

增强软件的鲁棒性：Java异常处理

主讲老师：申雪萍



2021/5/10

Xueping Shen



北京航空航天大学
COLLEGE OF SOFTWARE
BEIHANG UNIVERSITY 软件学院

- 异常概述
- JAVA异常类/异常的层次结构
- 异常处理机制
- 使用 try...catch...finally 处理异常
- 声明抛出异常 (throws)
- 人工抛出异常(throw)
- 创建用户自定义异常类
- 案例分析

讨论并回答问题

- **模拟场景：** 日常生活中，我们经常会去超市去购买物品，结算时使用银行卡进行刷卡款，会出现各种奇怪的问题。这些问题与Java异常情况有很多相似之处。我们来动动脑筋，分析并指出在超市刷卡付费过程中有可能会出现的异常

- 我对去超市购物时发生的异常总结了一下，大概有以下几种，不足的请同学们补充：
- 1、卡不能被识别异常（卡的类型错误、卡被注销、过期都有可能产生这个异常）；
- 2、网络异常
- 3、余额不足异常
- 密码错误一般不当做程序的异常来处理。

示例代码

```
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("the program is end!");  
    }  
}
```

Exception in thread "main" Hello world!

No, I mean it!

HELLO WORLD!!

java.lang.ArrayIndexOutOfBoundsException: 3

at buaa.com.exceptionEx.HelloWorld.main(HelloWorld.java:16)



示例代码

```
class ExceptionTest{  
    public static void main(String[] args){  
        int a=0;  
        System.out.println(5/a);  
    }  
}
```

Exception in thread "main" java.lang.ArithmeticException: / by zero
at buaa.com.exceptionEx.ExceptionTest.main(ExceptionTest.java:8)

示例代码

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

Exception in thread "main" [java.lang.NullPointerException](#)
at buaa.com.exceptionEx.NullRef.main([NullRef.java:8](#))

所以当

1. 数组元素下标越界;
2. 操作数超出预订范围, 例如, 除数为0;
3. 网络连接中断;
4. 想打开的文件不存在;
5. 在载入或链接Java程序时出错;
6. 超出了某些资源限制, 例如使用了太多的内存。
7.

都会出现异常。

编程时，经常遇到三类错误

三类 错误

编译错误(Compilation error)

逻辑错误(logic error)

运行时错误(runtime error)
在程序运行过程中如果发生了一个不可能执行的操作，就会出现运行时错误。

问题:

- 你能写出你最常见到的5种异常的类型吗？
 - 当学完本次课之后，希望你能对JAVA异常做出总结

- 一. 编译系统检查出来的语法错误，导致程序运行结果不正确的逻辑错误，都不属于异常的范围。
- 二. 异常是一个可以正确运行的程序在运行中可能发生的错误。

异常特点

- 一. 偶然性：程序运行中，异常并不总是会发生。
- 二. 可预见性：异常的存在和出现是可以预见的。
- 三. 严重性：一旦异常发生，程序可能终止，或者运行的结果不可预知。

什么是异常处理(Exception Handling)?

- 一. 程序设计的要求之一就是程序的健壮性。希望程序在运行时能够不出或者少出问题。但是，在程序的实际运行时，总会有一些因素会导致程序不能正常运行。
例如文件没找到，网络错误，非法的参数，数组越界，空指针等。
- 二. **异常处理 (Exception Handling) :**
就是要提出或者是研究一种机制，能够较好的处理程序不能正常运行的问题。

- 什么是异常?
 - 异常实际上是程序中错误导致中断了正常的指令流的一种事件。

没有处理错误的程序:

```
read-file {  
    openTheFile;  
    determine its size;  
    allocate that much memory;  
    closeTheFile;  
}
```

以常规方法处理错误

```
1. openFiles;
2.  if (theFilesOpen) {
3.     determine the length of the file;
4.     if (gotTheFileLength){
5.         allocate that much memory;
6.         if (gotEnoughMemory) {
7.             read the file into memory;
8.             if (readFailed) errorCode=-1;
9.             else errorCode=-2;
10.        }else errorCode=-3;
11.    }else errorCode=-4 ;
12. }else errorCode=-5;
```

常规方法处理错误的问题

- 一．开发人员的大部分精力花在出错处理。
- 二．思维的局限性决定：只把能够想到的错误考虑到,对以外的情况无法处理。
- 三．程序可读性差。
- 四．出错返回信息量太少。

用异常的形式处理错误

```
read-File;  
{ try {  
    openTheFile;  
    determine its size;  
    allocate that much memory;  
    closeTheFile;  
} catch(fileopenFailed) { dosomething; }  
  catch(sizeDetermineFailed) {dosomething;}  
  catch(memoryAllocateFailed){ dosomething;}  
  catch(readFailed){ dosomething;}  
  catch(fileCloseFailed) { dosomething; }  
}
```

异常的优点

- 一．把错误代码从常规代码中分离出来
- 二．按错误类型和错误差别分组
- 三．系统提供了对于一些无法预测的错误的捕获和处理
- 四．克服了传统方法的错误信息有限的问题

- 一. 运行时错误会引起异常(Exception),没有异常处理代码的程序可能非正常结束,引起严重问题。(银行问题)
- 二. Java给程序员提供了处理运行时错误的功能,有了这种称为异常处理(exception handling)的功能,就能开发用于重要计算的稳定程序。

加了异常处理的示例

```
public class HelloWorldE {  
    public static void main(String args[]) {  
        int i = 0;  
        String greetings[] = { "Hello world!", "No, I mean it!",  
                                "HELLO WORLD!!" };  
  
        try {  
            while (i < 4) {  
                System.out.println(greetings[i]);  
                i++;  
            }  
        } catch (Exception e) {  
            System.out.println(e.toString());  
            System.out.println(e.getMessage());  
        }  
        System.out.println("the program is end!");  
    }  
}
```

输出结果

```
Hello world!  
No, I mean it!  
HELLO WORLD!!  
java.lang.ArrayIndexOutOfBoundsException: 3  
3  
the program is end!
```

加了异常处理的示例

```
class ExceptionTestE {  
    public static void main(String[] args) {  
        try {  
            int a = 0;  
            System.out.println(5 / a);  
        } catch (ArithmeticException e) {  
            System.out.println(e.toString());  
            System.out.println(e.getMessage());  
        }  
        System.out.println("the program is finished");  
    }  
}  
java.lang.ArithmeticException: / by zero  
/ by zero  
the program is finished
```

加了异常处理的示例

```
public static void main(String[] args) {  
    NullRef t = new NullRef();  
    t = null;  
    try {  
        System.out.println(t.i);  
    } catch (NullPointerException e) {  
        System.out.println(e.toString());  
        System.out.println(e.getMessage());  
    }  
    System.out.println("the program is end!");  
}  
}
```

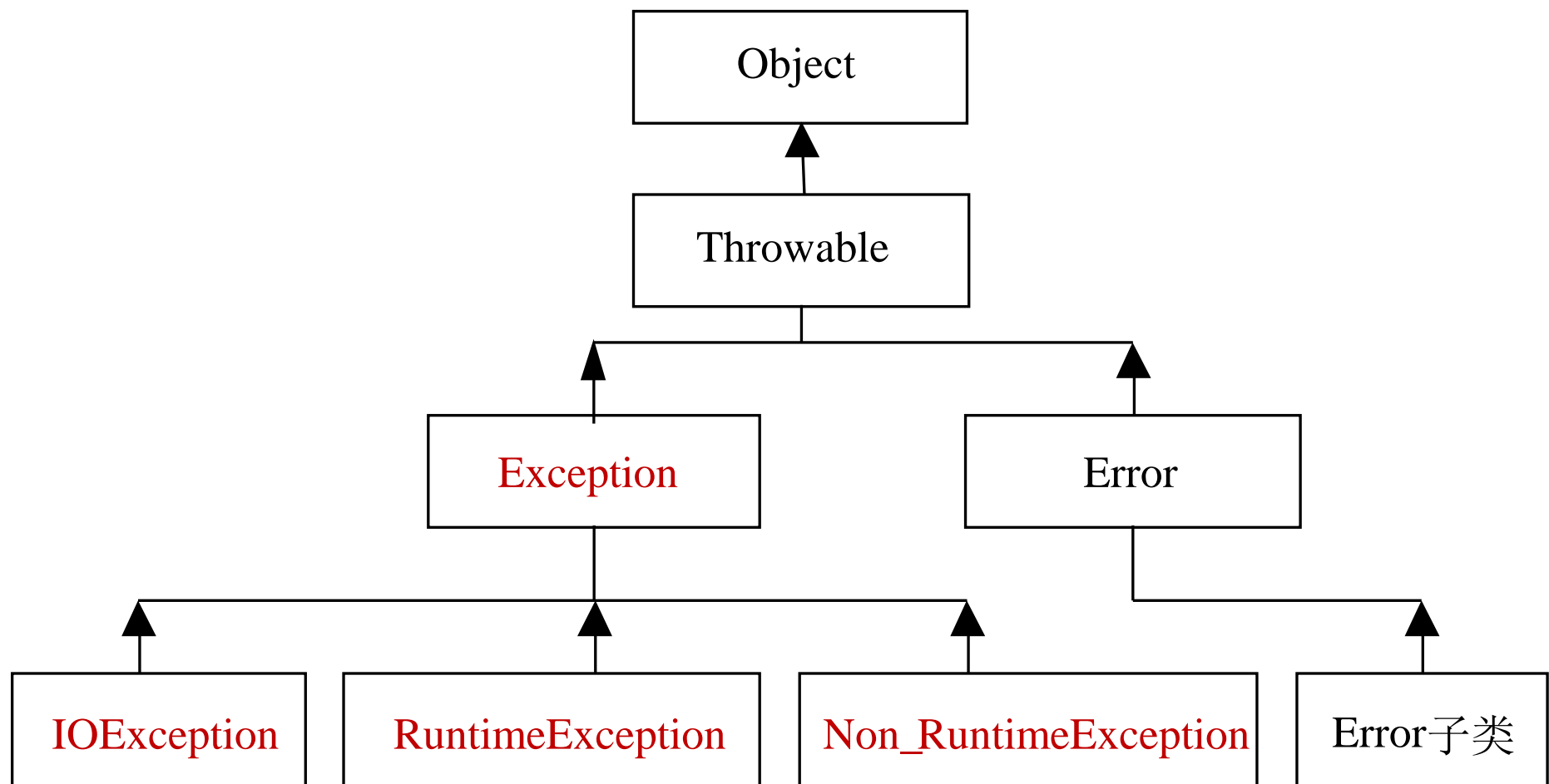
java.lang.NullPointerException
null
the program is end!

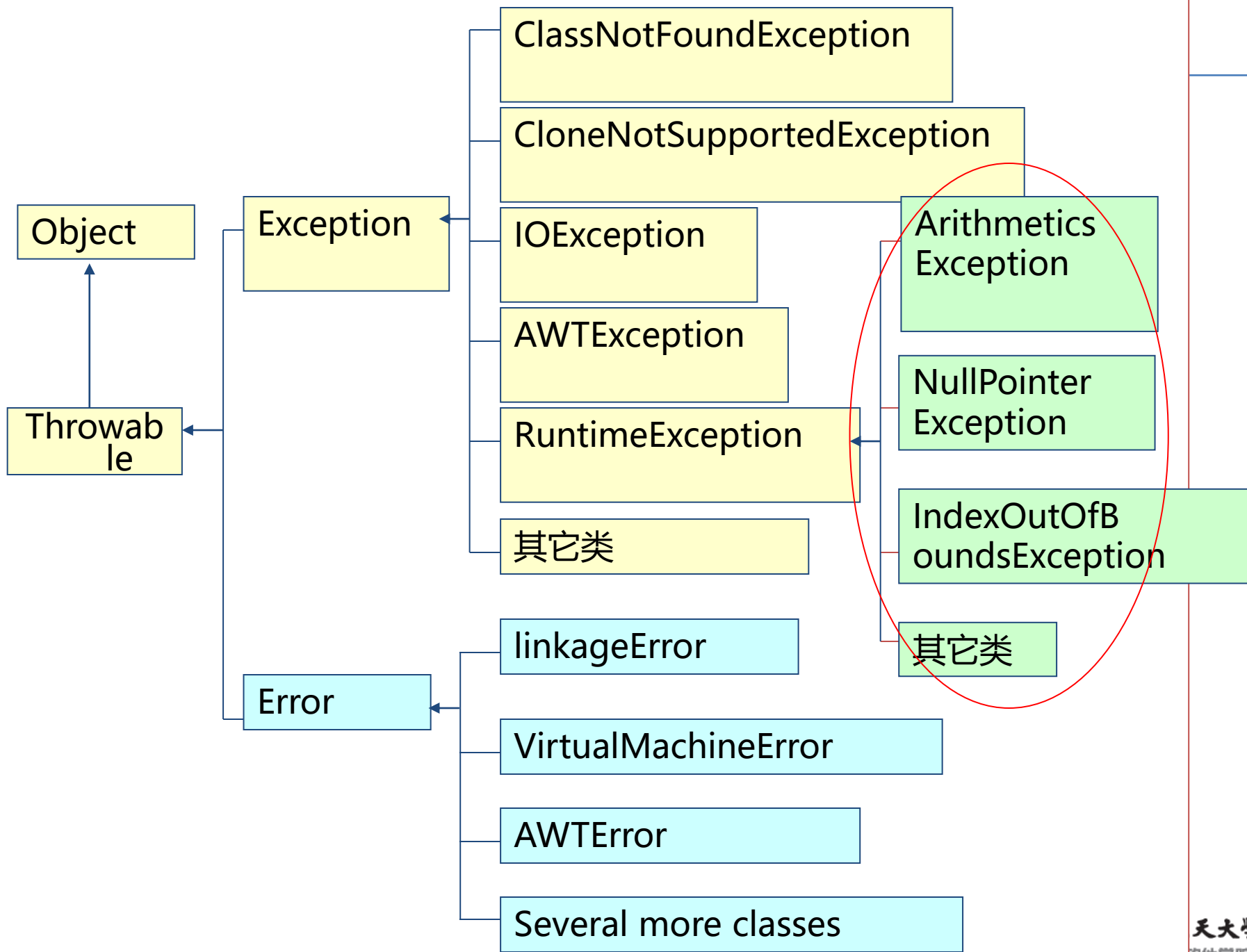
异常的重要性在于：

- 异常的重要性在于程序不但能发现异常，还能处理异常，使程序正常退出。
- 增强了程序的鲁棒性

- 异常概述
- **JAVA异常类/异常的层次结构**
- 异常处理机制
- 使用 try...catch...finally 处理异常
- 声明抛出异常 (throws)
- 人工抛出异常(throw)
- 创建用户自定义异常类

异常类的结构





异常的分类

- 一. Throwable是所有异常类的父类，它是Object的直接子类。是类库java.lang包中的一个类。
- 二. Exception类继承自Throwable类。所有的Throwable类的子孙类所产生的对象都是例外。
- 三. Error:由Java虚拟机生成并抛出,Java程序不做处理。

异常的分类

四. Runtime Exception:由系统检测, 用户的Java 程序可不做处理,系统将它们交给缺省的异常处理程序。

五. **非Runtime Exception:Java编译器要求Java程序必须捕获或声明所有的非运行时异常。**

六. **throw:用户自己产生异常。**

常见的异常

1. ArithmeticException
2. ArrayIndexOutOfBoundsException
3. ArrayStoreException
4. IOException
5. FileNotFoundException
6. NullPointerException
7. MalformedURLException
8. NumberFormatException
9. OutOfMemoryException

异常检查

```
class ExceptionTest1{  
    public static void main(String[] args){  
        int a=0;  
        System.out.println(5/a);  
        System.out.println  
            ("Execution continues");  
    }  
}
```

编译没有语法错误

异常检查

去掉声明异常，
编译有语法错误产生

```
import java.io.*;  
public class ThrowsTest5{  
    public static void main(String args[])  
        throws IOException{  
        FileInputStream fis  
        =new FileInputStream("a3.txt");  
    }  
}
```


异常分类

异常
(按编译
时是否受
检来分)

非受检异常：包括
RuntimeException及其子类、
Error及其子类。

受检异常：除了非受检
异常之外的异常（即其他的异常
类则是可检测的类）

编译器对非受检异常类不进行检查。

非受检异常(unchecked exception)

- 一. 非受检异常(unchecked exception): 这些异常只能在程序执行时被检测到, 不能在编译时被检测到。
- 二. 非受检异常主要包括RuntimeException及其子类。
- 三. 程序对这类异常可不做处理, 交由系统处理;

受检异常(checked exception)

- 一. 受检异常(checked exception): 这些异常在编译时就能被java编译器所检测到异常。
- 二. 除了RuntimeException及其子类以外的其他Exception的子类都是受检异常, 这些异常类是编译时可检测的异常。
- 三. 必须采用声明异常或者try、catch方式处理异常。

- 异常概述
- JAVA异常类/异常的层次结构
- **异常处理机制**
- 使用 try...catch...finally 处理异常
- 声明抛出异常 (throws)
- 人工抛出异常(throw)
- 创建用户自定义异常类

Java 异常处理机制

- Java提供的是异常处理的**抓抛模型**。
- Java程序的执行过程中如出现异常，会自动生成一个**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**。

Java 异常处理机制

- 如果一个方法内抛出异常，该异常会被抛到调用方法中。如果异常没有在调用方法中处理，它继续被抛给这个调用方法的调用者。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**。
- 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。

Java 异常处理机制

- Java语言按照面向对象的思想来处理异常：
 - 把各种不同类型的异常情况进行分类，用Java类来表示异常情况，这种类被称为异常类。
 - 用throws语句在方法声明处声明抛出特定异常，用throw语句在方法中抛出具体的异常
 - 用try-catch语句来捕获并处理异常。

- 异常概述
- JAVA异常类/异常的层次结构
- 异常处理机制
- **使用 try...catch...finally 处理异常**
- **声明抛出异常 (throws)**
- **人工抛出异常(throw)**
- 创建用户自定义异常类

处理异常的两种办法

- 如果方法中的代码块可能抛出异常，有两种处理办法：
 - （1）在当前方法中通过try-catch语句捕获并处理异常，例如：
 - （2）在方法的声明处通过throws语句声明抛出异常，例如：

捕获并处理异常

try...catch...finally
语句:

异常处理的两种方式

Throws语句:

声明异常

在当前方法中通过try-catch语句捕获并处理异常

```
public void methodA(int money){  
    try{  
        //以下代码可能会抛出SpecialException  
        if(--money<=0)  
            throw new SpecialException("Out of money");  
    }catch(SpecialException e){  
        处理异常  
    }  
}
```

在方法的声明处通过throws语句声明抛出异常

```
public void methodA(int money) throws SpecialException{  
    //以下代码可能会抛出SpecialException  
    if(--money<=0)  
        throw new SpecialException("Out of money");  
}
```

try与catch语句的语法格式

```
try {           //接受监视的程序块,在此区域内发生
                //的异常,由catch中指定的程序处理;
}
catch (ExceptionType1 e) {
    // 抛出ExceptionType1异常时要执行的代码
}
catch (ExceptionType2 e) {
    // 抛出ExceptionType2异常时要执行的代码
}.....
[finally {
    // //无条件执行的语句
}]
```

try, catch, finally语句

- 在Java语言中使用语句try...catch...finally进行异常处理，程序流程从引起异常的代码转移到最近的try语句的catch子句。
- Finally: 无条件执行的语句，一般用于释放资源

示例代码

```
public class ExceptionTest2 {  
    public static void main(String[] args) {  
        System.out.println("这是一个异常处理的例子\n");  
        try {  
            int i = 10;  
            String[] str = { "I", "Like", "Java" };  
            str[5] = "Hello";  
            i /= 0;  
            // str[5]="Hello";  
        } catch (ArithmeticException e) {  
            System.out.println("异常是: " + e.getMessage());  
        } catch (IndexOutOfBoundsException e) {  
            System.out.println("异常是: " + e.getMessage());  
        } finally {  
            System.out.println("finally 语句被执行!");  
        }  
        System.out.println("Everything is fine!!!");  
    }  
}
```

注意事项

- 一. 同一段程序可能产生不止一种异常。你可以放置多个catch子句来检查每一种异常类型，第一个与异常匹配的catch将会被执行。
- 二. 如果一个异常类和其子类都出现在catch子句中，应把子类放在前面，否则将永远不会到达子类。

finally语句

- 一. 实际应用中，确保一段代码不管发生什么异常都能被执行是必要的，关键字finally就是用来标识这样一段代码的。
- 二. Finally: 无条件执行的语句，一般用于释放资源。即使没有catch子句，finally语句块也会在执行了try语句块后立即执行。

finally 在资源处理时非常有用

```
try {  
    对文件进行处理的程序;  
}catch(IOException e) {  
    //对文件异常进行处理;  
}finally {  
    不论是否发生异常,都关闭文件;  
}
```

finally 在资源处理时非常有用

```
try {  
    对文件进行处理的程序;  
}  
finally {  
    不论是否发生异常,都关闭文件;  
}
```

正确的语句结构

Throws语句-声明异常

- 一. 声明异常：一个方法不处理它产生的异常,而是沿着调用层次向上传递,由调用它的方法来处理这些异常,叫声明异常。
- 二. Throws语句用来表明一个方法可能抛出的各种异常，并说明该方法会抛出但不捕获异常。

Throws语句-声明异常

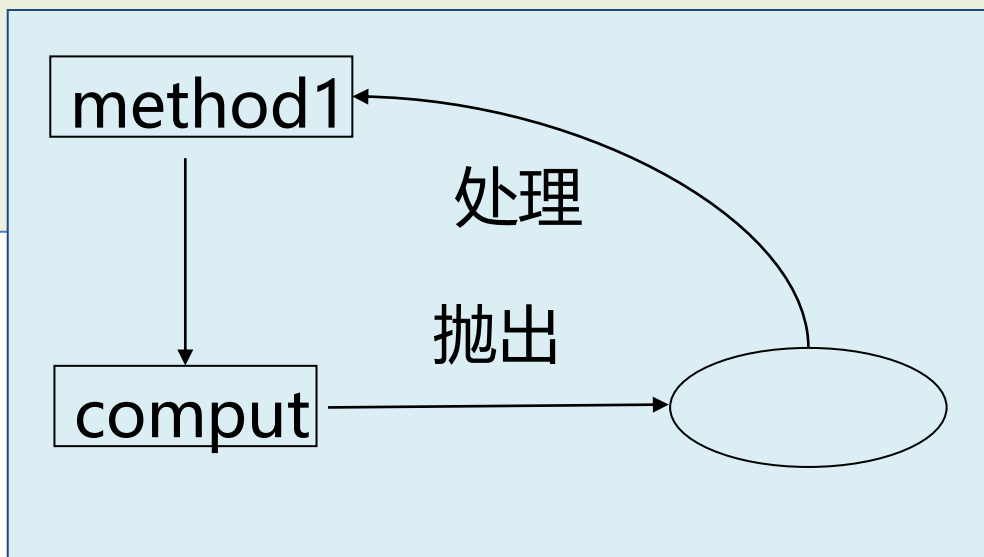
- 声明异常的格式:

<访问权限修饰符><返回值类型><方法名>(参数列表)
throws 异常列表{ }

- 此时, 这种异常交由调用此方法的方法进行处理, 或交由系统进行处理。

由调用方法处理该异常

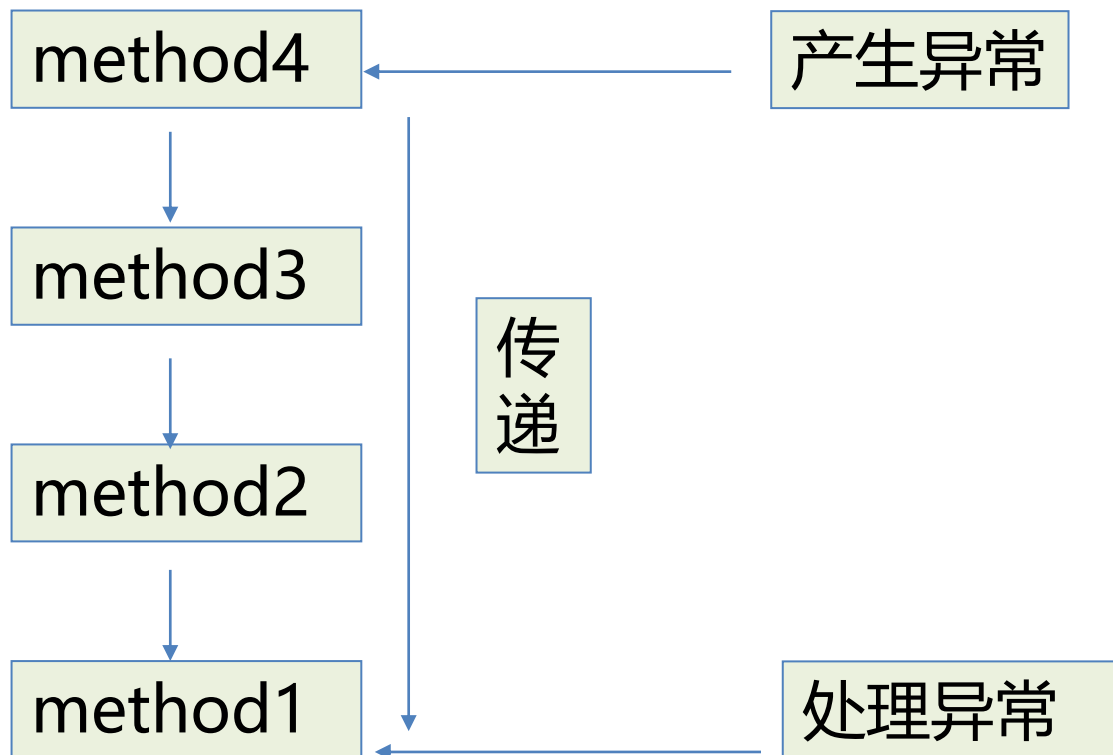
```
public int compute(int x) throws  
ArithmeticException  
{  
    return z=100/x;  
}
```



由调用方法处理该异常

```
public method1()
{   int x;
    try {
        x=System.in.read();
        compute(x);
    }
    catch(IOException ioe)
    {   System.out.println("read error"); }
    catch(ArithmeticException e)
    {   System.out.println("devided by 0"); }
}
```

由调用方法处理该异常



示例代码

```
//由调用方法处理该异常
class ThrowsTest4 {
    //声明异常，但本身并不处理异常
    static void method() throws IllegalAccessException {
        System.out.println("\n在method 中抛出一个异常");
        throw new IllegalAccessException();
    }

    public static void main(String args[])
    {
        try {
            method();
        } catch (IllegalAccessException e) {
            System.out.println("在main 中捕获异常: " + e);
        }
    }
}
```

由系统处理异常

- 如果最终方法也没有处理异常，异常将交给系统处理。

```
import java.io.FileInputStream;

public class ThrowsTest5 {
    public static void main(String args[]) 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();
    }
}
```

看程序输出结果

```
public class MainCatcher{  
    public void methodA(int money)throws SpecialException{  
        if(--money<=0) throw new SpecialException("Out of money");  
        System.out.println("methodA");  
    }  
    public void methodB(int money) throws SpecialException{  
        methodA(money);  
        System.out.println("methodB");  
    }  
    public static void main(String args[]){  
        try{  
            new MainCatcher().methodB(1);  
            System.out.println("main");  
        }catch(SpecialException e){  
            System.out.println("Wrong");  
        }  
    }  
}
```

该程序的打印结果
为： Wrong。

看程序输出结果

```
public class MainCatcher{  
    public void methodA(int money)throws SpecialException{  
        if(--money<=0) throw new SpecialException("Out of money");  
        System.out.println("methodA");  
    }  
    public void methodB(int money) throws SpecialException{  
        methodA(money);  
        System.out.println("methodB");  
    }  
    public static void main(String args[]){  
        try{  
            new MainCatcher().methodB(2);  
            System.out.println("main");  
        }catch(SpecialException e){  
            System.out.println("Wrong");  
        }  
    }  
}
```

该程序的打印结果
为：
methodA
methodB
main

看程序输出结果

```
public class WithFinally {  
    public void methodA(int money)throws SpecialException{  
        if(--money<=0)  
            throw new SpecialException("Out of money");  
        System.out.println("methodA");  
    }  
    public static void main(String args[]){  
        try{  
            new WithFinally().methodA(1);  
            System.out.println("main");  
        }catch(SpecialException e){  
            System.out.println("Wrong");  
        }finally{  
            System.out.println("Finally");  
        }}}
```

该程序的打印结果为：
Wrong
Finally

看程序输出结果

```
public class WithFinally {  
    public void methodA(int money)throws SpecialException{  
        if(--money<=0)  
            throw new SpecialException("Out of money");  
        System.out.println("methodA");  
    }  
    public static void main(String args[]){  
        try{  
            new WithFinally().methodA(2);  
            System.out.println("main");  
        }catch(SpecialException e){  
            System.out.println("Wrong");  
        }finally{  
            System.out.println("Finally");  
        }}}
```

该程序的打印结果为：
methodA
main
Finally

- 一. 对于程序中需要处理的异常，一般编写try-catch-finally语句捕获并处理;
- 二. 而对于程序中不需要处理的异常，可以使用throws语句在方法中抛出异常交由系统处理。

注意事项

- 重写方法不能抛出比被重写方法范围更大的异常类型

```
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 { //error  
        .....  
    }  
}
```


- 异常概述
- JAVA异常类/异常的层次结构
- 异常处理机制
- 使用 try...catch...finally 处理异常
- 声明抛出异常 (throws)
- 人工抛出异常(throw)
- **创建用户自定义异常类**

JAVA自定义异常

- 一. 不是由Java系统监测到的异常(下标越界, 被0-除等),而是由用户自己定义的异常.
- 二. 用户定义的异常必须由用户自己抛出
`throw new MyException.`

创建自己的异常

- 语句形式:

```
<class><自定义异常名>extends<Exception>  
{  
    ...  
}
```

Throw语句-抛出异常

- 一. 在前面讲述的例子中，异常对象是由Java在运行时由系统抛出的。
- 二. 程序中显示生成异常：抛出异常也可以通过代码来实现，throw语句可以明确的抛出一个异常。
- 三. Throw抛出异常主要用于自定义异常。

Throw语句-抛出异常

一. throw的语句格式为:

<throw> <异常对象>

二. 程序会在throw语句处立即终止, 转向
try...catch 寻找异常处理方法。

看程序输出结果

```
public class YourException extends Exception{
    private int id;
    public YourException() {
        // TODO Auto-generated constructor stub
    }
    public YourException(int id) {
        super();
        this.id = id;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
}
```

看程序输出结果

```
public class ThrowTest {  
    public static void regist(int number)throws YourException{  
        if (number<=0)  
            throw new YourException(number);  
        else  
            System.out.println("登记人数" + number);  
    }  
    public static void manager(){  
        try{  
            regist(100);}  
        catch(YourException e){  
            e.toString(); System.out.println("注册失败");  
        }  
    }  
    public static void main(String[] args) {  
        ThrowTest.manager();  
    }  
}
```

登记人数100

看程序输出结果

```
class MyException extends Exception { //自定义异常
    private int x;
    MyException(int a) {
        x = a;
    }
    public String toString() {
        return "MyException";
    }
}
```


看程序输出结果

```
public class MyExceptionTest {  
    public static void main(String args[]) {  
        try {  
            System.out.println("\n进入监控区，执行可能发生异常的程序段");  
            method(8);  
            method(20);  
            method(6);  
        } catch (MyException e) {  
            System.out.println("\t程序发生异常并在此处进行处理");  
            System.out.println("\t发生的异常为: " + e.toString());  
        }  
        System.out.println("这里可执行其他代码");  
    }  
    static void method(int a) throws MyException {  
        System.out.println("\t此处引用 method (" + a + ")");  
        if (a > 10)  
            throw new MyException(a); // 主动抛出MyException  
        System.out.println("正常返回");  
    }  
}
```

输出结果

进入监控区，执行可能发生异常的程序段

此处引用 `method (8)`

正常返回

此处引用 `method (20)`

程序发生异常并在此处进行处理

发生的异常为：`MyException`

这里可执行其他代码

案例分析（1）

- 重点内容
 - 掌握自定义异常
 - throws语句在方法声明处声明抛出特定异常
 - throw语句在方法中抛出具体的异常
 - 注意判断条件，由开发人员生成异常对象抛出
 - try catch处理异常，并在catch语句中，当条件满足时，抛出新的自定义异常

自定义异常类

```
{/** 表示车子出故障的异常情况 */  
public class CarWrongException extends Exception  
    public CarWrongException(){ }  
    public CarWrongException(String msg){ super(msg);}  
}
```

自定义异常类

/** 表示上班迟到的异常情况 */

public class LateException extends Exception{

private Date arriveTime; //迟到的时间

private String reason; //迟到的原因

public LateException(Date arriveTime,String reason){

 this.arriveTime=arriveTime;

 this.reason=reason;

}

public Date getArriveTime(){return arriveTime;}

public String getReason(){return reason;}

}

抛出异常

```
public class Car{  
    public void run()throws CarWrongException{  
        /*如果车子出故障，就创建一个CarWrongException  
        对象，并将其抛出*/  
        if(车子无法刹车)  
            throw new CarWrongException("车子无法刹车");  
        if(发动机无法启动)  
            throw new CarWrongException("发动机无法启动");  
    }  
}
```

throws语句在方法声明处声明抛出特定异常，
throw语句在方法中抛出具体的异常

处理并抛出异常

```
public class Worker{  
    private Car car;  
    public Worker(Car car){ this.car=car; }  
    public void gotoWork()throws LateException{  
        try{  
            car.run();  
        }catch(CarWrongException e){  
            walk();  
            Date date=new Date(System.currentTimeMillis());  
            String reason=e.getMessage();  
            throw new LateException(date,reason); }  
        }  
    }  
    public void walk(){.....} //步行上班
```

案例分析（2）

- 一. 银行转账时，若取钱数大于余额则作为异常处理(`InsufficientFundsException`)，请定义上述异常类。
- 二. 编写测试类，对自定义异常进行测试。

解决思路

一. 编写自定义异常类

InsufficientFundsException ;

二. 产生异常的条件是取钱时, 余额少于取款额度。

取钱是在withdral () 中定义的动作,程序在此应抛出自定义异常(throw) 。

三. withdral () 声明异常,由上级方法采用try、catch处理异常。

代码:

- ExceptionDemoBank.java



```
class Account {  
    double balance;  
  
    Account(double bal) {  
        balance = bal;  
    }  
  
    public void deposit(double dAmount) {  
        if (dAmount > 0.0) {  
            balance += dAmount;  
        }  
    }  
  
    public void withdrawal(double dAmount) throws InsufficientFundsException {  
        if (balance < dAmount) {  
            throw new InsufficientFundsException(this, dAmount);  
        }  
        balance = balance - dAmount;  
    }  
  
    public String show_balance() {  
        return "The balance is " + (int) balance;  
    }  
}
```



```

class InsufficientFundsException extends Exception {
    private Account excepbank;

    private double excepAmount;

    InsufficientFundsException(Account ba, double dAmount) {
        excepbank = ba;
        excepAmount = dAmount;
    }

    public String excepMesagge() {
        String str = excepbank.show_balance() + " The withdrawal was "
            + excepAmount;
        return str;
    }
}

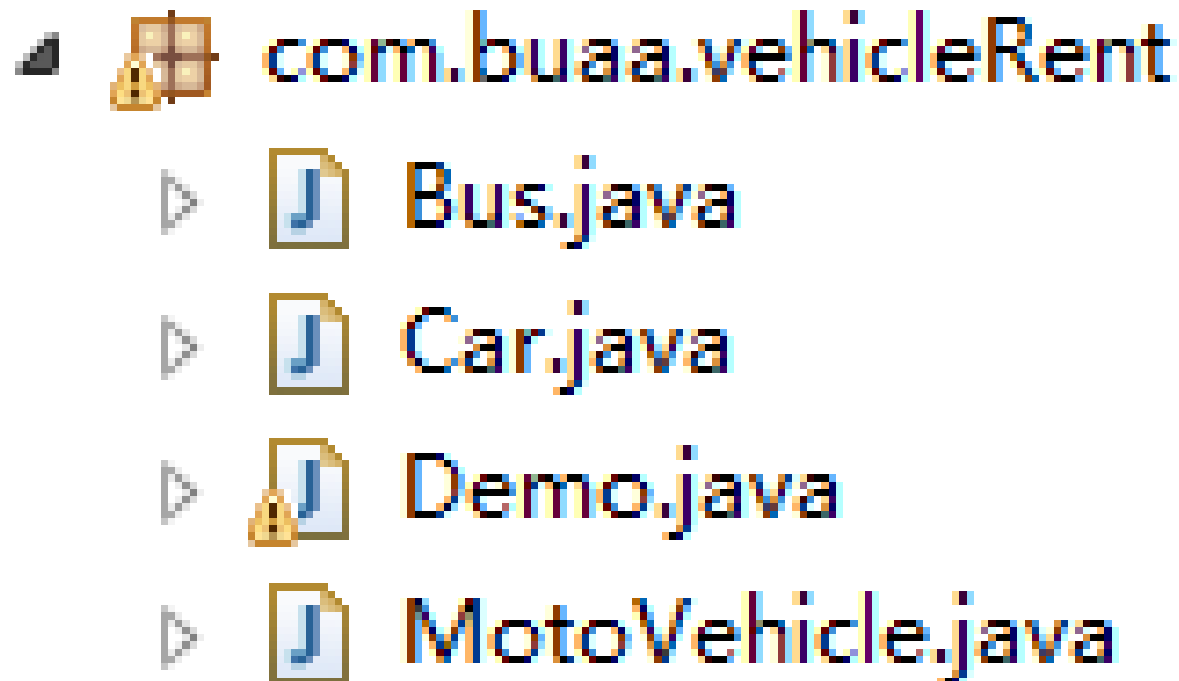
```

```
public class ExceptionDemoBank {  
    public static void main(String args[]) {  
        try {  
            Account ba = new Account(50);  
            ba.withdrawal(20);  
            System.out.println("Withdrawal successful!");  
        } catch (Exception e) {  
            System.out.println(e.toString());  
        }  
        try {  
            Account ba = new Account(50);  
            ba.withdrawal(200);  
            System.out.println("Withdrawal successful!");  
        } catch (InsufficientFundsException e) {  
            System.out.println(e.excepMesagge());  
        }  
    }  
}
```

案例分析（3）

- 继承和关闭继承（final关键字的使用）
- 枚举
- 泛型与集合框架（ArrayList）
- 异常处理

案例分析（3）



选择要租的车种类：

1： 轿车

2： 巴士

0： 退出

1|

选择要租的车型：

1： 别克 商务舱**GL8** 灰色

车牌：**AB66666**

日租金：**600.00**

2： 别克 林荫大道 黑色

车牌：**AB66666**

日租金：**300.00**

3： 宝马 **550i** 白色

车牌：**AB66666**

日租金：**500.00**

0： 退出

2|

您要租多少天？

20

租金： 6000.0

选择要租的车种类：

1： 轿车

2： 巴士

0： 退出

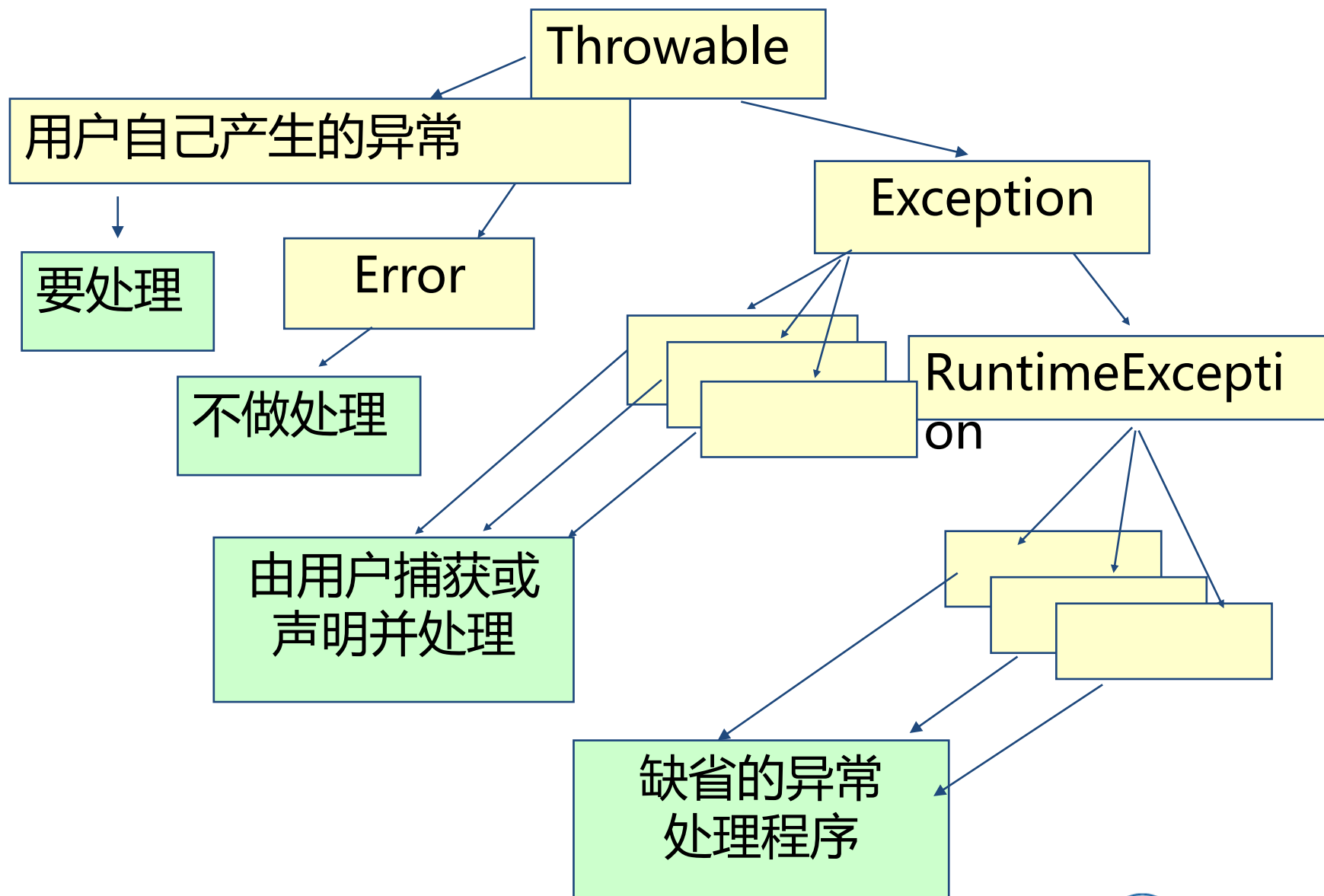
异常处理的原则

- 早处理
- 异常设计时需要分层次
- 只捕获特定异常：
 - 不要试图用一个catch块处理所有异常，也不要尝试把大量代码放在一个try语句中进行监测。这样不容易分析异常发生的位置，及分析异常的具体类型。
- 善用finally语句块
- Java 异常处理是使用 Java语言进行软件开发和测试脚本开发中非常重要的一个方面。**对异常处理的重视会使您开发出的代码更健壮，更稳定。**

小结

- 一. 异常处理机制：抓抛
- 二. try-catch-finally 语句
- 三. Throws声明异常
- 四. Java编译器要求Java程序必须捕获或声明所有的非运行时异常。
- 五. throw:用户自己产生异常。

小结



问题讨论:

- JAVA 语言如何进行异常处理
- 关键字: throws,throw,try,catch,finally分别代表什么意义?
- 在try 块中可以抛出异常吗?
- 请写出你最常见到的5个runtime exception

问题讨论:

- `try {}` 里有一个`return`语句, 那么紧跟在这个`try`后的`finally {}`里的`code`会不会被执行, 什么时候被执行, 在`return`前还是后?
 - 会执行, 在`return`前执行。

示例：看程序输出结果

```
package buaa.com.exceptionEx;
public class Test {
    public static void main(String[] args){
        System.out.println("main中x="+Test.test());
    }
    static int test(){
        int x=1;
        try{
            return x;
        }
        finally{
            ++x;
            System.out.println("finally中x="+x);
        }
    }
}
```

Problems @ Javadoc Declaration Console

<terminated> Test (6) [Java Application] C:\Program Files\Java

finally中x=2

main中x=1

谢谢！

