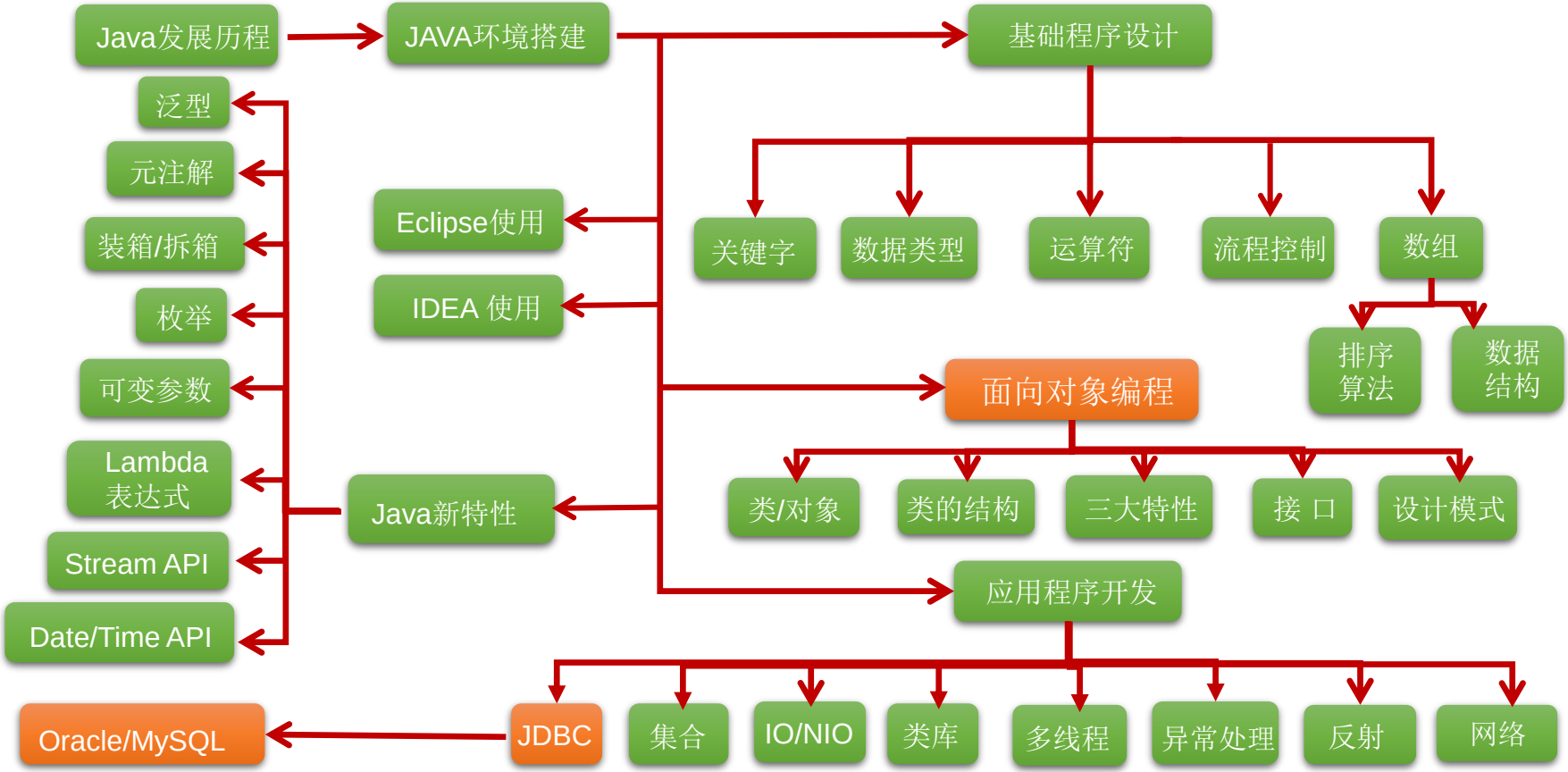
第7章
异常处理

讲师：宋红康
新浪微博：尚硅谷-宋红康

Java基础知识图解



目录

尚硅谷

7-1 异常概述与异常体系结构

在使用计算机语言进行项目开发的过程中，即使程序员把代码写得尽善尽美，在系统的运行过程中仍然会遇到一些问题，因为很多问题不是靠代码能够避免的，比如：客户输入数据的格式，读取文件是否存在，网络是否始终保持通畅等等。

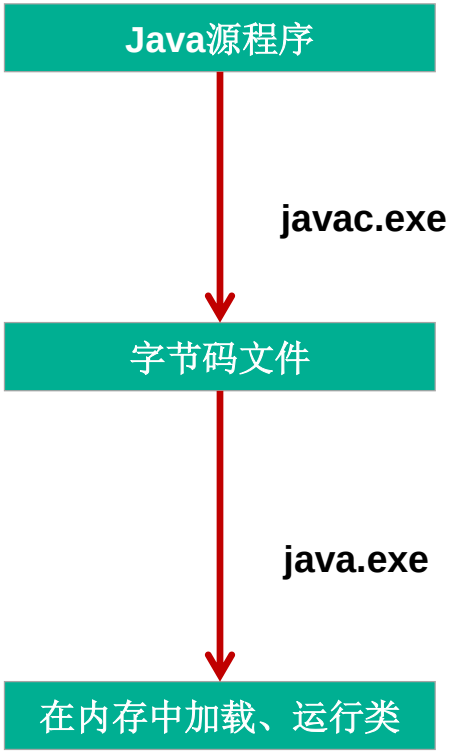
7.1 异常概述与异常体系结构

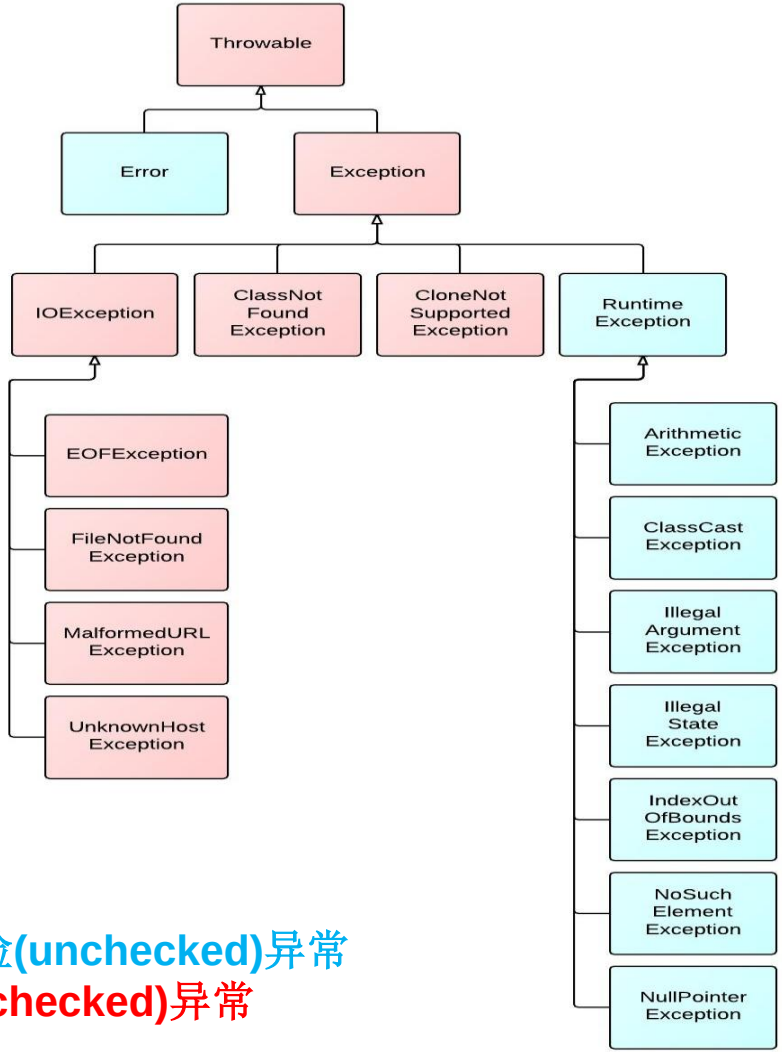
- 异常：在Java语言中，将程序执行中发生的不正常情况称为“异常”。(开发过程中的语法错误和逻辑错误不是异常)
- Java程序在执行过程中所发生的异常事件可分为两类：
  - **Error**：Java虚拟机无法解决的严重问题。如：JVM系统内部错误、资源耗尽等严重情况。比如：StackOverflowError和OOM。一般不编写针对性的代码进行处理。
  - **Exception**: 其它因编程错误或偶然的外在因素导致的一般性问题，可以使用针对性的代码进行处理。例如：
    - ✓空指针访问
    - ✓试图读取不存在的文件
    - ✓网络连接中断
    - ✓数组角标越界

- 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。

- 捕获错误最理想的是在**编译期间**，但有的错误只有在**运行时**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

7.1 异常概述与异常体系结构





蓝色：非受检(unchecked)异常

红色：受检(checked)异常

1.运行时异常

- 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

- 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。

7-2 常见异常

7.2 常见异常

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

7.2 常见异常：ArrayIndexOutOfBoundsException

```
public class IndexOutExp {
    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");
    }
}
```

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

```
lisa
bily
kessy
java.lang.ArrayIndexOutOfBoundsException
        at Test7_1.main(Test7_1.java:5)
Exception in thread "main"
```

7.2 常见异常：NullPointerException

```
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"
```

7.2 常见异常： ArithmeticException

```
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!");
    }
}
```

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"
```

7.2 常见异常：ClassCastException

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

程序Person.java编译正确，运行结果：java Person

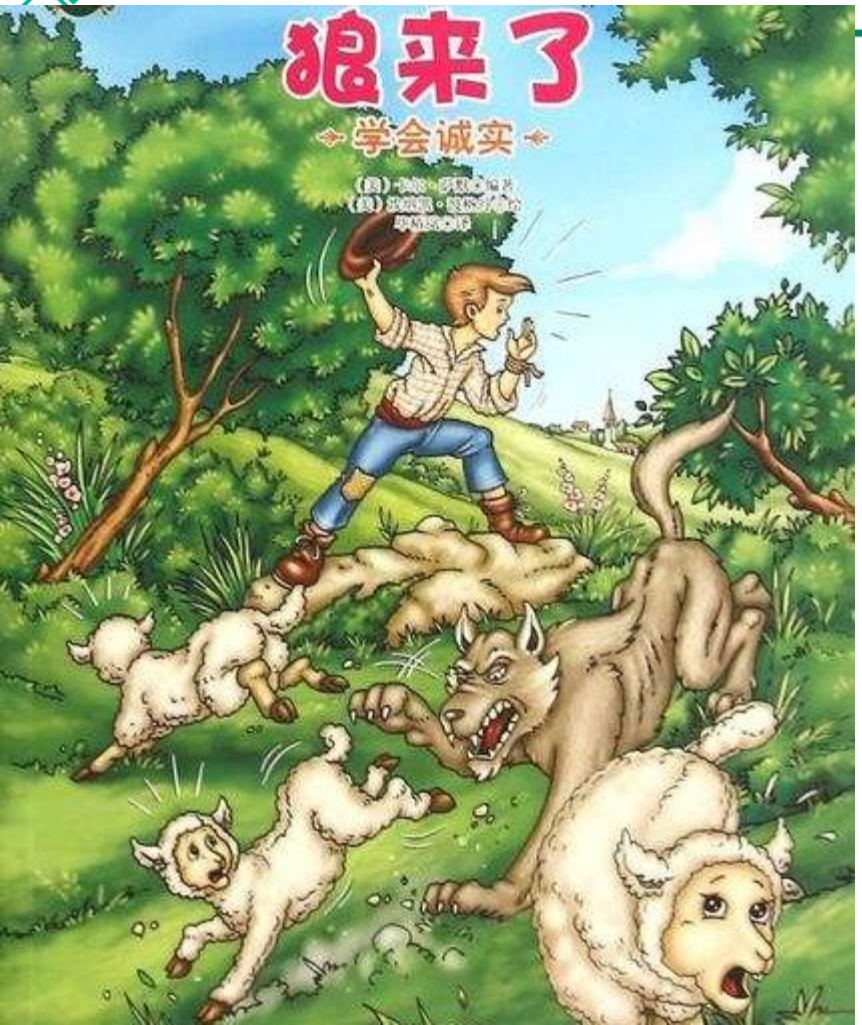
```
java.lang.java.lang.ClassCastException
        at Person.main(Person.java:5)
Exception in thread "main"
```

7-3 异常处理机制一：try-catch-finally

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if-else分支会导致程序的代码加长、臃肿，可读性差。因此采用异常处理机制。

Java异常处理

Java采用的异常处理机制，是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、优雅，并易于维护。



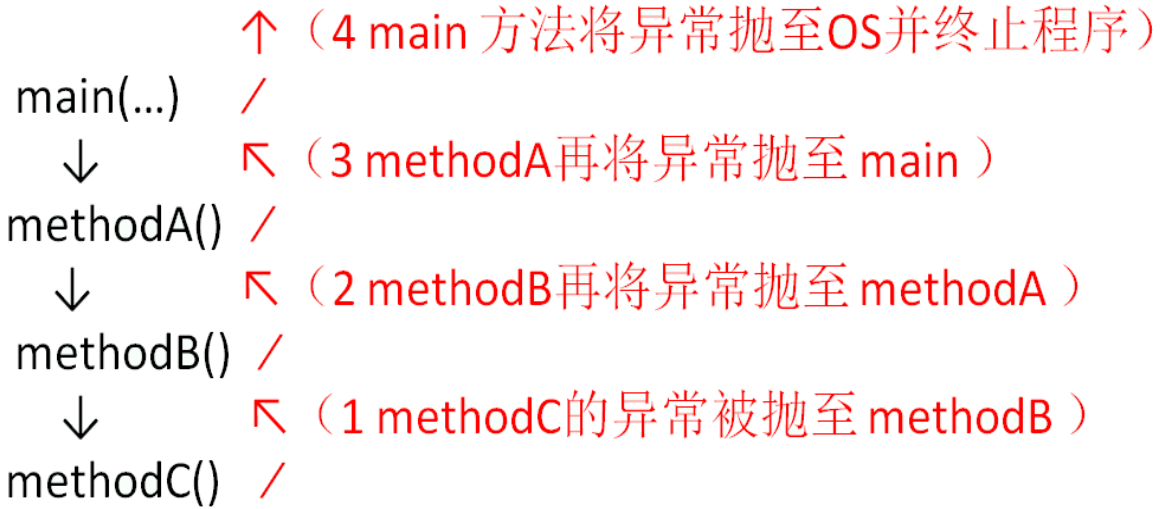
Java异常处理的方式：

方式一：try-catch-finally

方式二：throws + 异常类型

- 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。
- 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
  - 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

异常的抛出机制



为保证程序正常执行，代码必须对可能出现的异常进行处理。

7.3 异常处理机制一

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

异常处理是通过try-catch-finally语句实现的。

```
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
} ]
```

- **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。

●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语句和catch语句是任选的

7.3 异常处理机制一

➢捕获SomeException2时：

➢没有捕获到异常时：

7.3 异常处理机制一：举例

```
public class IndexOutExp {
    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");
        }
        System.out.println("\nthis is the end");
    }
}
```

程序IndexOutExp.java运行结果：java IndexOutExp

lisa
bily
kessy
index err
this is the end

7.3 异常处理机制一：举例

```
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("program ends ok!");
    }
}
```

程序DivideZero1运行结果：java DivideZero1

divide by zero error!

program ends ok!

练习1

编写一个类ExceptionTest，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。

体 会

- 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

- 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，否则编译错误。也就是说，我们必须处理编译时异常，将异常进行捕捉，转化为运行时异常

非运行是异常：SQLException,IOException,FileNotFoundException,ReflectiveOperationException(反射操作异常)

IOException异常处理举例(1)

```
import java.io.*;

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

输入一遍代码就知道了，这里如果不加上trycatch的话，周围会说未处理异常：java.io.IOException

IOException异常处理举例(2)

```
import java.io.*;
public class IOExp {
    public static void main(String[] args) {
        try {
            FileInputStream in = new FileInputStream("atguigushk.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!");
        }
    }
}
```

练习2
捕获和处理IOException异常

编译、运行应用程序IOExp.java，体会Java语言中异常的捕获和处理机制。

相关知识：FileInputStream类的成员方法read()的功能是每次从相应的(本地为ASCII码编码格式)文件中读取一个字节，并转换成0~255之间的int型整数返回，到达文件末尾时则返回-1。

7-4 异常处理机制二：throws

7.4 异常处理机制二：声明抛出异常

- 声明抛出异常是Java中处理异常的第二种方式
  - 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);
        ……
    }
```

7.4 异常处理机制二：声明抛出异常


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

7.4 异常处理机制二：声明抛出异常

```
public InputStream getInputStream(String filePath)
        throws FileNotFoundException {
    InputStream in = new FileInputStream(filePath);
    return in;
}

public void testGetInputStream()
        throws FileNotFoundException{
    getInputStream("aaa.txt");
}
```

在方法头部声明抛出

有可能找不到**filePath**指定的文件，所以抛出**FileNotFoundException**异常

继续向上抛，交给调用者处理

调用了一个声明抛出非运行时异常的方法

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。

```
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
            ……
        } }
```

7-5 手动抛出异常

- 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，也可根据需要使用人工创建并抛出。
  - 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;
```

  - 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");
```

7-6 用户自定义异常类

7.6 用户自定义异常类

- 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。
- 自定义异常需要提供serialVersionUID
- 自定义的异常通过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。

7.6 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 = 13465653435L;
    private int idnumber;

    public MyException(String message, int id) {
        super(message);
        this.idnumber = id;
    }

    public int getId() {
        return idnumber;
    }
}
```

7.6 用户自定义异常类


```
public class MyExpTest {
    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[]) {
        MyExpTest t = new MyExpTest();
        t.manager();
    }
}
```

练习3：判断程序的输出结果

```
public class ReturnExceptionDemo {
    static void methodA() {
        try {
            System.out.println("进入方法A");
            throw new RuntimeException("制造异常");
        }finally {
            System.out.println("用A方法的finally");
        }
    }

    static void methodB() {
        try {
            System.out.println("进入方法B");
            return;
        } finally {
            System.out.println("调用B方法的finally");
        }
    }

    public static void main(String[] args) {
        try {
            methodA();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        methodB();
    }
}
```

练习4

- 编写应用程序EcmDef.java，接收命令行的两个参数，要求不能输入负数，计算两数相除。

  对数据类型不一致(NumberFormatException)、缺少命令行参数(ArrayIndexOutOfBoundsException、

  除0(ArithmeticException)及输入负数(EcDef 自定义的异常)进行异常处理。

- 提示：

  (1)在主类(EcmDef)中定义异常方法(ecm)完成两数相除功能。

  (2)在main()方法中使用异常处理语句进行异常处理。

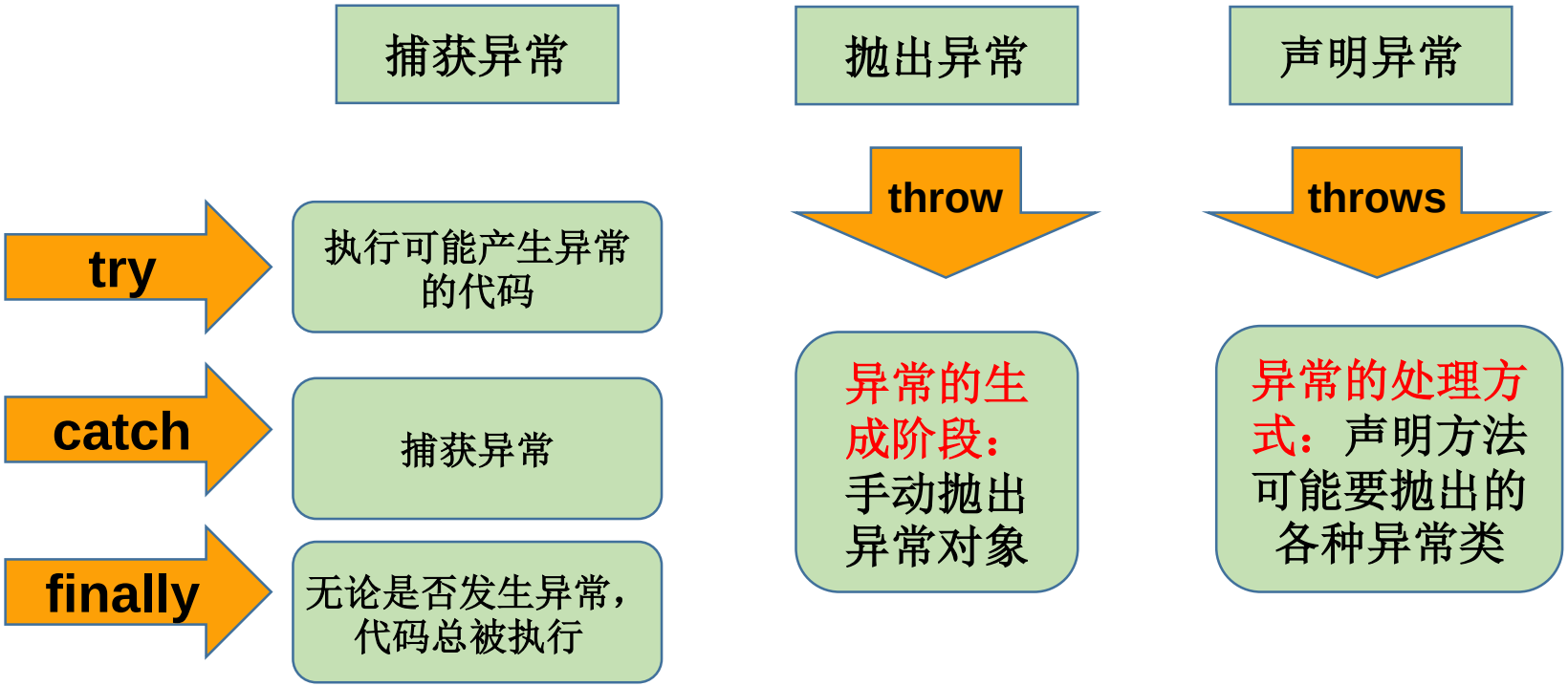
  (3)在程序中，自定义对应输入负数的异常类(EcDef)。

  (4)运行时接受参数 java EcmDef 20 10　//args[0]=“20” args[1]=“10”

  (5)Interger类的static方法parseInt(String s)将s转换成对应的int值。

  如：int a=Interger.parseInt(“314”);　　//a=314;

总结：异常处理5个关键字



例如：上游排污，下游治污

一首小悟结束异常处理

世界上最遥远的距离，是我在if里你在else里，似乎一直相伴又永远分离；

世界上最痴心的等待，是我当case你是switch，或许永远都选不上自己；

世界上最真情的相依，是你在try我在catch。无论你发神马脾气，我都默默承受，静静处理。到那时，再来期待我们的finally。


尚硅谷
让天下没有难学的的技术