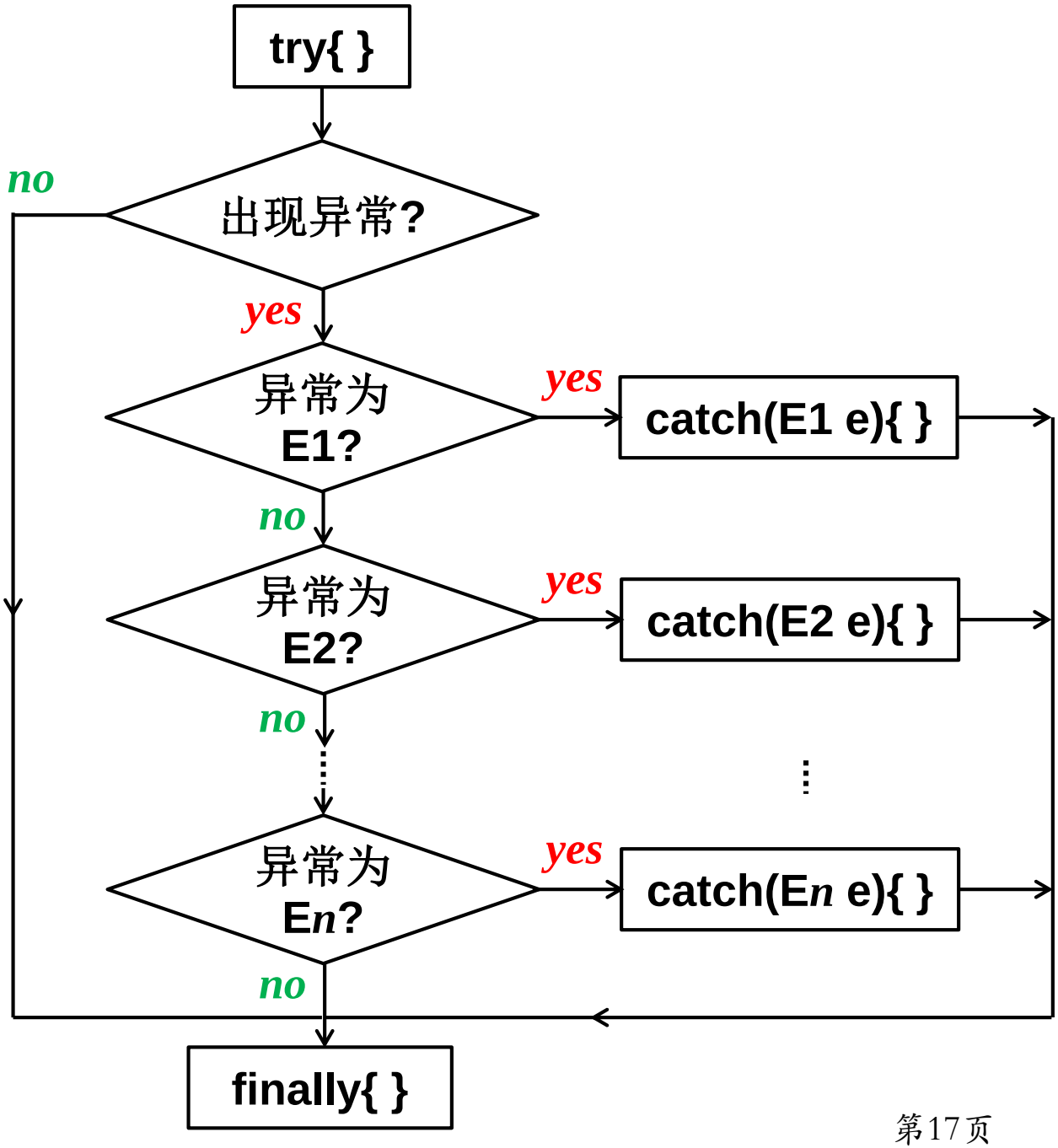
- try…catch…finally语句能捕获一个或多个异常并处理

```
try {
    可能产生异常的代码块;
} catch (异常类型1 e) {
    异常处理代码块;
} catch (异常类型2 e) {
    异常处理代码块;
} finally { // 可选的语句
    不论在try代码段是否产生异常，都要执行的代码;
}
```

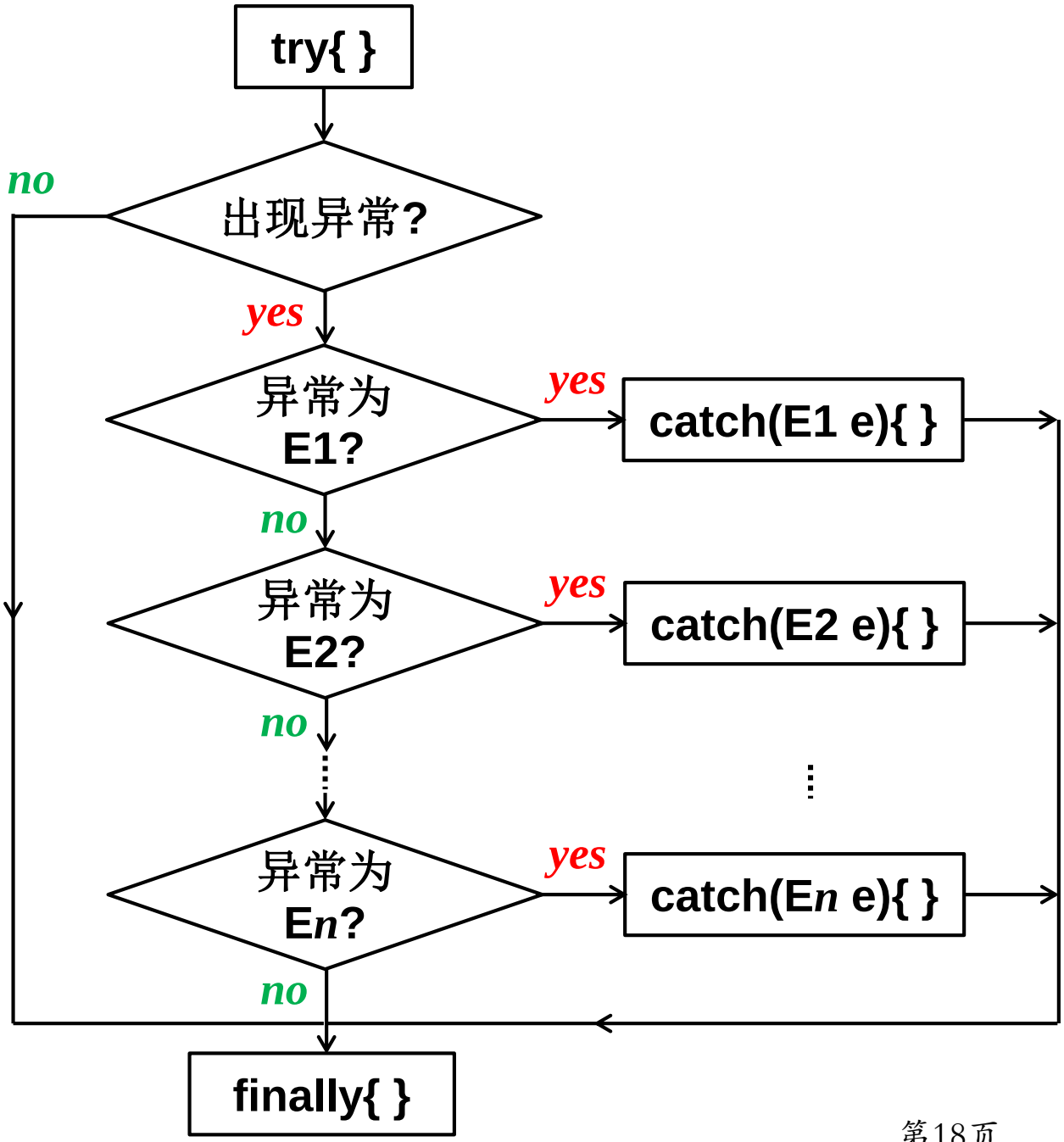
通常用到两个方法：
- **getMessage()** – 返回一个字符串对发生的异常进行描述。
- **printStackTrace()** – 给出方法的调用序列，一直到异常的产生位置

通常在这里释放内存以外的其他资源，例如关闭文件

```
try {
   code;
} catch (E1 e) {
   code;
} catch (E2 e) {
   code;
}   …
 catch (En e) {
   code;
} finally {
   code;
}
```



- 若异常类型E1、E2 …E*n*有继承关系，则子类在前，父类在后；
- catch(Exception e)可以捕获所有异常。如果添加，则它应当在最后；
- 不关心切确的异常类型时，可以只用一个catch(Exception e)来处理；



异常的处理：捕获异常

【例】读入两个整数，第一个数除以第二个数，之后输出

```
public class ExceptionTester1 {
    public static void main(String args[]) {
        System.out.print("Enter the first number:");
        int x = Keyboard.getInteger();
        System.out.print("Enter the second number:");
        int y = Keyboard.getInteger();
        int result = x / y;
        System.out.println(result);
    }
}
```

异常的处理：捕获异常

- Keyboard类：

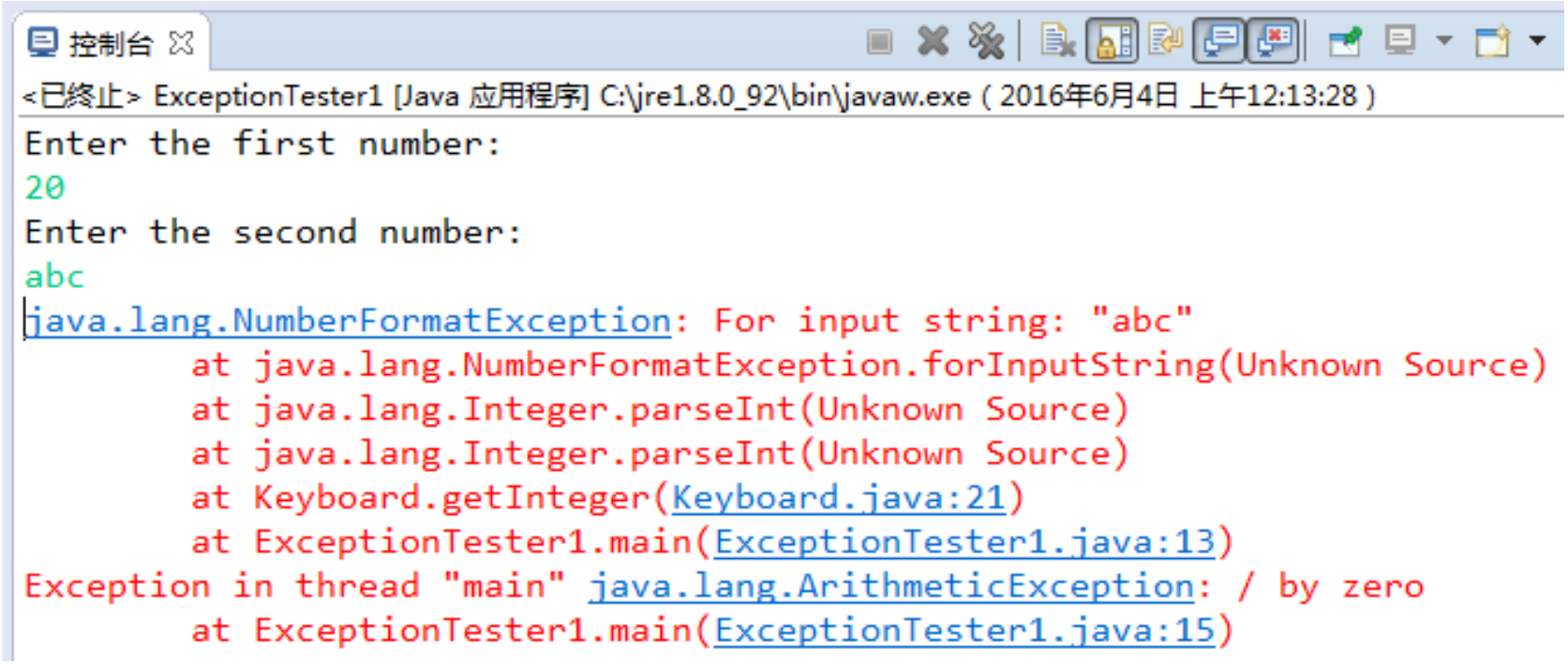
```
import java.io.*;
public class Keyboard{
    static InputStreamReader isr=new
    InputStreamReader(System.in); // 将键盘输入流包装字符输入流
    static BufferedReader br = new BufferedReader(isr); //对字符输
    入流包装，用于从字符输入流中读取文本
    public static int getInteger() {
            try{return (Integer.valueOf(br.readLine().trim()));
            }catch (Exception e) { e.printStackTrace(); return 0;  }
    }
    public static String getString() {
            try{return (br.readLine());
            }catch (IOException e) { return "0";}
    }
}
```

Integer类的valueOf方法：
public static Integer valueOf(String s)
throws NumberFormatException

InputStreamReader 类的readLine方法：
public String readLine() throws IOException

异常的处理：捕获异常

- 运行结果



异常的处理：捕获异常

【例】捕获NumberFormatException类型的异常

```
public class ExceptionTester2  {
    public static void main(String args[]) {
        int x=0, y=0;
        try {
            System.out.print("Enter the first number:");
            x = Integer.valueOf(Keyboard.getString());
            __.print("Enter the second number:");
            y = Integer.valueOf(Keyboard.getString());
        } catch (NumberFormatException e) {
            __.println("Those were not proper integers!");
            System.exit(-1);
        }
        int result = x / y;
        __.println(result);
    }
}
```

```
Enter the first number: 3o06
Those were not proper integers!
```

异常的处理：捕获异常

【例】捕获被零除的异常（ArithmeticException类型的异常）

```
try {  System.out.print("Enter the first number:");
    x = Integer.valueOf(Keyboard.getString());
    __.print("Enter the second number:");
    y = Integer.valueOf(Keyboard.getString());
    __.println(x + " / " + y + "=");
    result = x / y;
} catch (NumberFormatException e) {
    __.println("Those were not proper integers!");
    System.exit(-1);
} catch (ArithmeticException e) {
    __.println("Second number is 0, cannot do division!");
       System.exit(-1);
}
__.println(result);
```

```
Enter the first number:  50
Enter the second number: 0
50 / 0=
Second number is 0, cannot do division!
```

异常的处理：捕获异常

【例】对程序进行改进：重复提示输入，直到输入合法的数据。为了避免代码重复，可将数据存入数组

```
public static void main(String args[]) {
   int     result;
   int []  number= new int[2];
   boolean   valid;
   for (int i=0; i<2; i++) {
      valid = false;
      while (!valid) {
        try { System.out.println("Enter  number "+(i+1));
          number[i]=Integer.valueOf(Keyboard.getString());
          valid = true;
        }catch (NumberFormatException e) {
          __.println("Invalid integer entered. Please try again.");
        }
      }
   } // end of for()
```

异常的处理：捕获异常

```
    try {
        result = number[0] / number[1];
        __.print(number[0] + " / " + number[1] + "=" + result);
    } catch (ArithmeticException e) {
        __.println("Second number is 0, cannot do division!");
    }
}
```

控制台

<已终止> ExceptionTester4 [Java 应用程序] C:\jre1.8.0_92\bin\javaw.exe

```
Enter  number 1
3o8
Invalid integer entered. Please tryagain.
Enter  number 1
308
Enter  number 2
abc
Invalid integer entered. Please tryagain.
Enter  number 2
4
308 / 4=77
```

生成异常对象并抛出

- 在捕获异常之前，必须生成异常对象并抛出它。异常对象生成并抛出有三种方式
  - 由Java虚拟机生成并抛出
  - 由Java类库中的某些类生成并抛出
  - 在程序中人为地生成自己的异常对象并抛出
- 生成异常对象都是通过throw语句实现，生成的异常对象必须是Throwable或其子类的实例
  - throw new 异常类名();
  - 例如： throw new ArithmeticException();

    或:ArithmeticException e = new ArithmeticException();

    throw e;
- 注意：如果抛出的是检查型异常，则应当在方法头用throws语句声明抛出异常的类型（因为编译器要检查）

生成异常对象并抛出

【例】方法的参数非法，抛出IllegalArgumentException异常

```
public class ThrowExistingException {
   public static void main(String[] args) {
         System.out.println("sqrt(5)="+squareRoot(5));
        try{  __.print("sqrt(-5)=");   __.println(squareRoot(-5));
         }catch(Exception e){e.printStackTrace();}
   }
   static double squareRoot(double d){
         if(d<0)  throw new IllegalArgumentException("不能
                                                  对负数开平方");
         return Math.sqrt(d);
   }
```

```
sqrt(5)=2.23606797749979
sqrt(-5)=java.lang.IllegalArgumentException: 不能对负数开平方
        at ThrowExistingException.squareRoot(ThrowExistingException.java:27)
        at ThrowExistingException.main(ThrowExistingException.java:21)
```

声明自己的异常类

- 除使用系统预定义的异常类外，用户还可声明自己的异常类
- 自定义的所有异常类都必须是Exception的子类
- 一般的声明方法如下

```
public class 新异常类名 extends 父异常类{
public 新异常类名() { //构造方法
    super("Some string explaining the exception");
}
```

- 含有throw语句的方法，如果在方法中未进行异常处理，则应当在该方法头用throws语句声明所有可能抛出的异常，以便通知所有此方法的调用者进行异常捕获和处理。

生成异常对象并抛出

【例】声明当除数为零时抛出的异常类DivideByZeroException

```
public class DivideByZeroException extends
    ArithmeticException {
    public DivideByZeroException() {
        super("Attempted to divide by zero");
    }
}
```

生成异常对象并抛出

```
public class DivideByZeroExceptionTester {
   public static void main(String[] args) {
         int x=10, y=5;
         __.println(x + " / " + y + "=" + quotient(x, y));
         y=0;
         __.println(x + " / " + y + "=" + quotient(x, y));
   }

    // numerator(分子)为被除数，denominator(分母)为除数，
   quotient 为商
   static int quotient(int numerator, int denominator) throws
   DivideByZeroException {
         if (denominator == 0)
               throw new DivideByZeroException();
         return (numerator / denominator);
   }
}
```

习 题

1. P147习题6.1~6.3大题
2. 上机：
   a) 编写一个类的静态方法Sum用于计算整数数组中元素之和。访问数组可能产生数组下标越界异常。
      ① 请在Sum方法中对该异常进行捕获并处理，然后main方法中调用Sum方法测试
      ② 或者在main方法中调用Sum，并对Sum方法可能产生的异常进行捕获并处理

习 题

b) 编写一个类的静态方法getFileSize用于获取指定文件的大小，用java.io.FileInputStream类实现，此时可能产生FileNotFoundException和IOException两种检查型异常，请在getFileSize方法中对此进行恰当的处理，并在main方法中测试。

【提示：关于FileInputStream类】

- **FileInputStream(String name) throws FileNotFoundException** 构造函数
- **public int read(byte[] b) throws IOException** 从此输入流中将最多 b.length 个字节的数据读入一个缓冲区(byte 数组)b中。返回读入缓冲区的字节总数，如果因为已经到达文件末尾而没有更多的数据，则返回 -1。
- **public void close() throws IOException** 关闭此文件输入流

谢谢大家！