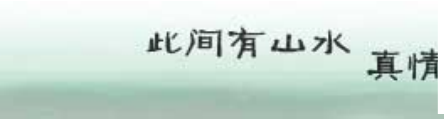
面向对象程序设计

第 10 讲 OOP 开发编程—异常处理

刘进

2230652597@qq.com

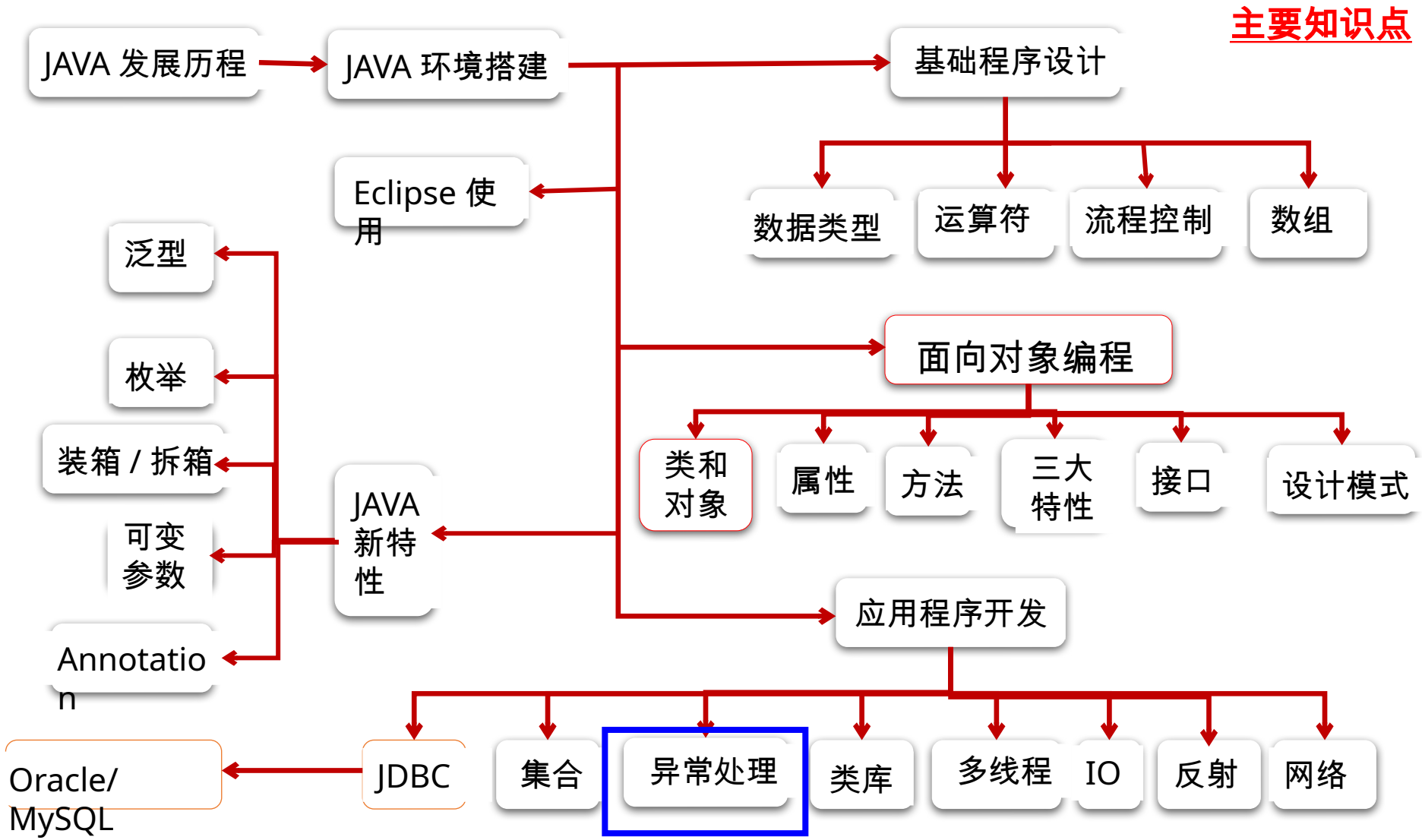


OOP 教辅 2025 秋季 QQ 群：

305915615

群名称：OOP教辅2022秋季
群　号：305915615

Java 基础知识图解


主要知识点
JAVA 发展历程
JAVA 环境搭建
基础程序设计
Eclipse 使用
数据类型
运算符
流程控制
数组
泛型
枚举
装箱 / 拆箱
可变参数
Annotation
JAVA 新特性
面向对象编程
类和对象
属性
方法
三大特性
接口
设计模式
应用程序开发
Oracle/ MySQL
JDBC
集合
异常处理
类库
多线程
IO
反射
网络

Java 异常（ 1 ）

异常

异常是程序中的一些错误，但并不是所有的错误都是异常，并且错误有时候是可以避免的。

比如说代码少了一个分号，那么运行出来结果是提示是错误 java.lang.Error ；如果你用 System.out.println(11/0) ，是因为用 0 做了除数，会抛出 java.lang.ArithmeticException 的异常。

异常发生的原因有很多，通常包含以下几大类：

- 用户输入了非法数据。
- 要打开的文件不存在。
- 网络通信时连接中断，或者 JVM 内存溢出。

这些异常有的是因为**用户错误**引起，有的是**程序错误引**起的，还有其它一些是因为**物理 / 系统 错误**引起的。

Java 异常（ 2 ）

- 异常：在 Java 语言中，将程序执行中发生的不正常情况称为“异常”。（开发过程中的语法错误和逻辑错误不是异常）
- Java 程序在执行过程中所发生的异常事件可分为两类：

➢ **Error:** Java 虚拟机无法解决的严重问题。如： JVM 系统内部错误、资源耗尽等严重情况。一般不编写针对性的代码进行处理。

➢ **Exception:** 其它因编程错误或偶然的外在因素导致的一般性问题，可以使用针对性的代码进行处理。例如：

✓空指针访问

✓试图读取不存在的文件

Error 是指系统崩溃出现的错误，主要是有 OutOfMemoryError。Error 用来指示运行时环境发生的错误。例如，JVM 内存溢出。一般地，程序不会从错误中恢复。Java 程序通常不捕获错误。错误一般发生在严重故障时，它们在 Java 程序处理的范畴之外。

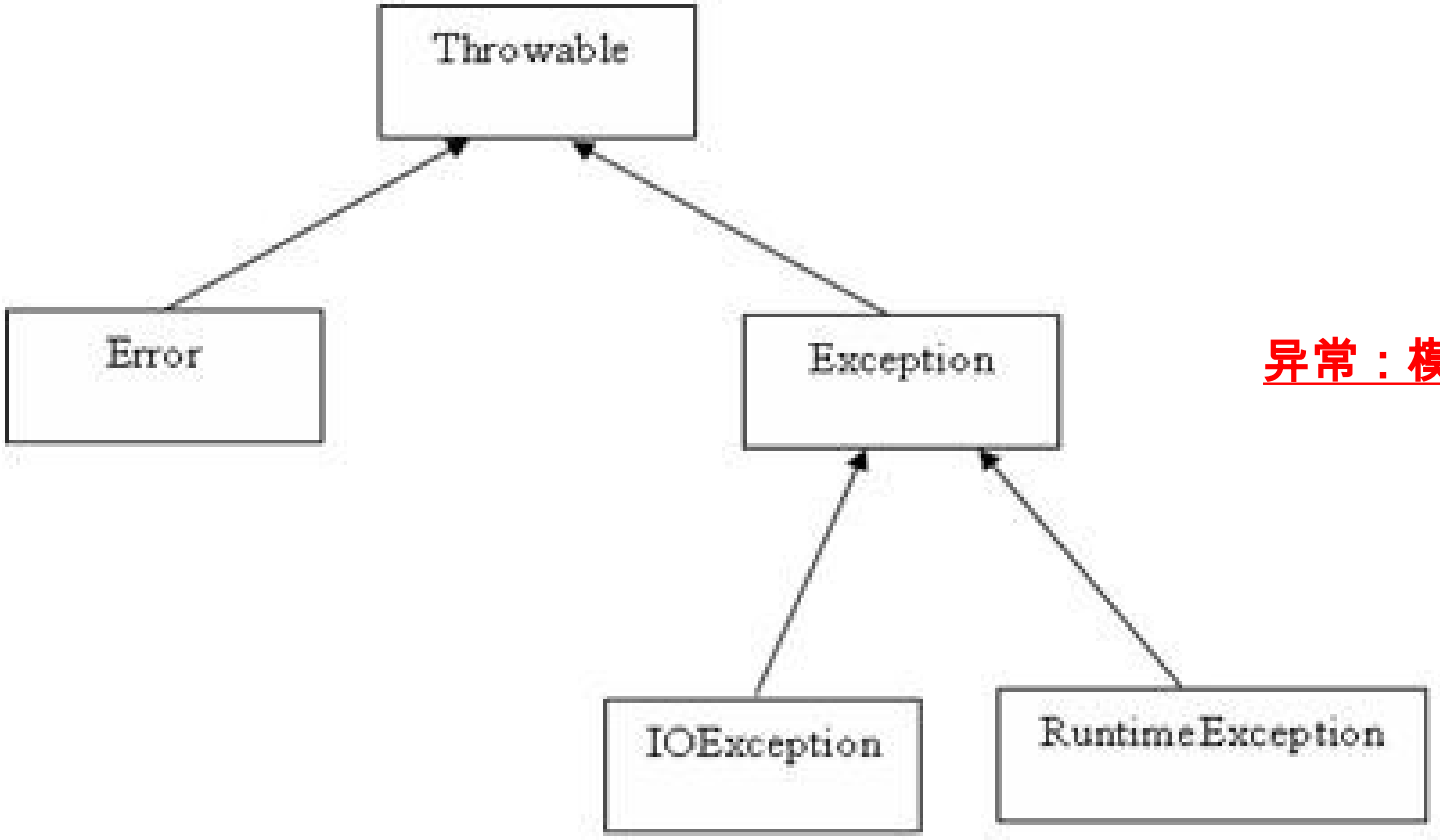
Exception 是指程序中的异常。 区别

Error 和 Exception 的最大区别是程序本身是否能处理。

Error 程序是处理不了的，是系统出现的错误。而 Exception 是程序内部可以解决的问题。

所有的异常类是从 **java.lang.Exception** 类继承的子类。

Exception 类是 Throwable 类的子类。除了 Exception 类外，Throwable 还有一个子类 Error



异常：模型、思想

异常类有两个主要子类： IOException 类和 RuntimeException 类。

常见异常

- java.io.IOExeption-checked
  - FileNotFoundException
  - EOFException
- java.lang.ClassNotFoundException
- java.lang.InterruptedException
- java.io.FileNotFoundException
- java.sql.SQLException
- java.lang.RuntimeException -unchecked
  - ClassCastException
  - ArrayIndexOutOfBoundsException
  - NullPointerException
  - ArithmeticException
  - 。。。

checked 编译时检查

和

unchecked 编译时不检查

Java 异常

Checked 、 Unchecked 、 Error

需要了解三种类型的异常：

- **Checked 检查性异常：**最具代表的检查性异常是用户错误或问题引起的异常，这是程序员无法预见的。例如要打开一个不存在文件时，一个异常就发生了，这些**异常在编译时不能被忽略**。
- **Unchecked 运行时异常：** 运行时异常是可能被程序员避免的异常。与检查性 cheched 异常相反，运行时**异常可以在编译时可被忽略**。
- **Error 错误：** 错误不是异常，而是脱离程序员控制的问题。错误在代码中通常被忽略。例如，当栈溢出时，一个错误就发生了，它们在编译时也检查不到的。

Java 异常（ 4 ）Checked 、 Unchecked 、 Error

1. 运行时异常 -unchecked

- 是指编译器不要求强制处置的异常。一般是指编程时的逻辑错误，是程序员应该积极避免其出现的异常。**java.lang.RuntimeException** 类及它的子类都是运行时异常。
- 对于这类异常，可以不作处理，因为这类异常很普遍，若全处理可能会对程序的可读性和运行效率产生影响。

2. 编译时异常 -checked

- 是指编译器要求必须处置的异常。即程序在运行时由于外界因素造成的一般性异常。编译器要求 java 程序必须捕获或声明所有编译时异常。
- 对于这类异常，如果程序不处理，可能会带来意想不到的结果。

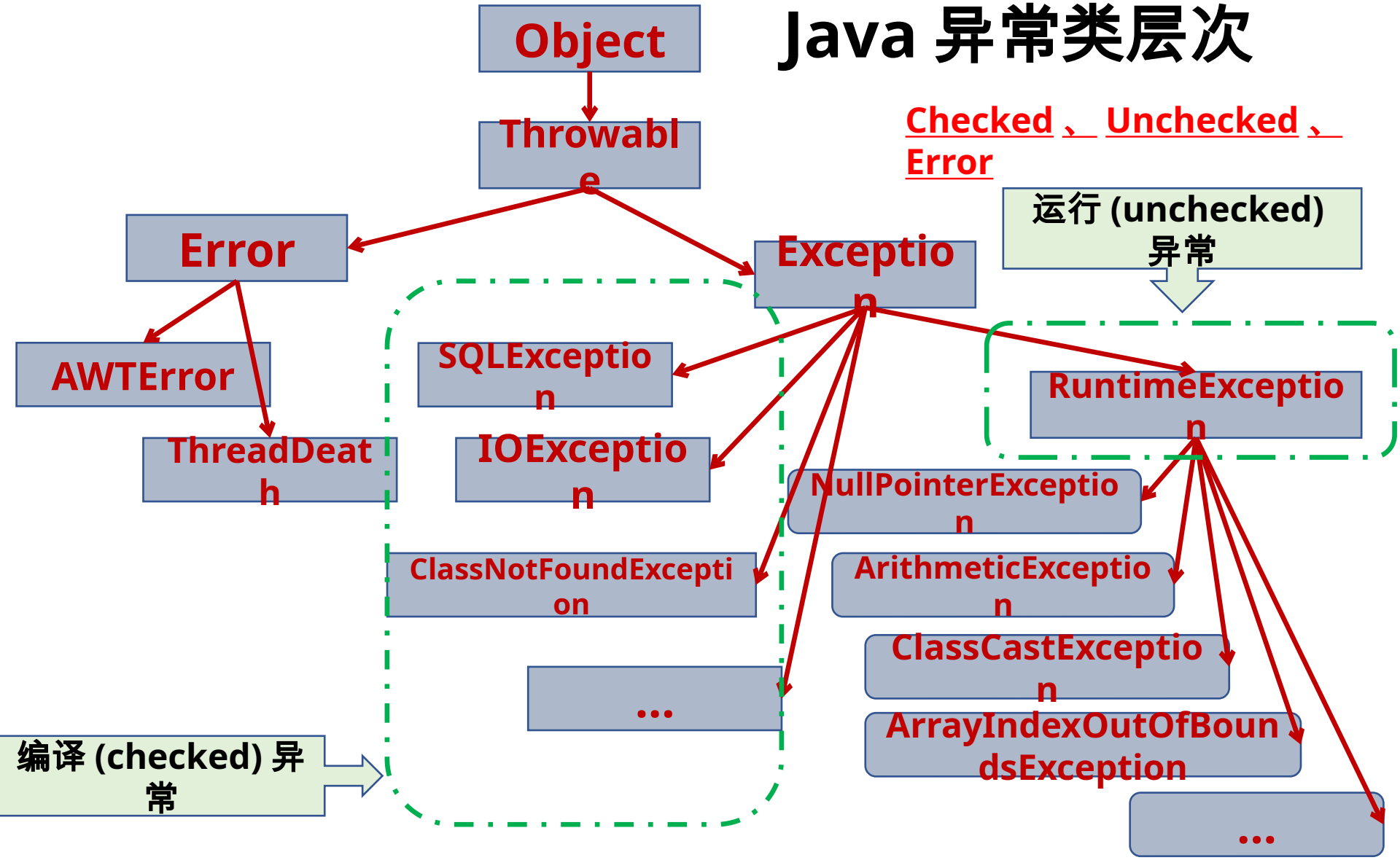
Java 异常（ 3 ）

Checked 、 Unchecked 、 Error

- 对于这些错误，一般有两种**解决方法**：一是遇到错误就终止程序的运行。另一种方法是由程序员在编写程序时，就考虑到**错误的检测、错误消息的提示，以及错误的处理。（ Checked ）**

- 捕获错误最理想的是在**编译期间 -checked** ，但有的错误只有在**运行时 -unchecked** 才会发生。比如：除数为 0 ，数组下标越界等
  - 分类：编译时异常和运行时异常

Java 异常类层次


Object
Throwable
Error
Exception
AWTError
ThreadDeath
SQLException
IOException
ClassNotFoundException
...
RuntimeException
NullPointerException
ArithmeticException
ClassCastException
ArrayIndexOutOfBoundsException
...
Checked、Unchecked、Error
运行 (unchecked) 异常
编译 (checked) 异常

Java 内置异常类

java.lang

Java 语言定义了一些异常类在 java.lang 标准包中。

标准运行时异常类的子类是最常见的异常类。由于 java.lang 包是默认加载到所有的 Java 程序的，所以大部分从运行时异常类继承而来的异常处置都可以直接使用。

java.lang
checked

下面的表中列出了 Java 定义在 java.lang 包中的检查性异常类。

异常	描述
ClassNotFoundException	应用程序试图加载类时，找不到相应的类，抛出该异常。
CloneNotSupportedException	当调用 Object 类中的 clone 方法克隆对象，但该对象的类无法实现 Cloneable 接口时，抛出该异常。
IllegalAccessException	拒绝访问一个类的时候，抛出该异常。
InstantiationException	当试图使用 Class 类中的 newInstance 方法创建一个类的实例，而指定的类对象因为是一个接口或是一个抽象类而无法实例化时，抛出该异常。
InterruptedException	一个线程被另一个线程中断，抛出该异常。
NoSuchFieldException	请求的变量不存在
NoSuchMethodException	请求的方法不存在

Java 的 unchecked 非检查性异常

异常	描述
ArithmeticException	当出现异常的运算条件时，抛出此异常。例如，一个整数"除以零"时，抛出此类的一个实例。
ArrayIndexOutOfBoundsException	用非法索引访问数组时抛出的异常。如果索引为负或大于等于数组大小，则该索引为非法索引。
ArrayStoreException	试图将错误类型的对象存储到一个对象数组时抛出的异常。
ClassCastException	当试图将对象强制转换为不是实例的子类时，抛出该异常。
IllegalArgumentException	抛出的异常表明向方法传递了一个不合法或不正确的参数。
IllegalMonitorStateException	抛出的异常表明某一线程已经试图等待对象的监视器，或者试图通知其他正在等待对象的监视器而本身没有指定监视器的线程。
IllegalStateException	在非法或不适当的时间调用方法时产生的信号。换句话说，即 Java 环境或 Java 应用程序没有处于请求操作所要求的适当状态下。
IllegalThreadStateException	线程没有处于请求操作所要求的适当状态时抛出的异常。
IndexOutOfBoundsException	指示某排序索引（例如对数组、字符串或向量的排序）超出范围时抛出。
NegativeArraySizeException	如果应用程序试图创建大小为负的数组，则抛出该异常。
NullPointerException	当应用程序试图在需要对象的地方使用 null 时，抛出该异常
NumberFormatException	当应用程序试图将字符串转换成一种数值类型，但该字符串不能转换为适当格式时，抛出该异常。
SecurityException	由安全管理器抛出的异常，指示存在安全侵犯。
StringIndexOutOfBoundsException	此异常由 String 方法抛出，指示索引或者为负，或者超出字符串的大小。
UnsupportedOperationException	当不支持请求的操作时，抛出该异常。

java.lang unchecked

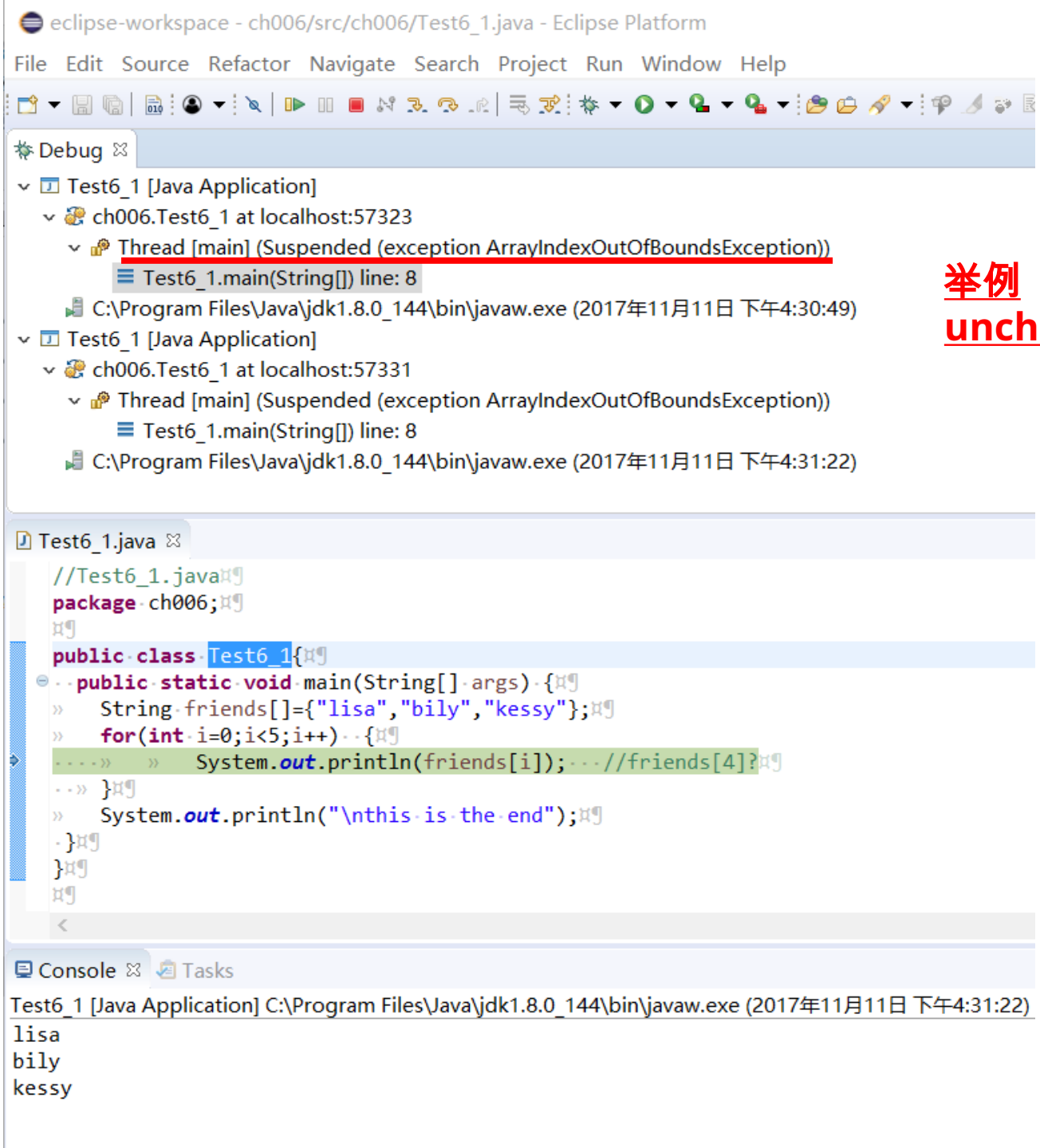
Java 异常举例 (1-unchecked)

```
public class Test6_1{
        public static void main(String[] args) {
      String friends[]={"lisa","bily","kessy"};
      for(int i=0;i<5;i++)  {
            System.out.println(friends[i]);   //friends[4]?
      }
      System.out.println("\nthis is the end");
   }
}
```

举例 unchecked

程序 Test6_1 编译正确，运行结果： java Test6_1

```
lisa
bily
kessy
java.lang.ArrayIndexOutOfBoundsException
        at Test6_1.main(Test6_1.java:5)
Exception in thread "main"
```



举例
unchecked

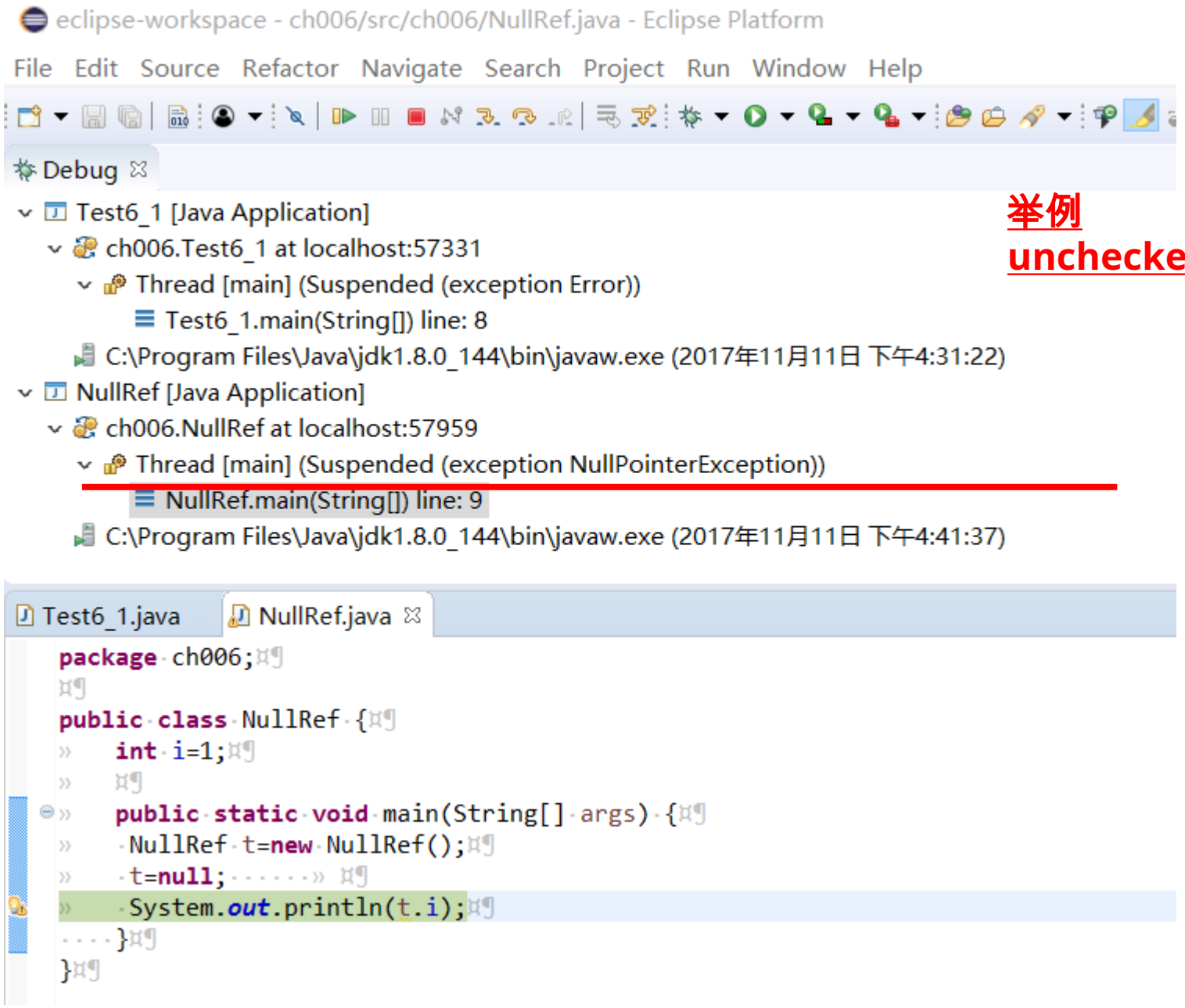
Java 异常举例 (2)

举例
unchecked

```
public class NullRef{
    int i=1;
    public static void main(String[] args) {
        NullRef t=new NullRef();
        t=null;
        System.out.println(t.i);
    }
}
```

程序 NullRef.java 编译正确，运行结果： java NullRef

*java.lang.**NullPointerException***
at NullRef.main(NullRef.java:6)
Exception in thread "main"



举例
unchecked

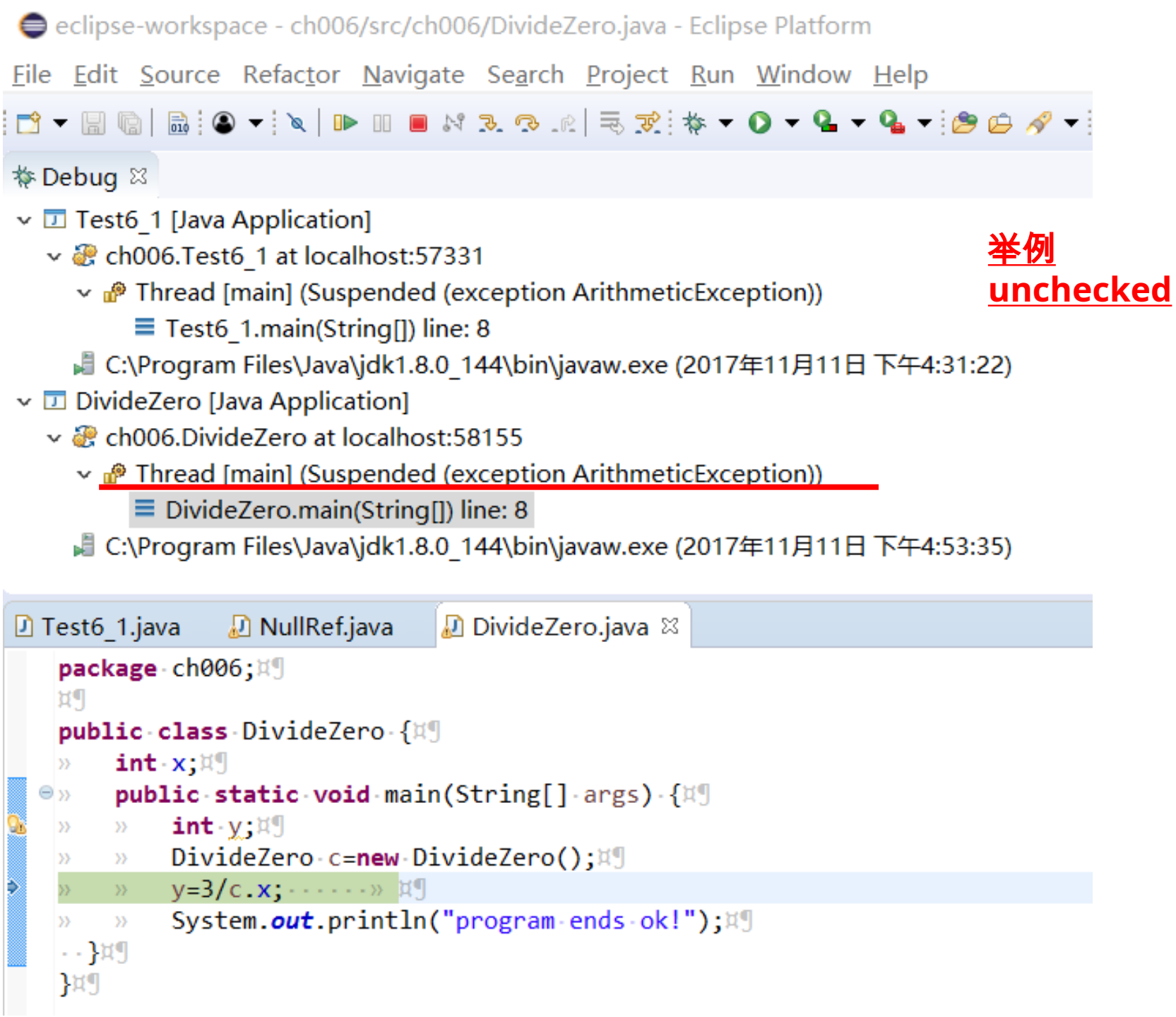
Java 异常举例 (3)

```
public class DivideZero{
    int x;
    public static void main(String[] args) {
        int y;
        DivideZero c=new DivideZero();
        y=3/c.x;
        System.out.println("program ends ok!");
    }
}
```

举例
unchecked

程序 DivideZero.java 编译正确，运行结果： java DivideZero

```
java.lang.ArithmeticException: / by zero
        at DivideZero.main(DivideZero.java:6)
Exception in thread "main"
```



举例
unchecked

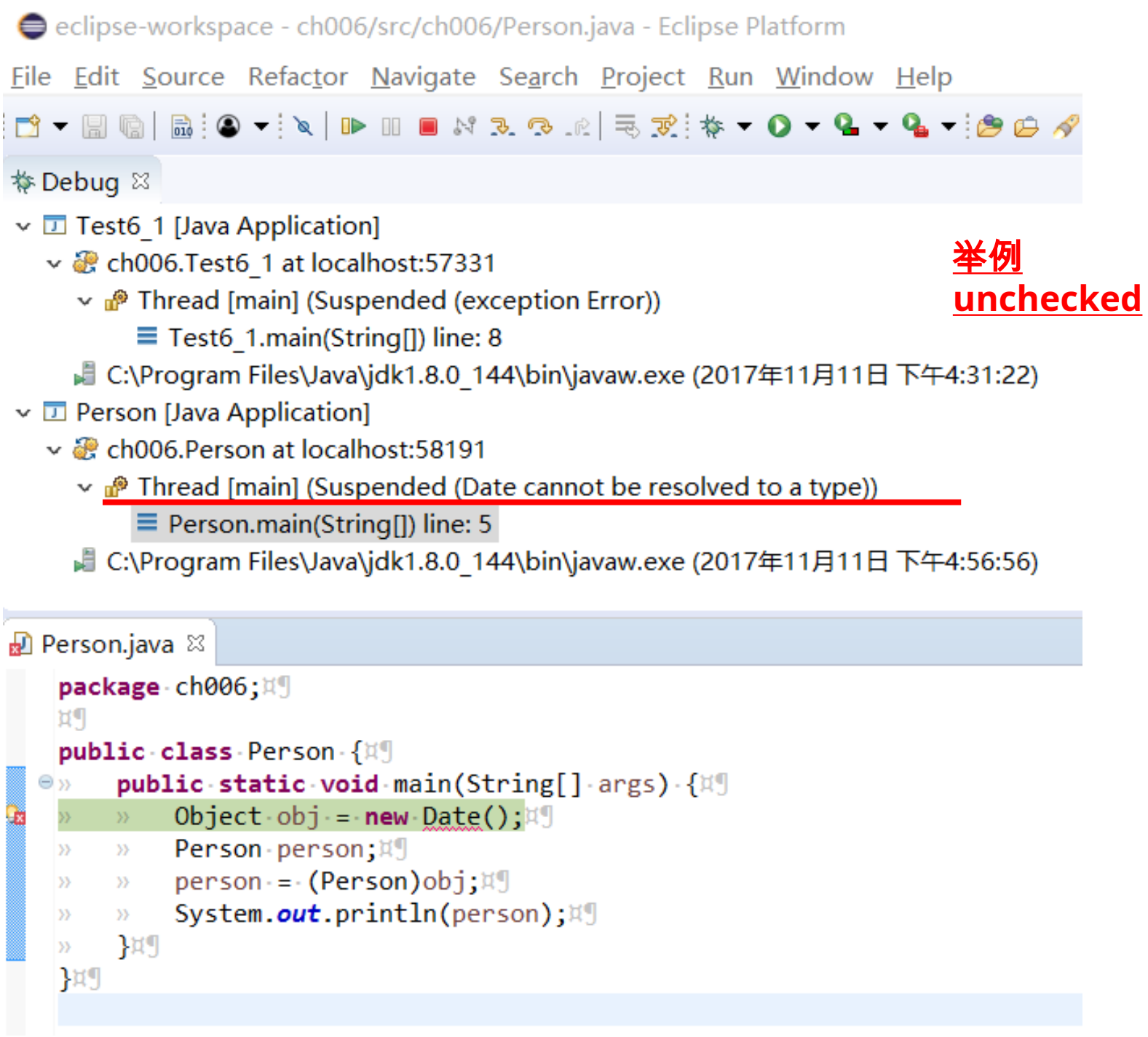
Java 异常举例 (4)

```
class Person {
  public static void main(String[] args) {
        Object obj = new Date();
        Person person;
        person = (Person)obj;
        System.out.println(person);
  }
}
```

举例
unchecked

程序 Person.java 编译正确，运行结果： java Person

*java.lang.*java.lang.ClassCastException
at Person.main(Person.java:5)
Exception in thread "main"


eclipse-workspace - ch006/src/ch006/Person.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Test6_1 [Java Application]
ch006.Test6_1 at localhost:57331
Thread [main] (Suspended (exception Error))
Test6_1.main(String[]) line: 8
C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (2017年11月11日 下午4:31:22)
Person [Java Application]
ch006.Person at localhost:58191
Thread [main] (Suspended (Date cannot be resolved to a type))
Person.main(String[]) line: 5
C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (2017年11月11日 下午4:56:56)
举例
unchecked
Person.java
package ch006;
public class Person {
public static void main(String[] args) {
Object obj = new Date();
Person person;
person = (Person)obj;
System.out.println(person);
}
}

异常处理机制（ 1 ）

在编写程序时，经常要在可能出现错误的地方加上检测的代码，如进行 x/y 运算时，要检测分母为 0 ，数据为空，输入的不是数据而是字符等。过多的分支会导致程序的代码加长，可读性差。因此采用异常机制。

Java 异常处理

Java 采用异常处理机制，将异常处理的程序代码集中在一起，与正常的程序代码分开，使得程序简洁，并易于维护。

异常处理机制（２）

- Java 提供的是异常处理的**抓抛模型 catch-throw**。
- Java 程序的执行过程中如出现异常，会生成一个异常类对象，该异常对象将被提交给 Java 运行时系统，这个过程称为**抛出 (throw) 异常**。

抓抛模型 catch-throw 模型

- 异常对象的生成
  - 由虚拟机**自动生成**：程序运行过程中，虚拟机检测到程序发生了问题，如果在当前代码中没有找到相应的处理程序，就会在后台自动创建一个对应异常类的实例对象并抛出——自动抛出

抛出异常

  - 由开发人员**手动创建**： Exception exception = new ClassCastException();
    —— 创建好的异常对象不抛出对程序没有任何影响，和创建一个普通对象一样

异常处理机制（ 3 ）

- 如果一个方法内抛出异常，该异常对象会被抛给调用者方法中处理。如果异常没有在调用者方法中处理，它继续被抛给这个调用方法的上层方法。这个过程将一直继续下去，直到异常被处理。这一过程称为**捕获 (catch) 异常**。

抓抛模型 catch-throw 模型

- **如果一个异常回到 main() 方法，并且 main() 也不处理，则程序运行终止。**

捕获异常

- **程序员通常只能处理 Exception ，而对 Error 无能为力。**

抛出异常

异常处理机制（ 4 ）

异常处理是通过 try-catch-finally 语句实现的。

抓抛模型 catch-throw 模型

语法框架 try-catch-finally

```
try{
  ……  // 可能产生异常的代码
}
catch( ExceptionName1 e ){
  ……  // 当产生 ExceptionName1 型异常时的处置措施
}
catch( ExceptionName2 e ){
……    // 当产生 ExceptionName2 型异常时的处置措施
}
[ finally{
……     // 无论是否发生异常，都无条件执行的语句
  } ]
```

捕获异常 (1)

catch 可以抓多个类型的异常事件

- **try**

  捕获异常的第一步是用 try{...} 语句块选定捕获异常的范围，将可能出现异常的代码放在 try 语句块中。

- **catch** (Exceptiontype e)

  在 catch 语句块中是对异常对象进行处理的代码。每个 try 语句块可以伴随一个或多个 catch 语句，用于处理可能产生的不同类型的异常对象。

如果明确知道产生的是何种异常，可以用该异常类作为 catch 的参数；也可以用其父类作为 catch 的参数。

比如：可以用 ArithmeticException 类作为参数的地方，就可以用 RuntimeException 类作为参数，或者用所有异常的父类 Exception 类作为参数。但不能是与 ArithmeticException 类无关的异常，如 NullPointerException（ catch 中的语句将不会执行）。

捕获异常

try-catch 必选动作

使用 try 和 catch 关键字可以捕获异常。try/catch 代码块放在异常可能发生的地方。

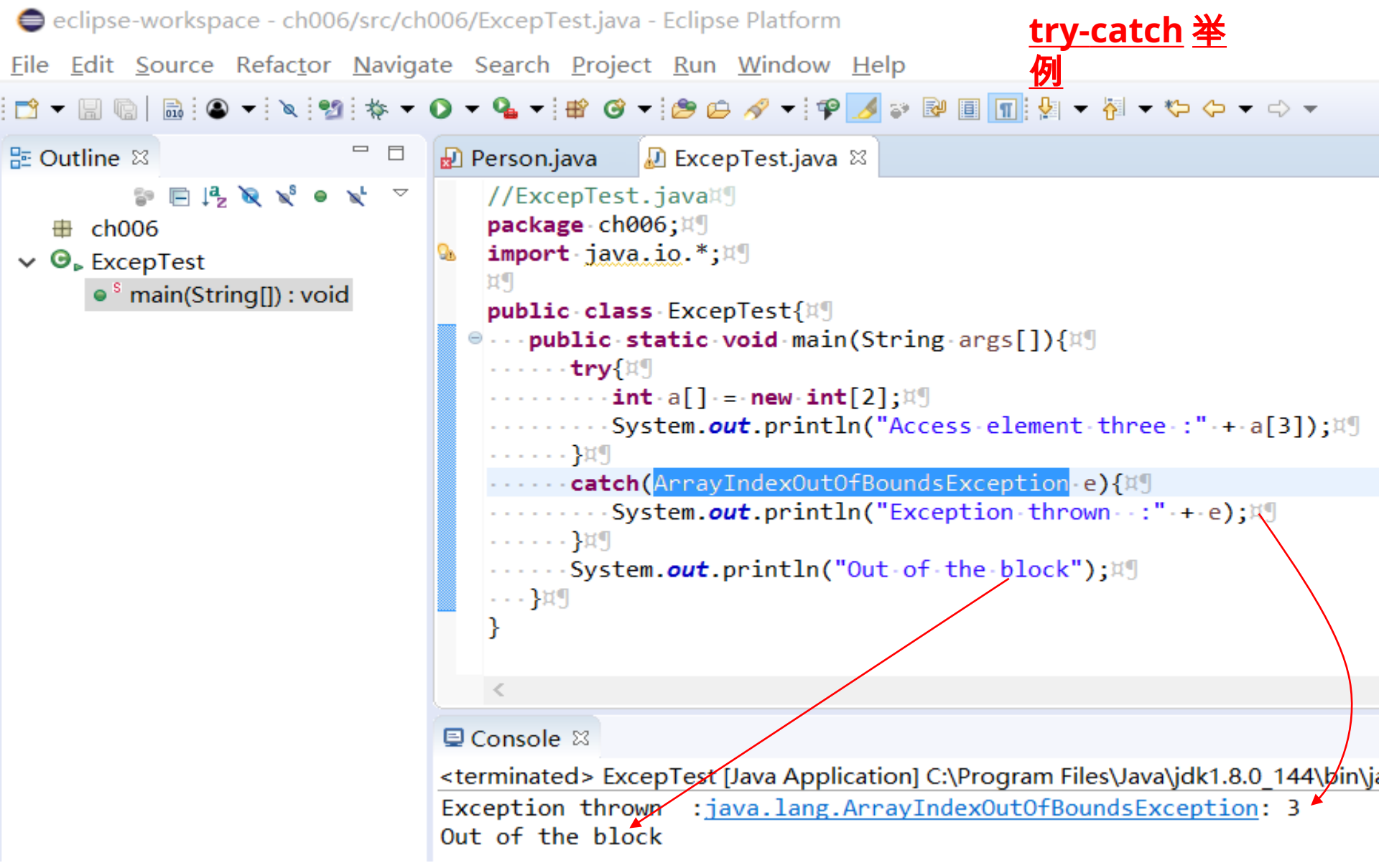
try/catch代码块中的代码称为保护代码，使用 try/catch 的语法如下：

```
try
{
    // 程序代码
}catch(ExceptionName e1)
{
    //Catch 块
}
```

Catch 语句包含要捕获异常类型的声明。当保护代码块中发生一个异常时，try 后面的 catch 块就会被检查。

如果发生的异常包含在 catch 块中，异常会被传递到该 catch 块，这和传递一个参数到方法是一样。

Try Catch 举例—对有两个元素的一个数组，当代码试图访问数组的第三个元素的时候抛出一个异常



多重捕获块

一个 try 代码块后面跟随多个 catch 代码块的情况就叫多重捕获。

多重捕获块的语法如下所示：

```
try{
    // 程序代码
}catch(异常类型1 异常的变量名1){
  // 程序代码
}catch(异常类型2 异常的变量名2){
  // 程序代码
}catch(异常类型2 异常的变量名2){
  // 程序代码
}
```

catch 可以抓多个类型的异常事件

上面的代码段包含了 3 个 catch块。

可以在 try 语句后面添加任意数量的 catch 块。

如果保护代码中发生异常，异常被抛给第一个 catch 块。

如果抛出异常的数据类型与 ExceptionType1 匹配，它在这里就会被捕获。

如果不匹配，它会被传递给第二个 catch 块。

如此，直到异常被捕获或者通过所有的 catch 块。

实例

该实例展示了怎么使用多重 try/catch。

catch 可以抓多个类型的异常事件

```
try
{
  file = new FileInputStream(fileName);
  x = (byte) file.read();
}catch(IOException i)
{
  i.printStackTrace();
  return -1;
}catch(FileNotFoundException f) //Not valid!
{
  f.printStackTrace();
  return -1;
}
```

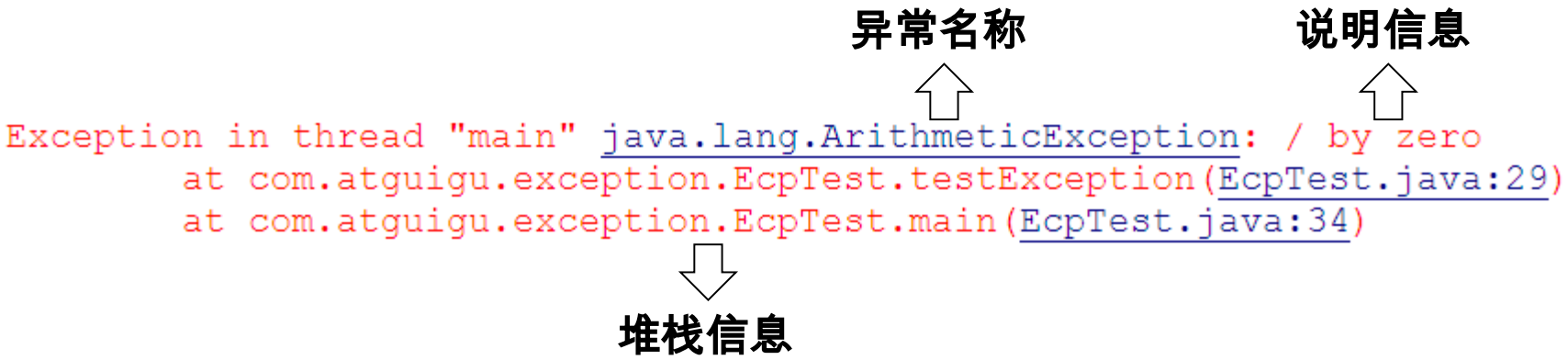
捕获异常 (2)

捕获异常通常捕获到什么？

- 捕获异常的有关信息：

  与其它对象一样，可以访问一个异常对象的成员变量或调用它的方法。

  - getMessage()　获取异常信息，返回字符串
  - printStackTrace()　获取异常类名和异常信息，以及异常出现在程序中的位置。返回值 void 。



throws/throw 关键字：

如果一个方法没有捕获一个检查性异常，那么该方法必须使用 throws 关键字来声明。throws 关键字放在方法签名的尾部。

也可以使用 throw 关键字抛出一个异常，无论它是新实例化的还是刚捕获到的。

下面方法的声明抛出一个 RemoteException 异常：

```
import java.io.*;
public class className
{
  public void deposit(double amount) throws RemoteException
  {
    // Method implementation
    throw new RemoteException();
  }
  //Remainder of class definition
}
```

throws ： checked exception
1、跟在函数后面，并注明异常类，可以跟多个异常类
2、 throws 语句中声明的异常称为受控（ checked ）的异常，通常直接派生自 Exception 类

一个方法可以声明抛出多个异常，多个异常之间用逗号隔开。

例如，下面的方法声明抛出 RemoteException 和 InsufficientFundsException：

```
import java.io.*;
public class className
{
   public void withdraw(double amount) throws RemoteException,
                            InsufficientFundsException
   {
       // Method implementation
   }
   //Remainder of class definition
}
```

throw ：
1、用在函数内，后面跟的是异常对象

throws/throw 关键字：

如果一个方法没有捕获一个检查性异常，那么该方法必须使用 throws 关键字来声明。throws 关键字放在方法签名的尾部。

也可以使用 throw 关键字抛出一个异常，无论它是新实例化的还是刚捕获到的。

下面方法的声明抛出一个 RemoteException 异常：

```
import java.io.*;
public class className
{
  public void deposit(double amou
  {
    // Method implementation
    throw new RemoteException();
  }
  //Remainder of class definition
}
```

throws ： checked exception
2、 throws 用来声明异常，让调用者只知道该功能可能出现的问题，可以给出预先得处理方式，但它自己不能处理这些异常，而需要由调用者来处理。
3、表示出现异常的可能性，并不一定会发生该异常

一个方法可以声明抛出多个异常，多个异常之间用逗号隔开。

例如，下面的方法声明抛出 RemoteException 和 InsufficientFundsException：

```
import java.io.*;
public class className
{
   public void withdraw(double amount
                                Insuffi
   {
       // Method implementation
   }
   //Remainder of class definition
}
```

throw ：
2、抛出具体的问题对象，执行到 throw 。功能就已经结束了跳转到调用者，并将具体的问题对象抛给调用者，也就是说 throw 语句独立存在时，下面不要定义其他语句，因为执行不到
3、执行 throw 则一定抛出了某种异常对象。

捕获异常 (3)

finally

- **finally**
  - 捕获异常的最后一步是通过 finally 语句为异常处理提供一个统一的出口，使得在控制流转到程序的其它部分以前，能够对程序的状态作统一的管理。
  - 不论在 try 代码块中是否发生了异常事件，catch 语句是否执行，catch 语句是否有异常，catch 语句中是否有 return ，finally 块中的语句都会被执行。
  - finally 语句是可选的

捕获异常 (4)

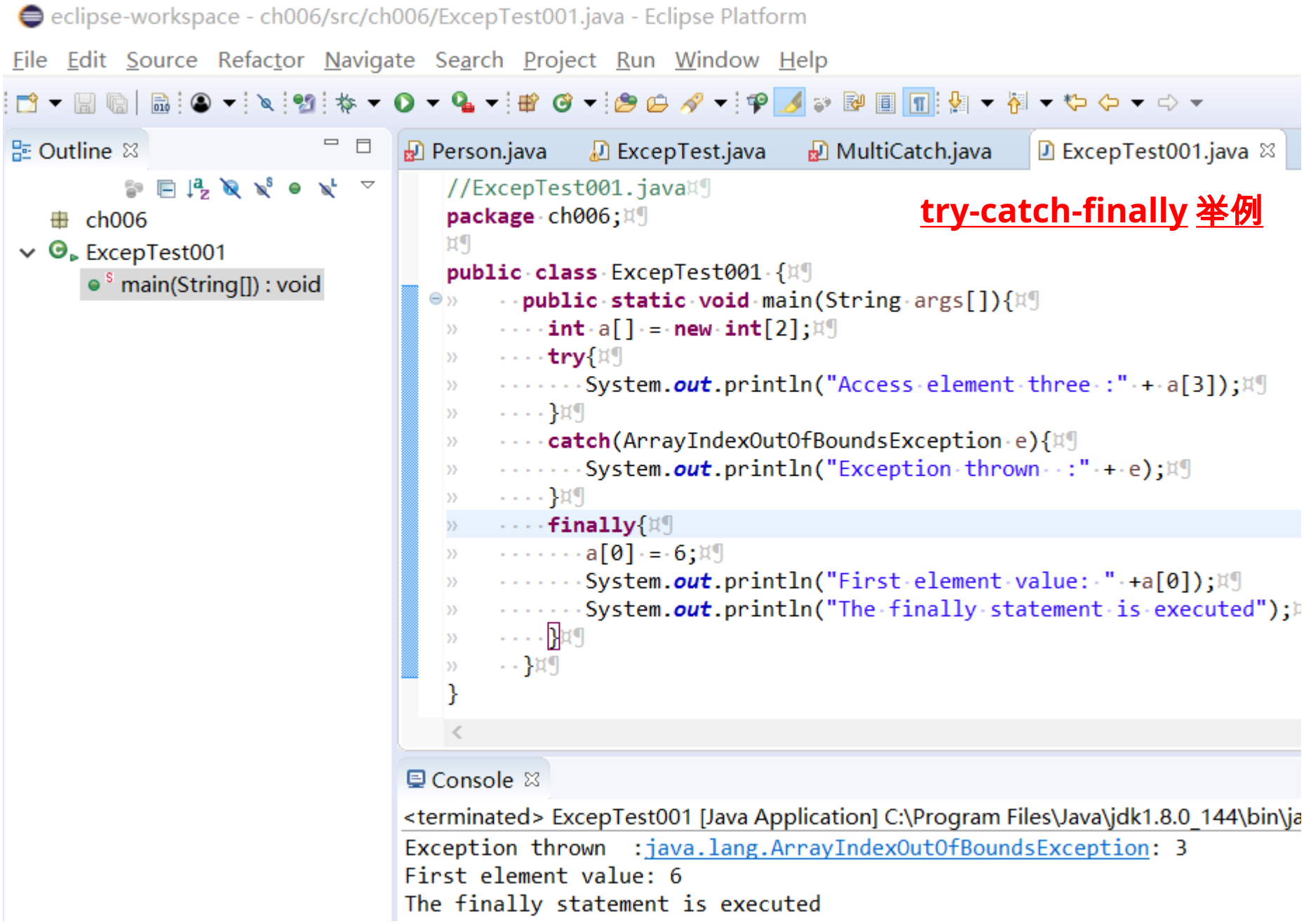
try-catch-finally 执行

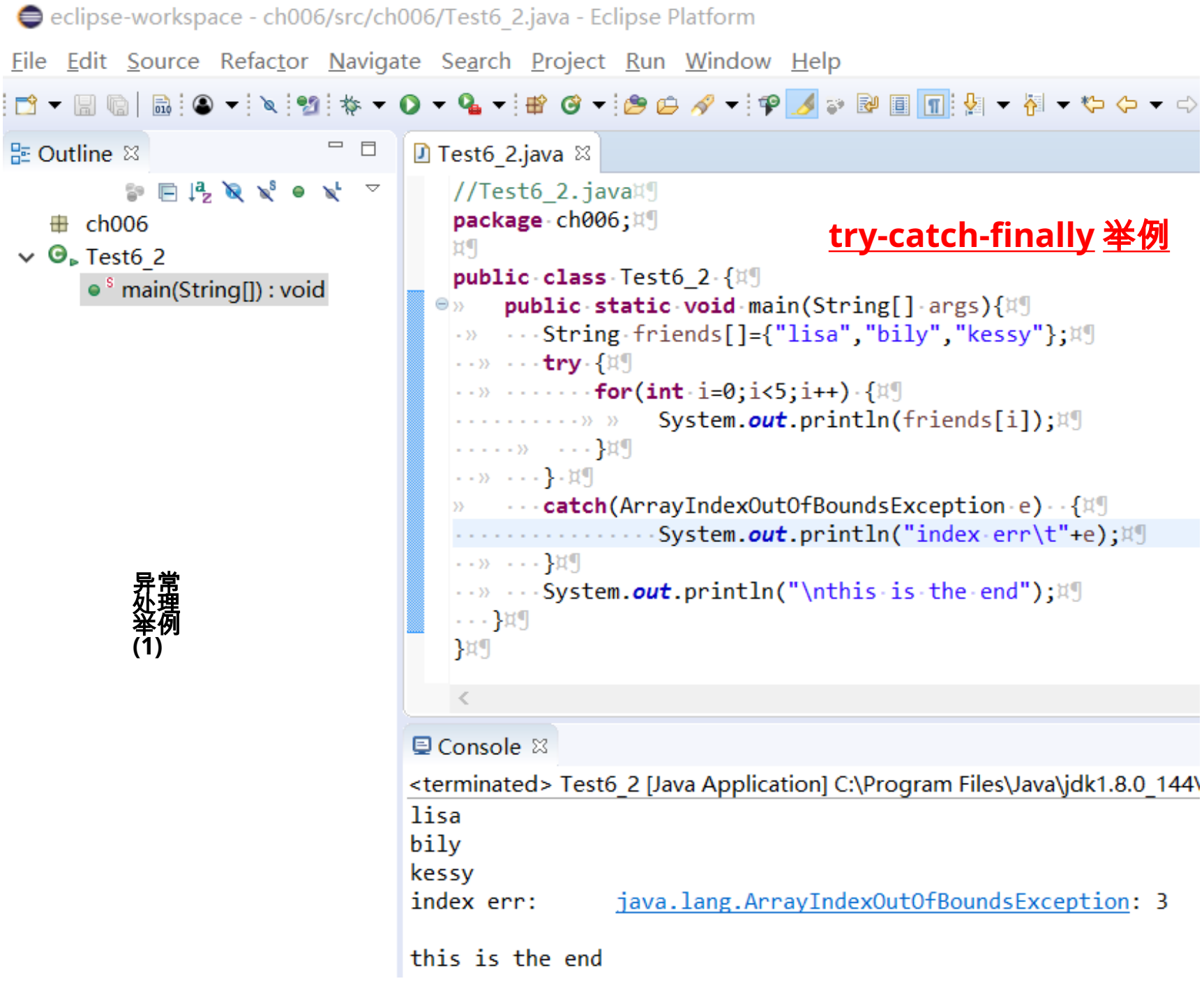
➢捕获SomeException2时：

```
try {
   语句 1;
   语句 2;
}
catch ( SomeException1 e )
{ … … …}
catch ( SomeException2 e )
{ … … …}
finally { … … …}
后面的语句;
```

➢没有捕获到异常时：

```
try {
   语句 1;
   语句 2;
}
catch ( SomeException1 e )
{ … … …}
catch ( SomeException2 e )
{ … … …}
finally { … … …}
后面的语句;
```


eclipse-workspace - ch006/src/ch006/ExcepTest001.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Outline
ch006
ExcepTest001
main(String[]) : void
Person.java
ExcepTest.java
MultiCatch.java
ExcepTest001.java
try-catch-finally 举例
//ExcepTest001.java
package ch006;
public class ExcepTest001 {
public static void main(String args[]){
int a[] = new int[2];
try{
System.out.println("Access element three :" + a[3]);
}
catch(ArrayIndexOutOfBoundsException e){
System.out.println("Exception thrown :" + e);
}
finally{
a[0] = 6;
System.out.println("First element value: " +a[0]);
System.out.println("The finally statement is executed");
}
}
}
Console
<terminated> ExcepTest001 [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\ja
Exception thrown :java.lang.ArrayIndexOutOfBoundsException: 3
First element value: 6
The finally statement is executed


eclipse-workspace - ch006/src/ch006/Test6_2.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Outline
ch006
Test6_2
main(String[]) : void
Test6_2.java
//Test6_2.java
package ch006;
public class Test6_2 {
public static void main(String[] args){
String friends[]={"lisa","bily","kessy"};
try {
for(int i=0;i<5;i++) {
System.out.println(friends[i]);
}
}
catch(ArrayIndexOutOfBoundsException e) {
System.out.println("index err\t"+e);
}
System.out.println("\nthis is the end");
}
}
try-catch-finally 举例
Console
<terminated> Test6_2 [Java Application] C:\Program Files\Java\jdk1.8.0_144\
lisa
bily
kessy
index err: java.lang.ArrayIndexOutOfBoundsException: 3
this is the end

异常处理举例(1)

实例

ExcepTest.java 文件代码：

```
public class ExcepTest{
  public static void main(String args[]){
    int a[] = new int[2];
    try{
       System.out.println("Access element three :" + a[3]);
    }catch(ArrayIndexOutOfBoundsException e){
       System.out.println("Exception thrown  :" + e);
    }
    finally{
       a[0] = 6;
       System.out.println("First element value: " +a[0]);
       System.out.println("The finally statement is executed");
    }
  }
}
```

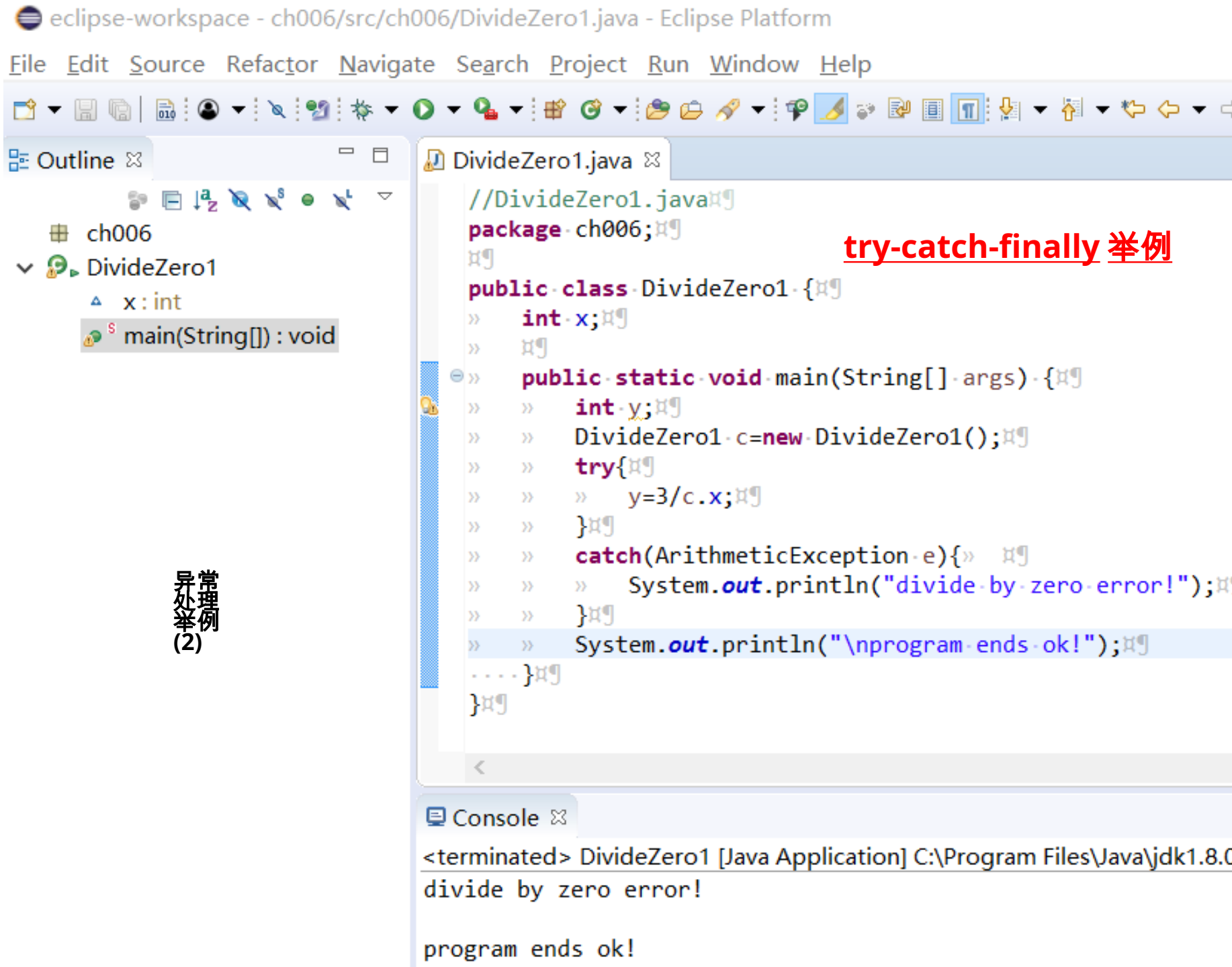
try-catch-finally 举例

以上实例编译运行结果如下：

```
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
First element value: 6
The finally statement is executed
```

注意下面事项：

- catch 不能独立于 try 存在。
- 在 try/catch 后面添加 finally 块并非强制性要求的。
- try 代码后不能既没 catch 块也没 finally 块。
- try, catch, finally 块之间不能添加任何代码。


eclipse-workspace - ch006/src/ch006/DivideZero1.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Outline
ch006
DivideZero1
x : int
main(String[]) : void
DivideZero1.java
//DivideZero1.java
package ch006;
public class DivideZero1 {
int x;
public static void main(String[] args) {
int y;
DivideZero1 c=new DivideZero1();
try{
y=3/c.x;
}
catch(ArithmeticException e){
System.out.println("divide by zero error!");
}
System.out.println("\nprogram ends ok!");
}
}
try-catch-finally 举例
Console
<terminated> DivideZero1 [Java Application] C:\Program Files\Java\jdk1.8.0
divide by zero error!
program ends ok!

异常处理举例(2)

练习 1

try-catch-finally 举例

- 编写一个类 TestException ，在 main 方法中使用 try 、 catch 、 finally ，要求：
  - 在 try 块中，编写被零除的代码。
  - 在 catch 块中，捕获被零除所产生的异常，并且打印异常信息
  - 在 finally 块中，打印一条语句。

DivideZero1.java | DivideZeroPractice.java

```
package ch006;

public class DivideZeroPractice {
    int x;

    public static void main(String[] args) {
        int y;
        DivideZero1 c=new DivideZero1();
        try{
            y=3/c.x;
        }
        catch(ArithmeticException e){
            System.out.println("divide by zero error!");
        }
        finally
        {
            System.out.println("\nprogram ends ok!");
        }
    }
}
```

try-catch-finally 举例

Problems | Console

<terminated> DivideZeroPractice [Java Application] C:\Program Files\Java\jdk

```
divide by zero error!

program ends ok!
```

体 会

try-catch-finally 举例

- 捕获和不捕获异常，程序的运行有什么不同。
- 体会 try 语句块中可能发生多个不同异常时的处理。
- 体会 finally 语句块的使用。

不捕获异常时的情况

RuntimeException

- 前面使用的异常都是 RuntimeException 类或是它的子类，这些类的异常的特点是：

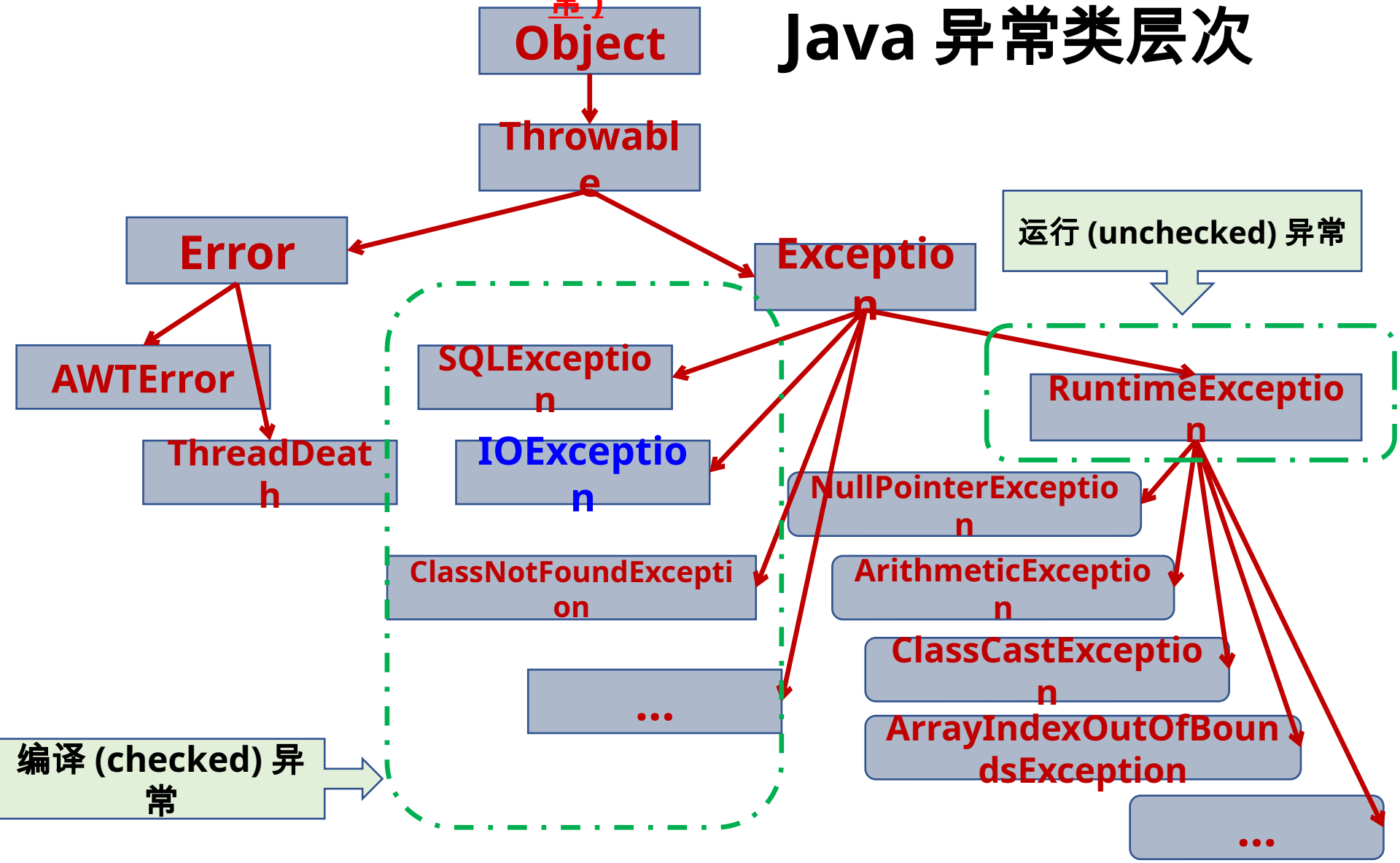
  即使没有使用 try 和 catch 捕获， Java 自己也能捕获，并且编译通过 （ 但运行时会发生异常使得程序运行终止 ）。

- 如果抛出的异常是 **IOException** 等类型的非运行时异常，则必须捕获 **checked** ，否则编译错误。也就是说，我们必须**处理编译时异常 checked ，将异常进行捕捉，转化为运行时异常处理**

IOException
Checked

CheckedException 编译器要求你必须处置的异常
IOException (操作输入流和输出流时可能出现的异常)

Java 异常类层次



ClassCastException(类型转换异常类)

IOException 异常处理举例 (1)

```
import java.io.*;
public class Test6_3{
        public static void main(String[] args)    {
                                        FileInputStream        in=new
  FileInputStream("myfile.txt");
        int b;
        b = in.read();
        while(b!= -1) {
                System.out.print((char)b);
                b = in.read();
        }
        in.close();
    }
}
```

IOException 作为 Checked Exception InputStream 需要有 try/catch 或其语法平替 throws 捕捉 编译器强制要求你处理这里的异常

IOException Checked

*Test6_3.java | FileInputStreamTest.java

```
//Test6_3.java
package ch006;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

public class Test6_3 {
    public static void main(String[] args) {
Unhandled exception type FileNotFoundException tStream("f:\\myfile.txt");
        //FileInputStream fis = new FileInputStream("src\\ch006\\FileInputStreamTest.java");
        int b;
        b = fis.read();
        //*
        while(b!= -1){
                        System.out.print((char)b);
            b = fis.read();
        }//*/
        fis.close();
        System.out.println("\n\nThat is all.");
    }
}
```

IOException 作为 Checked Exception InputStream 需要有 try/catch 或其语法平替 throws 捕捉
编译器强制要求你处理这里的异常

Problems | Javadoc | Declaration | Console

<terminated> Test6_3 [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (2017年11月11日 下午7:58:36)

```
myfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfile

That is all.
```

Test6_3.java | FileInputStreamTest.java

```java
//Test6_3.java
package ch006;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

public class Test6_3 {
    public static void main(String[] args) throws IOException{
        FileInputStream fis = new FileInputStream("f:\\myfile.txt");
        //FileInputStream fis = new FileInputStream("src\\ch006\\FileInputStreamTest.java");
        int b;
        b = fis.read();
        //*
        while(b!= -1){
                System.out.print((char)b);
            b = fis.read();
        }//*/
        fis.close();
        System.out.println("\n\nThat is all.");
    }
}
```

IOException 作为 Checked Exception InputStream 需要有 try/catch 或其语法平替 throws 捕捉 编译器强制要求你处理这里的异常

Problems | Javadoc | Declaration | Console

```
<terminated> Test6_3 [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (2017年11月11日 下午7:58:36)
myfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfile

That is all.
```

IOException 异常处理举例 (2)

```
import java.io.*;
public class Test6_3{
    public static void main(String[] args){
        try{
            FileInputStream in=new FileInputStream("myfile.txt");
            int b;
            b = in.read();
            while(b!= -1)      {
                System.out.print((char)b);
                b = in.read();
            }
            in.close();
        }catch (IOException e) {
            System.out.println(e);
        }finally {
            System.out.println(" It's ok!");
        }
    }
}
```

IOException 作为 Checked Exception InputStream 需要有 try/catch 或其语法平替 throws 捕捉 编译器强制要求你处理这里的异常

Test6_3.java | FileInputStreamTest.java

```java
//Test6_3.java
package ch006;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

public class Test6_3 {
    public static void main(String[] args) throws IOException{
        FileInputStream fis = new FileInputStream("f:\\myfile.txt");
        //FileInputStream fis = new FileInputStream("src\\ch006\\FileInputStreamTest.java");
        int b;
        b = fis.read();
        //*
        while(b!= -1){
                System.out.print((char)b);
            b = fis.read();
        }//*/
        fis.close();
        System.out.println("\n\nThat is all.");
    }
}
```

IOException 作为 Checked Exception InputStream 需要有 try/catch 或其语法平替 throws 捕捉 编译器强制要求你处理这里的异常

Problems | Javadoc | Declaration | Console

```
<terminated> Test6_3 [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (2017年11月11日 下午7:58:36)
myfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfile

That is all.
```

Test6_3.java | FileInputStreamTest.java

```java
//Test6_3.java
package ch006;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

public class Test6_3 {
    public static void main(String[] args){
        try {
        FileInputStream fis = new FileInputStream("f:\\myfile.txt");
        //FileInputStream fis = new FileInputStream("src\\ch006\\FileInputStreamTest.java");
        int b;
        b = fis.read();
        //*
        while(b!= -1){
                        System.out.print((char)b);
            b = fis.read();
        }//*/
        fis.close();
        }
        catch (IOException e) {
            System.out.println(e);
        }
        finally {
            System.out.println("\n\nThat is all.");
        }
    }
}
```

IOException 作为 Checked Exception InputStream 需要有 try/catch 或其语法平替 throws 捕捉
编译器强制要求你处理这里的异常

Problems | Javadoc | Declaration | Console

```
<terminated> Test6_3 [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (2017年11月11日 下午8:06:14)
myfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfile

That is all.
```

声明抛出异常 (1)

- **声明抛出异常是 Java 中处理异常的第二种方式** throws
  - **如果一个方法 (中的语句执行时) 可能生成某种异常，但是并不能确定如何处理这种异常，则此方法应显示地声明抛出异常，表明该方法将不自行对这些异常进行处理，而由该方法的调用者负责处理。**
  - 在方法声明中用 throws 语句可以声明抛出异常的列表， throws 后面的异常类型可以是方法中产生的异常类型，也可以是它的父类。
- 声明抛出异常举例：

```
public void readFile(String file) throws FileNotFoundException {
        ……
        // 读文件的操作可能产生 FileNotFoundException 类型的异常
        FileInputStream fis = new FileInputStream(file);
        ……..
}
```

Test6_3.java | FileInputStreamTest.java

```java
//FileInputStreamTest.java
package ch006;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

public class FileInputStreamTest {
    public static void main(String[] args) throws IOException{
        //创建字节输入流，路径可以是绝对路径，也可以是相对路径（相对于项目工程为根目录）
        //FileInputStream fis = new FileInputStream("F:\eclipse-workspace\ch006\src\ch006\\FileInputStreamTest.java");
        FileInputStream fis = new FileInputStream("src\\ch006\\FileInputStreamTest.java");

        /*
        File file = new File("src\ch006\\FileInputStreamTest.java");
        System.out.println(file.getAbsolutePath());
        FileInputStream fis = new FileInputStream(file);
       */
        //创建一个长度为1024的内存空间，1024足够大，可以一次将文件读取完，不会出现中文注释乱码问题
        byte[] bbuf = new byte[1024];

        //用于保存实际读取的字节数
        int hasRead = 0;
        //使用循环来重复读取数据
        while( (hasRead = fis.read(bbuf)) > 0){

            //将字节数组转换为字符串输出
            System.out.print(new String(bbuf,0,hasRead));

        }

        //关闭文件输出流，放在finally块里更安全
        fis.close();
    }
}
```

声明抛出异常 (2)

```
import java.io.*;
public class Test6_4{
  public static void main(String[] args){
        Test6_4 t = new Test6_4();
        try{
              t.readFile();
         }catch(IOException e){   }
  }
  public void readFile() throws IOException {
        FileInputStream in=new FileInputStream("myfile.txt");
        int b;
        b = in.read();
        while(b!= -1)   {
              System.out.print((char)b);
              b = in.read();
        }
        in.close();
  }
}
```

方法调用的形式
编译器强制要求上级调用
的程序处理这里的异常

Test6_3.java | Test6_4.java

```java
//Test6_4.java
package ch006;
import java.io.*;

public class Test6_4 {
    public static void main(String[] args){
        Test6_4 t = new Test6_4();
        try{
            t.readFile();
        }
        catch(IOException e){...}
        System.out.println("\n\nThat's all.");
    }

    public void readFile() throws IOException {
        FileInputStream in=new FileInputStream("f:\\myfile.txt");
        int b;
        b = in.read();
        while(b!= -1)...{
            System.out.print((char)b);
            b = in.read();
        }
        in.close();
    }
}
```

方法调用的形式
编译器强制要求上级调用的程序处理这里的异常

Problems | Javadoc | Declaration | Console

```
<terminated> Test6_4 [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (2017年1
myfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfile

That's all.
```

Test6_3.java | Test6_4.java

```
public class Test6_4 {
    public static void main(String[] args){
        Test6_4 t = new Test6_4();
        try{
            t.readFile();
        }
        catch(IOException e){...}
        System.out.println("\n\nThat's all.");
    }

    public void readFile() throws IOException {//IOException
        //FileInputStream in=new FileInputStream("f:\\myfile.txt");
        FileInputStream in = getInputStream("f:\\myfile.txt");
        int b;
        b = in.read();
        while(b!= -1) {
            System.out.print((char)b);
            b = in.read();
        }
        in.close();
    }

    public FileInputStream getInputStream (String filepath) throws FileNotFoundException{
        return new FileInputStream(filepath);
    }
}
```

方法调用的形式
编译器强制要求上级调用
的程序处理这里的异常

Problems | Javadoc | Declaration | Console

```
<terminated> Test6_4 [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (2017年11月11日 下午8:37:03)
myfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfilemyfile

That's all.
```

重写方法声明抛出异常的原则

- **重写方法不能抛出比被重写方法范围更大的异常类型。**

在多态的情况下，对 methodA() 方法的调用 - 异常的捕获按父类声明的异常处理。

重写中的异常

```
public class A {
        public void methodA() throws IOException {
            ……
        } }
public class B1 extends A {
        public void methodA() throws
FileNotFoundException {
            ……
        } }
public class B2 extends A {
        public void methodA() throws Exception {   // 报错
            ……
        } }
```

人工抛出异常

- Java 异常类对象除在程序执行过程中出现异常时由系统自动生成并抛出，也可根据需要人工创建并抛出。
  - 首先要生成异常类对象，然后通过 throw 语句实现抛出操作 (提交给 Java 运行环境) 。

```
IOException e = new IOException();
throw e;
```

自己定义异常

必须是 Throwable

  - 可以抛出的异常必须是 Throwable 或其子类的实例。下面的语句在编译时将会产生语法错误：

```
throw new String("want to throw");
```

创建用户自定义异常类

自己定义异常RuntimeException

- 一般地，用户自定义异常类都是 RuntimeException 的子类。
- 自定义异常类通常需要编写几个重载的构造器。
- 自定义的异常类对象通过 throw 抛出。
- 自定义异常最重要的是异常类的名字，当异常出现时，可以根据名字判断异常类型。

声明自定义异常

在 Java 中你可以自定义异常。编写自己的异常类时需要记住下面的几点。

- 所有异常都必须是 Throwable 的子类。
- 如果希望写一个检查性异常类，则需要继承 Exception 类。
- 如果你想写一个运行时异常类，那么需要继承 RuntimeException 类。

可以像下面这样定义自己的异常类：

```
class MyException extends Exception{
}
```

自己定义异常

自定义异常类型继承

Exception

只继承Exception 类来创建的异常类是检查性异常类。

下面的 InsufficientFundsException 类是用户定义的异常类，它继承自 Exception。

一个异常类和其它任何类一样，包含有变量和方法。

创建用户自定义异常类

用户自定义异常类 MyException ，用于描述数据取值范围错误信息。用户自己的异常类**必须继承**现有的异常类。

```
class MyException extends Exception {
        static final long serialVersionUID = 1L;
        private int idnumber;
        public MyException(String message, int id) {
                super(message);
                this.idnumber = id;
        }
        public int getId() {
                return idnumber;
        }
}
```

自己定义异常

自定义异常类型继承

Exception

Test6_5.java | MyException.java

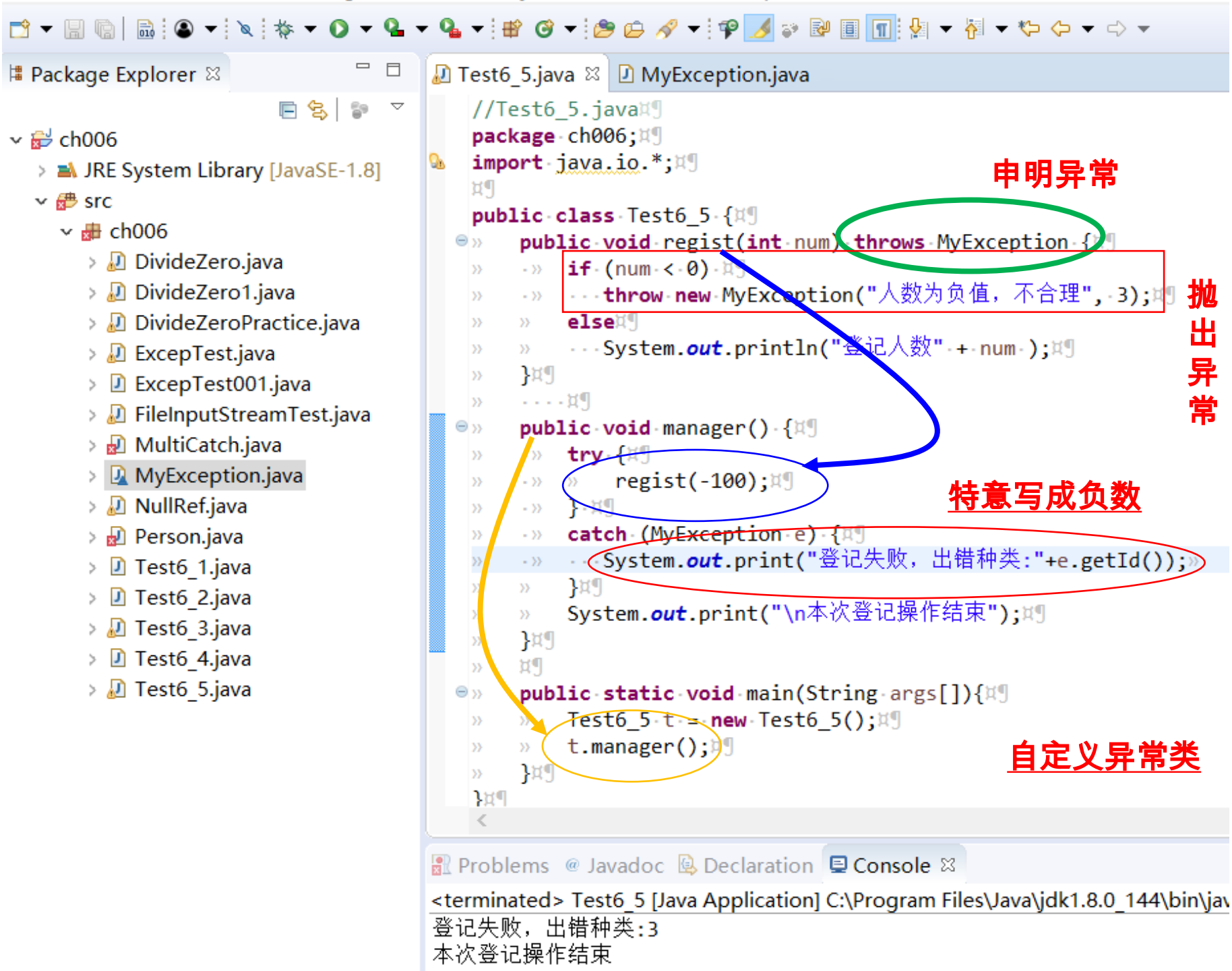
```
//MyException.java
package ch006;

class MyException extends Exception {
    static final long serialVersionUID = 1L;
    private int idnumber;
    public MyException(String message, int id) {
        super(message);
        this.idnumber = id;
    }
    public int getId() {
        return idnumber;
    }
}
```

自定义异常类

使用用户自定义异常类

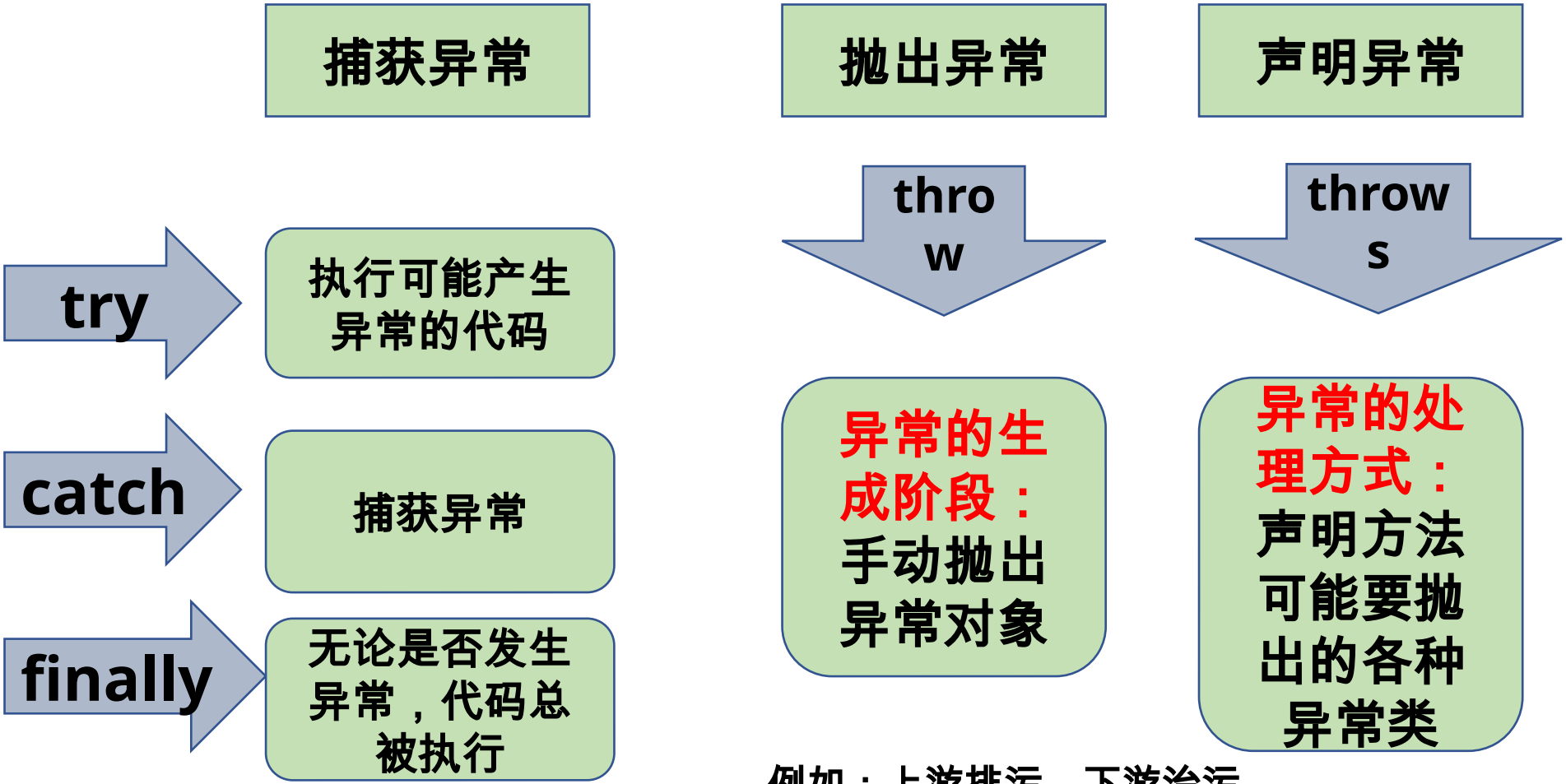
```
public class Test6_5{
    public void regist(int num) throws MyException {
        if (num < 0)
            throw new MyException( “人数为负值，不合理”，3);
        else
            System.out.println(" 登记人数 " + num );
    }

    public void manager() {
        try {
            regist(100);
        } catch (MyException e) {
            System.out.print(" 登记失败，出错种类 "+e.getId());

        }
        System.out.print(" 本次登记操作结束 ");
    }

    public static void main(String args[]){
        Test6_5 t = new Test6_5();
        t.manager();
    }
}
```

自定义异常类


Package Explorer
ch006
JRE System Library [JavaSE-1.8]
src
ch006
DivideZero.java
DivideZero1.java
DivideZeroPractice.java
ExcepTest.java
ExcepTest001.java
FileInputStreamTest.java
MultiCatch.java
MyException.java
NullRef.java
Person.java
Test6_1.java
Test6_2.java
Test6_3.java
Test6_4.java
Test6_5.java
Test6_5.java
MyException.java
//Test6_5.java
package ch006;
import java.io.*;
public class Test6_5 {
public void regist(int num) throws MyException {
if (num < 0)
throw new MyException("人数为负值，不合理", 3);
else
System.out.println("登记人数" + num);
}
public void manager() {
try {
regist(-100);
}
catch (MyException e) {
System.out.print("登记失败，出错种类:"+e.getId());
}
System.out.print("\n本次登记操作结束");
}
public static void main(String args[]){
Test6_5 t = new Test6_5();
t.manager();
}
}
申明异常
抛出异常
特意写成负数
自定义异常类
Problems
Javadoc
Declaration
Console
<terminated> Test6_5 [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\jav
登记失败，出错种类:3
本次登记操作结束

异常处理 5 个关键字

抓抛模型



例如：上游排污，下游治污

方法调用的形式

编译器强制要求上级调用的程序处理这里的异常

一个银行账户的模拟，通过银行卡的号码完成识别，可以进行存钱和取钱的操作

Test6_5.java | MyException.java | InsufficientFundsException.java | *Chec

```
//InsufficientFundsException.java
//一个银行账户的模拟，通过银行卡的号码完成识别，可以进行存钱和取钱的操作
package ch006;
import java.io.*;

//自定义异常类，继承Exception类
public class InsufficientFundsException extends Exception
{
    //此处的amount用来储存当出现异常（取出钱多于余额时）所缺乏的钱
    private double amount;
    public InsufficientFundsException(double amount)
    {
        this.amount = amount;
    }

    public double getAmount()
    {
        return amount;
    }
}
```

另一个自定义异常例子

Test6_5.java | MyException.java | InsufficientFundsException.java | *CheckingAccount.java

```java
//CheckingAccount.java
//一个银行账户的模拟，通过银行卡的号码完成识别，可以进行存钱和取钱的操作
package ch006;
import java.io.*;

//此类模拟银行账户
public class CheckingAccount
{
//balance为余额，number为卡号
private double balance;
private int number;
public CheckingAccount(int number)
{
    this.number = number;
}
//方法：存钱
public void deposit(double amount)
{
    balance += amount;
}
//方法：取钱
public void withdraw(double amount) throws InsufficientFundsException
{
    if(amount <= balance)
    {
        balance -= amount;
    }
    else
    {
        double needs = amount - balance;
        throw new InsufficientFundsException(needs);
    }
}
//方法：返回余额
public double getBalance()
{
    return balance;
}
//方法：返回卡号
public int getNumber()
{
    return number;
}
}
```

账号

属性：账号

功能：存钱、取钱*、查账

needs 不合理多取的钱

Test6_5.java | MyException.java | BankDemo.java | InsufficientFundsException.jav

```java
//BankDemo.java
//一个银行账户的模拟，通过银行卡的号码完成识别，可以进行存钱和取钱的操作
package ch006;

public class BankDemo
{
    public static void main(String [] args)
    {
        CheckingAccount c = new CheckingAccount(101);
        System.out.println("Depositing $500...");
        c.deposit(500.00);
        try
        {
            System.out.println("\nWithdrawing $100...");
            c.withdraw(100.00);
            System.out.println("\nWithdrawing $600...");
            c.withdraw(600.00);
        }catch(InsufficientFundsException e)
        {
            System.out.println("Sorry, but you are short $"+ e.getAmount());
            e.printStackTrace();
            System.out.println("\nThat's all.");
        }
    }
}
```

示例代码

Problems | Javadoc | Declaration | Console

```
<terminated> BankDemo [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (201
Depositing $500...

Withdrawing $100...

Withdrawing $600...
Sorry, but you are short $200.0
ch006.InsufficientFundsException

That's all.
        at ch006.CheckingAccount.withdraw(CheckingAccount.java:30)
        at ch006.BankDemo.main(BankDemo.java:16)
```

void java.lang.Throwable.printStackTrace()

Prints this throwable and its backtrace to the standard error stream. This method prints a stack trace for this `Throwable` object on the error output stream that is the value of the field `System.err`. The first line of output contains the result of the `toString()` method for this object. Remaining lines represent data previously recorded by the method `fillInStackTrace()`. The format of this information depends on the implementation, but the following example may be regarded as typical:

```
java.lang.NullPointerException
        at MyClass.mash(MyClass.java:9)
        at MyClass.crunch(MyClass.java:6)
```

printStackTrace()

printStackTrace()方法的意思是：在命令行打印异常信息在程序中出错的位置及原因。

只要记住这是对虚拟机内部的错误信息的提交并显示到前台的人机交换的机制就行了。

举例：编译上面后三个文件抛出的错误信息中：

```
InsufficientFundsException
//类名.方法名（文件名：抛出异常的代码所在行数）
at CheckingAccount.withdraw(CheckingAccount.java:25)
//类名。方法名（文件名：catch到异常的代码所在行数）
at BankDemo.main(BankDemo.java:13)
```

```
Depositing $500...

Withdrawing $100...

Withdrawing $600...
Sorry, but you are short $200.0
ch006.InsufficientFundsException

That's all.
        at ch006.CheckingAccount.withdraw(CheckingAccount.java:30)
        at ch006.BankDemo.main(BankDemo.java:16)
```

通用异常

在Java中定义了两种类型的异常和错误。

- **JVM(Java虚拟机) 异常：**由 JVM 抛出的异常或错误。例如：NullPointerException 类，ArrayIndexOutOfBoundsException 类，ClassCastException 类。
- **程序级异常：**由程序或者API程序抛出的异常。例如 IllegalArgumentException 类，IllegalStateException 类。

回顾

名词解释：

- 1、检查性异常: 不处理编译不能通过
- 2、非检查性异常:不处理编译可以通过，如果有抛出直接抛到控制台
- 3、运行时异常: 就是非检查性异常
- 4、非运行时异常: 就是检查性异常

veblen 7个月前 (04-14)

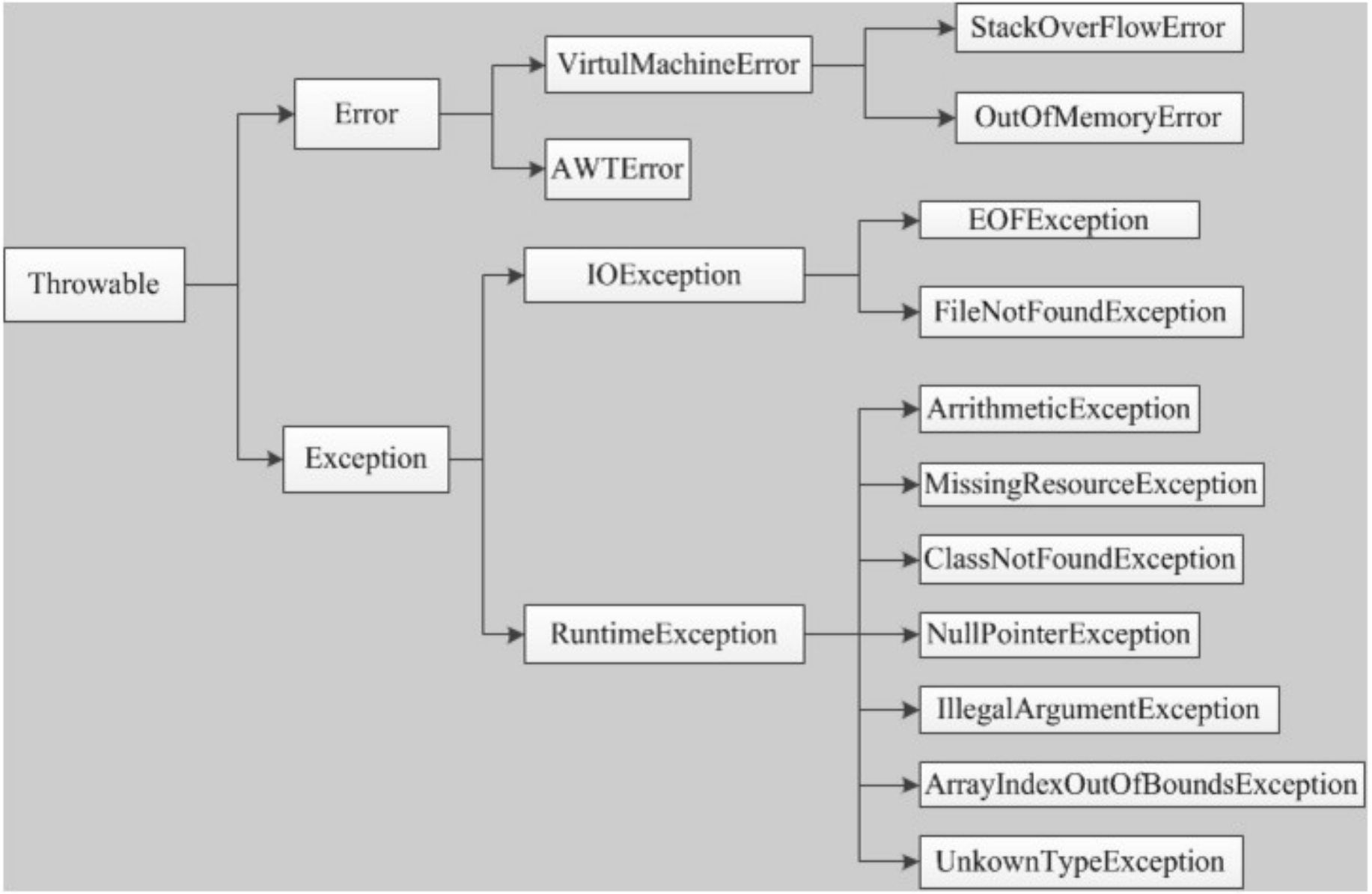
异常使用可遵循下面的原则：

1：在当前方法被覆盖时，覆盖他的方法必须抛出相同的异常或异常的子类；

2：在当前方法声明中使用try-catch语句捕获异常；

3：如果父类抛出多个异常，则覆盖方法必须抛出那些异常的一个子集，不能抛出新异常。

为我所望 5个月前 (05-29)


Throwable
Error
VirtulMachineError
StackOverFlowError
OutOfMemoryError
AWTError
Exception
IOException
EOFException
FileNotFoundException
RuntimeException
ArrithmeticException
MissingResourceException
ClassNotFoundException
NullPointerException
IllegalArgumentException
ArrayIndexOutOfBoundsException
UnkownTypeException

如图可以看出所有的异常跟错误都继承与Throwable类，也就是说所有的异常都是一个对象。

从大体来分异常为两块：

1、error---错误 ： 是指程序无法处理的错误，表示应用程序运行时出现的重大错误。例如jvm运行时出现的OutOfMemoryError以及Socket编程时出现的端口占用等程序无法处理的错误。

2、Exception --- 异常 ：异常可分为运行时异常跟编译异常

- 1）**运行时异常**：即RuntimeException及其之类的异常。这类异常在代码编写的时候不会被编译器所检测出来，是可以不需要被捕获，但是程序员也可以根据需要进行捕获抛出。常见的RUNtimeException有：NullpointException（空指针异常），ClassCastException（类型转换异常），IndexOutOfBoundsException（数组越界异常）等。
- 2）**编译异常**：RuntimeException以外的异常。这类异常在编译时编译器会提示需要捕获，如果不进行捕获则编译错误。常见编译异常有：IOException（流传输异常），SQLException（数据库操作异常）等。

3、java处理异常的机制：抛出异常以及捕获异常 ，一个方法所能捕捉的异常，一定是Java代码在某处所抛出的异常。简单地说，异常总是先被抛出，后被捕捉的。

4、throw跟throws的区别:

```
public void test thorws Exception(){
    throw new Exception();
}
```

从上面这一段代码可以明显的看出两者的区别。throws表示一个方法声明可能抛出一个异常，throw表示此处抛出一个已定义的异常（可以是自定义需继承Exception，也可以是java自己给出的异常类）。

5、接下来看一下如何捕获异常：

1）首先java对于异常捕获使用的是try---catch或try --- catch --- finally 代码块，程序会捕获try代码块里面的代码，若捕获到异常则进行catch代码块处理。若有finally则在catch处理后执行finally里面的代码。然而存在这样两个问题：

a.看如下代码：

```
try{
    //待捕获代码
}catch（Exception e）{
    System.out.println("catch is begin");
    return 1 ;
}finally{
     System.out.println("finally is begin");
}
```

在catch里面有一个return，那么finally会不会被执行呢？答案是肯定的，上面代码的执行结果为：

```
catch is begin
finally is begin
```

也就是说会先执行catch里面的代码后执行finally里面的代码最后才return1 ；

b.看如下代码：

```
try{
   //待捕获代码
}catch（Exception e）{
    System.out.println("catch is begin");
    return 1 ;
}finally{
     System.out.println("finally is begin");
     return 2 ;
}
```

代码分析

在b代码中输出结果跟a是一样的，然而返回的是return 2 ； 原因很明显，就是执行了finally后已经return了，所以catch里面的return不会被执行到。也就是说finally永远都会在catch的return前被执行。

专门处理特定的异常

6、对于异常的捕获不应该觉得方便而将几个异常合成一个Exception进行捕获，比如有IO的异常跟SQL的异常，这样完全不同的两个异常应该分开处理！而且在catch里处理异常的时候不要简单的e.printStackTrace()，而是应该进行详细的处理。比如进行console打印详情或者进行日志记录。

注意：异常和错误的区别：异常能被程序本身可以处理，错误是无法处理。

Demo.java | Test6_5.java | MyException.java | BankDemo.java | InsufficientFundsException.java

```java
//Demo.java
/*异常:
 * finally不一定被执行，，例如catch块中有退出系统的语句System.exit(-1); finally就不会被执行
*/
package ch006;
import java.io.*;

public class Demo {
public static void main(String[] args) {
        FileReader fr=null;//检查异常1.打开文件
        try {
            fr=new FileReader("f:\\myfile.txt");
            System.out.println("myfile");//上面出现异常时，这句代码就不执行了
        } catch (Exception e) {
            System.out.println("进入catch");// 文档读取异常
            // System.exit(-1);
            System.out.println("message="+e.getLocalizedMessage());  //没有报哪一行出错
            e.printStackTrace();   // 打印出错异常还出现可以报出错先异常的行
        }
        // 这个语句块不管发生没有发生异常，都会执行。一般需要去关闭资源，文件，连接，内存等
        finally
        {   System.out.println("进入finally");
            if(fr!=null);
            {   try {
                    fr.close();
                } catch (Exception e) {
                    // TODO: handle exception
                    e.printStackTrace();
                }
            }
        }
        System.out.println("OK");
    }
}
```

顺利执行的效果

```
myfile
进入finally
OK
```

Demo.java | Test6_5.java | MyExcepti...

```
//Demo.java
/*异常:
 * finally不一定被执行，，例如catch块中
*/
package ch006;
import java.io.*;

public class Demo {
public static void main(String[] args) {

        FileReader fr=null;//检查异常1.
        try {
            fr=new FileReader("f:\\myfile1.txt");
            System.out.println("myfile");//上面出现异常时，这句代码就不执行了
        } catch (Exception e) {
            System.out.println("进入catch");// 文档读取异常
            // System.exit(-1);
            System.out.println("message="+e.getLocalizedMessage());  //没有报哪一行出错
            e.printStackTrace();   // 打印出错异常还出现可以报出错先异常的行
        }
        // 这个语句块不管发生没有发生异常，都会执行。一般需要去关闭资源，文件，连接，内存等
        finally
        {   System.out.println("进入finally");
            if(fr!=null);
            {   try {
                    fr.close();
                } catch (Exception e) {
                    // TODO: handle exception
                    e.printStackTrace();
                }
            }
        }
        System.out.println("OK");
    }
}
```

```
进入catch
message=f:\myfile1.txt (系统找不到指定的文件。)
java.io.FileNotFoundException: f:\myfile1.txt (系统找不到指定的文件。)
        at java.io.FileInputStream.open0(Native Method)
        at java.io.FileInputStream.open(FileInputStream.java:195)
        at java.io.FileInputStream.<init>(FileInputStream.java:138)
        at java.io.FileInputStream.<init>(FileInputStream.java:93)
        at java.io.FileReader.<init>(FileReader.java:58)
        at ch006.Demo.main(Demo.java:13)
进入finally
java.lang.NullPointerException
        at ch006.Demo.main(Demo.java:26)
OK
```

不顺利执行的效果

本周家庭作业

1. 下面关于类及其修饰符的描述中，不正确的是
   A. abstract 类只能用来派生子类，不能用来创建对象实例。
   B. final 类不但可以用来派生子类，也可以用来创建对象实例。
   C. abstract 与 final 不能用来修饰同一个类。
   D. abstract 方法只能声明在 abstract 类中，但 abstract 类中可以没有 abstract 方法。

2. 简述 Java 运行时异常与一般异常的区别。
3. 什么是异常？简述 Java 异常处理机制。
4. Error 类和 Exception 类有什么区别？
5. 什么是抽象类？抽象类与接口的区别是什么？
6. 多态是面向对象程序设计的一个重要特征，在 Java 语言有哪些表现形式？分别简述它们的含义。

7、阅读下列程序，分析该程序是否有语法错误，如果有错误则进行修改，如果无错误写出该程序的执行结果。

```
import java.io.IOException;
public class ExceptionTest {
  public static void main(String args[]){
    try{
       method();
    }catch(IOException e) {
       System.out.println("Caught Exception");
    }
  }

  public void method() {
    throw new IOException();
  }
}
```

参考答案：程序代码错误，首先 method 是非静态方法，不能出现在 main 中；其次，即使 method 是静态方法，IOException 是可检测异常，必须要被 catch 捕获并被 throws 声明，但程序中没有声明，所以 try 语句块不可能抛出 IOException 这个异常，而 catch 也就无法捕获它。

修改如下：

static public void method() throws IOException

这段代码存在几个问题，主要是与异常处理相关的问题。以下是代码的错误和如何修复它们：

1. **方法method()没有处理异常**：在`method()`方法中抛出了一个`IOException`异常，但是没有在方法内或方法签名中声明异常，也没有用`try-catch`块捕获它。修复方法是要么在方法内部处理异常，要么在方法签名中使用`throws`关键字声明它。

```java
public void method() throws IOException {
    throw new IOException();
}
```

1. **try-catch块放置错误**：在`main`方法中，`try-catch`块的位置不正确。`method()`方法是在`main`方法中调用的，所以应该将`try-catch`块放在`method()`方法调用的地方。

```java
public static void main(String args[]) {
    ExceptionTest test = new ExceptionTest();
    try {
        test.method();
    } catch (IOException e) {
        System.out.println("Caught Exception");
    }
}
```

截图(Alt + A)

修复后的代码会在`main`方法中创建一个`ExceptionTest`对象，然后调用`method()`方法，并在调用中处理可能抛出的`IOException`异常。

这样修复后，代码将能够正常编译和运行，而且可以捕获和处理`method()`方法中抛出的异常。

8、描述程序的执行流程，指出存在的问题和修改方案。

```
import java.lang.*;
public class ExceptionTest {
    public static void main(String args[]) {
        try {
            int a = args.length;
            System.out.println(a);
            int b = 10 / a;            // 除零错
            int m[] = {1, 2, 3};
            m[3] = 12;  // 越界错
        } catch (Exception e) {
            System.out.println("Exception");
        } catch (ArithmeticException e) {
            System.out.println(e.toString());
        } catch (ArrayIndexOutOfBoundException e) {
            System.out.println(e.toString());
        }
    }
}
```

参考答案：程序代码错误，功能就是输出程序运行前输入参数的个数， 并捕获相应的异常。存在的问题： Exception 会捕获所有的异常， 后面使用的 ArithmeticExceptioin 和 ArrayIndexOutOfBoundException 就不会执行． 将 Exception 放到最后捕捉就行。

9 、列举一个程序示例，明确体现 Java 面向对象编程中典型的封装、继承和多态特点。

10 、最大公约数，最小公倍数的算法

本周课后作业

第 10 讲 ppt 上的示例程序代码编写（与调试）一遍，代码发给 2230652597@qq.com

编程作业

- 1. 编写程序，要求从键盘输入一个 double 型的圆的半径，计算并输出其面积。测试当输入的数据不是 double 型数据（如字符串“ abc ”）会抛出什么异常？试用异常处理方法修改程序。
- 2. 编写程序，定义一个 static 方法 methodA() ，令其声明抛出一个 IOException 异常，再定义另一个 static 方法 methodB() ，在该方法中调用 methodA() 方法，在 main() 方法中调用 methodB() 方法。试编译该类，看编译器会报告什么？对于这种情况应如何处理？由此可得到什么结论？