# 面向对象

### 面向对象的基本概念

#### 属性

对象具有的各种特征

##### 方法

对象的行为

##### 对象

用来描述客观事物的一个实体，由一组属性和方法构成

##### 类

具有相同属性和方法的一组对象的集合

类是对象的抽象，对象是类的具体

可以认为类是一个模板，通过给模板加入数据，就生成了一个一个对象。

##### 创建对象

使用new关键字，创建对象，这个过程叫实例化

Student stu=new Student();

如果初始化对象，没有赋值，其默认值规则如下：

\* 数值类型（byte,short,int,long） 0

\* 数值（flaot,double) 0.0

\* boolean类型 false

\* char类型 /u0000

\* 引用类型 null

##### 引用对象的属性和行为

使用“.”stu.name=”蓝翔”；

#### 成员的方法

###### 类中包含部分

类成员主要包含两部分：成员变量和成员方法，内部类。

###### 成员变量

定义在类中，方法外的变量。

如果不赋值，有默认的值：

\* 数值类型（byte,short,int,long） 0

\* 数值（flaot,double) 0.0

\* boolean类型 false

\* char类型 /u0000

\*引用类型 null

###### 成员方法

定义在类中的方法。

public void work(){

System.out.println(“北大青鸟”);

}

###### 形参和实参

形参：方法中定义的参数

实参：调用方法时传入的参数

###### 方法重载

在一个类中定义多种同名的方法，但要求每个方法具有不同的参数类型或参数个数。

## 方法重载的特点

·在同一个类中

·方法名相同

·参数的个数或类型不同

·方法的返回值不能作为判断方法之间是否构成重载的依据

·方法的修饰符(public/private...) 不能作为判断方法之间构成重载的依据

###### 构造方法

\* 1、构造方法没有返回值，也不能用void表示

\* 2、构造方法名与类名一致

#### 成员变量

###### 成员变量的作用域

类中的属性，也就是直接在类中定义的变量，定义阿紫方法的外部

###### 局部变量的作用域

局部变量就是定义在方法中的变量。

###### 成员变量和局部变量的区别

1. 作用域不同
2. 初始值不同
3. 在同一个方法中，不允许有同名的局部变量
4. 局部变量可以和成员变量同名

###### 数据类型

变量的类型分为两种：八大基本类型和引用类型

引用类型：类，数组，接口

###### This的用法

This:代指当前对象

一个对象的默认引用。每个实例方法内部都有一个this引用变量，指向调用 这个方法的对象

This的三种使用场景

\* 使用this调用成员变量 ：

解决成员变量与局部变量名冲突问题。

在方法中只能涉及成员变量，都应该加this。

只不过当名字不冲突的话，这个this可不写。

\* 使用this调用成员方法，this可以省略

使用this调用重载的构造方法。

注意：只能在构造方法中使用。且必须是构造方法的第一条语句

#### 封装

面向对象的三大特征之一

###### package

package 必须是java源文件的第一条非注释语句。且一个源文件只\* 编码规范：

1、一个唯一的包名前缀通常是全部小写的ASCII字母(abcd.....)。并且是一个顶级域名com,edu,gov,net,org。通常使用组织的网络域名逆序。

比如：域名为：javagroup.net

包声明： package net.javagroup.mypackage

Import

实例化Scanner,如下写法

Java.util.Scanner sca = new java.util.Scanner(System.in);

先导入——import java.util.Scanner;

Scanner input=new Scanner(System.in);

###### 使用访问修饰符

类和类成员的访问限制：

同一包中 非同一包中

Public: 可以使用 可以使用

默认修饰符（不写） 可以使用 不可以使用

同一类中 同一包中 子类中 其他地方

Private 可以使用 不可以使用 不可以使用 不可以使用

默认修修饰符 可以使用 可以使用 不可以使用 不可以使用

Protected 可以使用 可以使用 可以使用 不可以使用

Public 可以使用 可以使用 可以使用 可以使用

###### 用static关键字修饰

Static修饰属性

称为静态变量/类变量。可以直接使用类名调用静态变量。

被static修饰的变量

1、在内存中的静态区只有一个拷贝

2、在类部，可以再任何方法内直接访问静态变量。

3、在其他类中，可以直接通过类名访问

Static修饰方法

称为静态方法/类方法

1. 在静态方法中不能直接访问实例变量和实例方法
2. 在静态方法中可以调用静态方法和静态变量

Static块

static块属于类的代码块，在类加载时执行的代码块，只执行一次，可以用来在软件中加载静态资源。

staitc块常常用于加载静态资源（图片，音频，视频.....

1. 如果有多个静态块，按顺序加载。
2. 每个静态代码块中只会执行一次。

###### Final的用法

Final修饰成员变量和局部变量

Final修饰成员变量

1. 意为不可改变
2. 两种方法的初始化

声明时同时初始化

构造方法中初始化

Final修饰局部变量

1. 意为不可改变
2. 在使用之前初始化即可

Final修饰方法

不可被子类重写。其意义在于：防止子类在定义新方法时，不经意将方法重写掉。

Final修饰类：意为不可被继承

意义在于：保护类不被继承，控制滥用继承对系统造成危害。

例如：String类就是被final修饰

###### Static final修饰的成员变量

1. 不能改变，并且唯一
2. 必须声明的同事初始化
3. 常量的名称建议全部大写
4. 在编译时，被直接替换为具体的值。效率高
5. 可以用类名去调用常量
6. 在方法区里面，只有一份，不可更改

### 继承和多态

###### 继承的基本概念

继承是面向对象的三大特征之一，继承可以解决变成中代码冗余的问题，是实现代码重要手段之一。

在java中，继承通过extends关键字实现。

在java中，子类可以从父类中继承以下内容

1、可以继承publiche protected修饰的属性和方法，不论子类和父类是否在同一个包里

2、可以继承默认访问修饰符的属性和方法，但是子类和服必须在同一个包里

3、无法继承父类的构造方法

###### 继承的应用

Super的用法

1. 访问父类属性
2. 访问父类方法
3. 访问父类构造方法

注意

1. 使用super关键字，super代表父类
2. 在子类构造方法中调用且必须放在第一句
3. 不可访问父类中定位为private的属性和方法

继承条件下构造方法的调用规则

1. 子类构造方法无法通过super显示调用父类的有参构造方法，

也没有通过this显示调用自身其他构造方法。系统默认调用父类的午参构造方法

1. 子类构造方法通过super显示调用父类的有参构造方法，

则执行父类相应构造方法，而不执行父类的无参构造方法。

###### 重写

子类根据需求对父类继承的方法进行重新编写

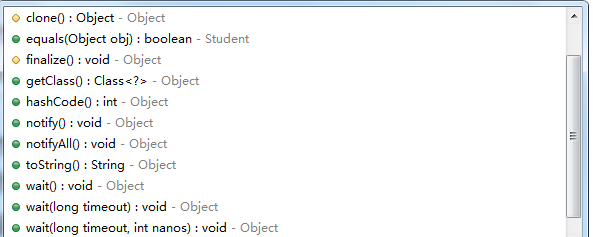
方法重写的规则

1. 方法名相同，参数列表相同
2. 返回值类型要么相同，要么是其子类
3. 访问权限不能严于父类
4. 父类有静态方法，子类重写也必须是静态方法
5. 父类私有的（private）方法不能被被子类重写
6. 子类不能抛出比父类方法更多的异常

Object类

如果没有自定义继承，则默认继承顶级父类object类

Object类提供的方法



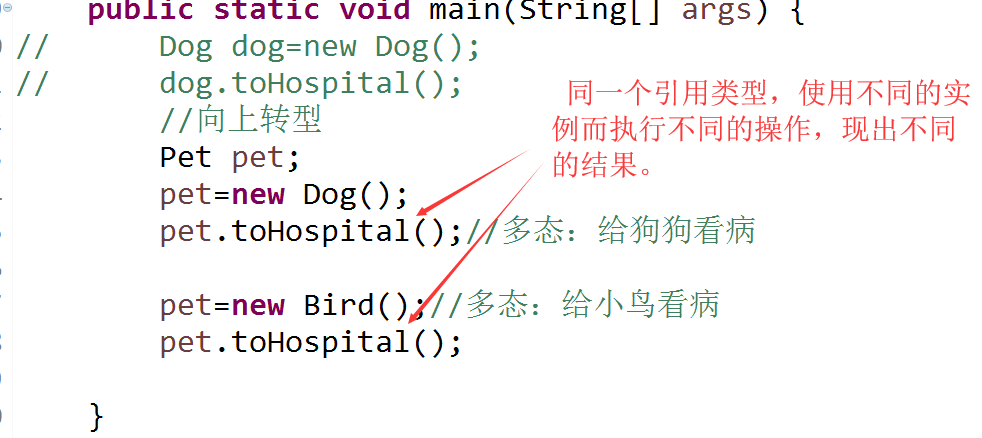
自己定义的类可以去重写object中的方法，比如重写equals方法

常用String提供的equals方法，也是重写object的equals方法。

#### 多态的应用

面对对象的三大特征之一

1. 同一引用类型，使用不同的实例方法而执行不同的操作，呈现出不同的结果
2. 多态依靠重写实现



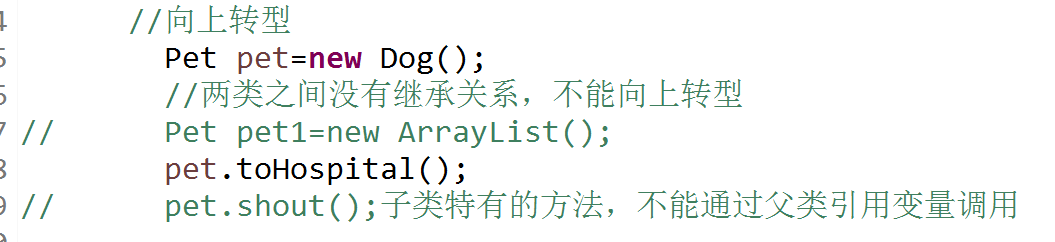
向上转型

父类引用指向子类的对象，自动进行类型转换

语法格式 父类型 引用变量 = new 子类（）；

注意

1. 通过父类引用变量调用的方法是子类重写的方法
2. 父类引用变量无法调用子类特有的方法。



向下转型

将父类对象赋值给子类，需要强制转换

语法格式： 子类型 引用变量 = （子类型）（父类型引用变量）



Instanceof关键字使用



因为pt1对象并没有指向一个Bird对象，肯定强转不了。 为了防止强制转换，出现上述错误，导致程序停止，最好先判断下能不能强制转换。



# 抽象与接口

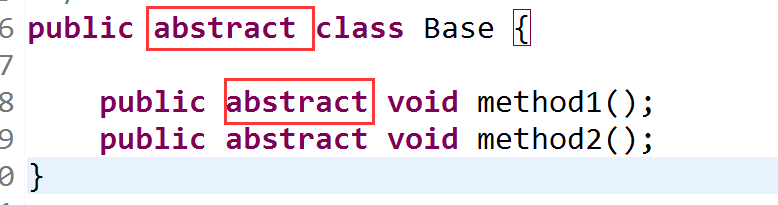
##### 抽象

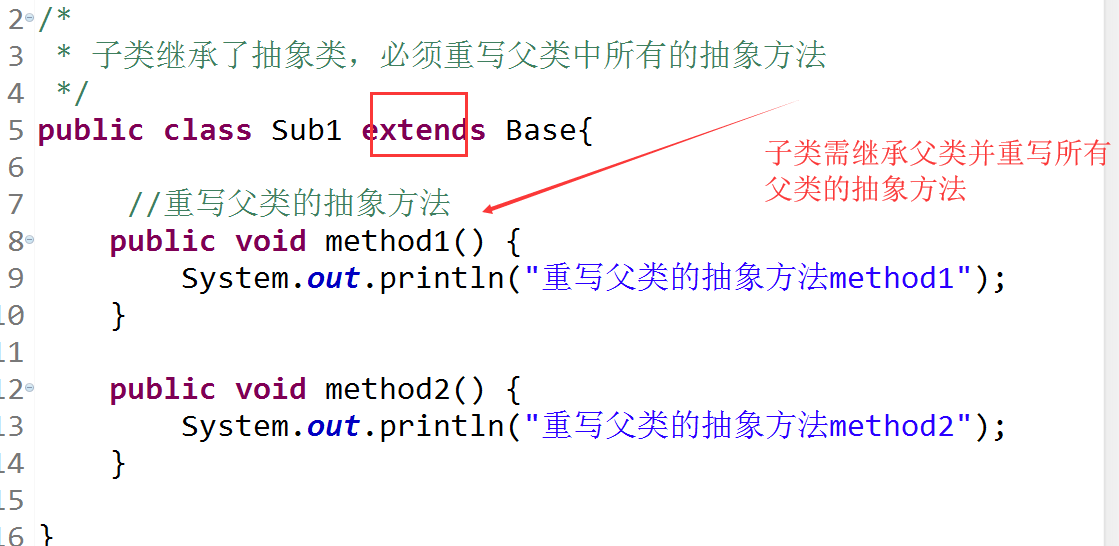
抽象类

1、当一个类的方法被abstract关键之修饰，该方法称为抽象方法。（抽象方法所在的类，必须被abstract修饰）

2、被abstract修饰的方法称为抽象方法。

3、子类继承抽象类，必须实现父类中所有的抽象类（子类为抽象类就不用全部实现父类的所有抽象方法）





区分普通方法和抽象方法

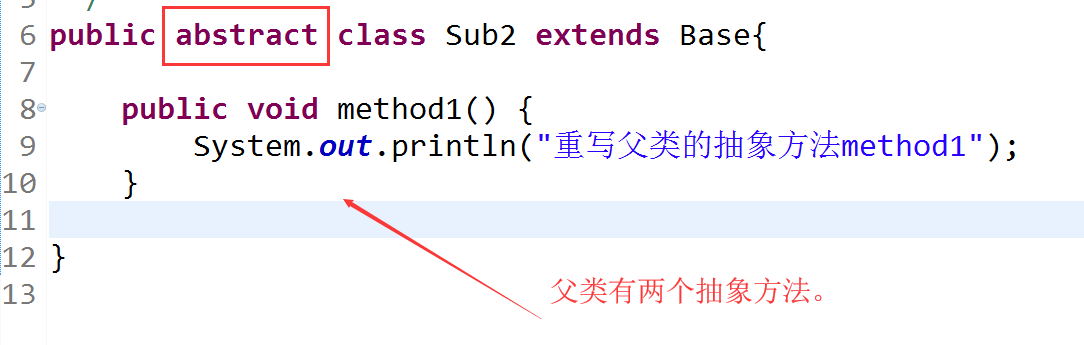
1. 抽象方法需要用abstractx修饰
2. 抽象方法没有方法体。而普通方法有方法体

区分普通类和抽象类

1. 抽象类必须用abstract修饰
2. 普通方法可以实例化，而抽象类不能

子类继承抽象类

子类继承了抽象类，必须重写父类中所有的抽象方法。如果子类没有重写父类所有的抽象方法，子类必须定义为抽象类。并且后续应该还有子类重写Sub2的抽象方法，不然Sub2类没有存在的意义。



抽象类的意义

父类的意义：封装公共的属性和行为被子类共用。为所有子类提供一种统一的类型。

特有意义：包含抽象方法，为所有子类提供一个统一的入口。

##### 接口

所有接口只是用来规定标准，具体的实现内容，由其子类具体实现。

（可以认为接口为特殊的抽象类）

接口特性

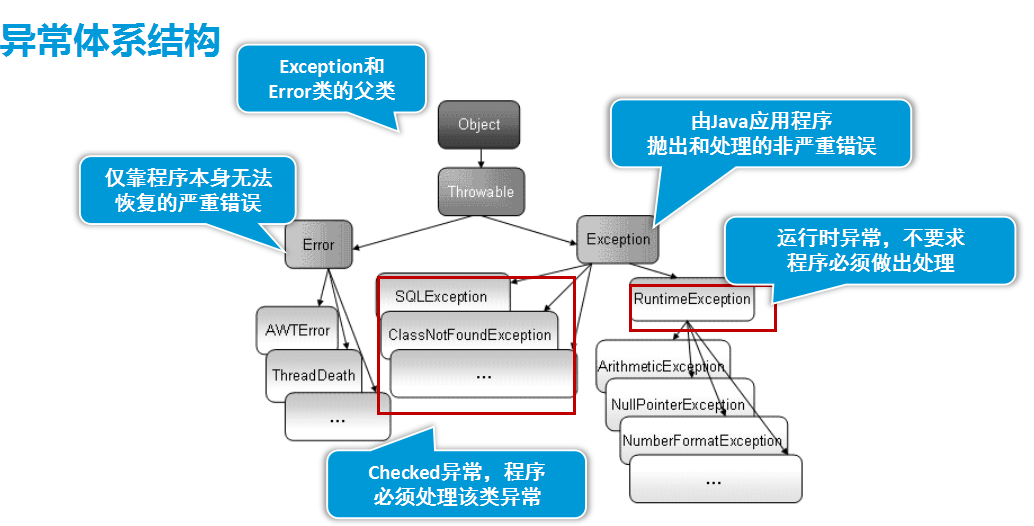
1. 接口不能被实例化
2. 实现类必须实现（implements）接口的所有抽象方法
3. 实现类可以通过implements实现多个接口，多个接口用逗号隔开。若又继承又实现，必须先继承后实现
4. 接口可以继承。但接口不能实现另一个接口
5. 、接口中的变量都是静态常量（public static final）
6. 接口中的方法都为抽象方法
7. 接口中的方法定义好之后不要轻易改动。

接口和抽象类的区别

1. 接口用interface修饰，抽象类用abstract修饰
2. 接口成员只能是常量、抽象方法。而抽象类中可以有普通类的所有成员和方法。
3. 子类通过extends继承抽象类。而接口的子类通过implements实现接口
4. 继承抽象类是单一继承。而实现接口可以多个。

# 异常

##### 异常体系

  
·Error类：表示紧靠程序本身无法恢复的错误。

比如：内存溢出，动态链接失败，虚拟机错误。

·Exception类：java程序抛出和处理不严重的错误。

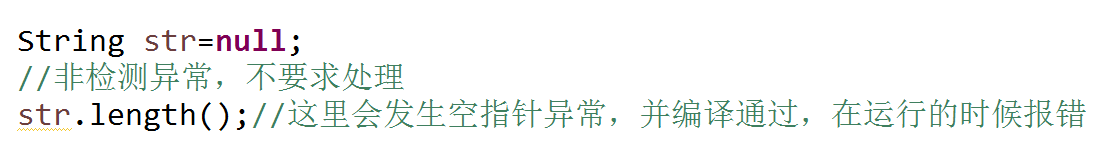
比如:数组下标越界，空指针异常，类型转换错误，算术运算错误

等等.....

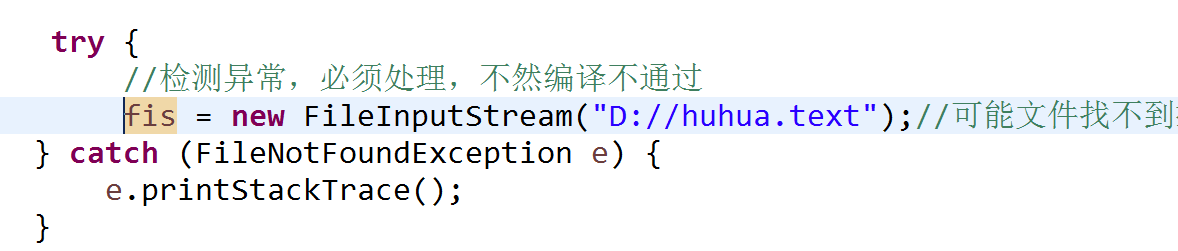
我们程序员关注点在Exception类。

Exception分为两大异常

1. 非检测异常（运行异常RumtimeException）:不要求程序必须对它们处理。



1. 检测异常（非运行异常）：编译不通过，必须进行异常处理（try-catch/throws）



Try-catch进行异常处理

语法：try{

可能发生异常的代码

}catch(异常类型 变量){

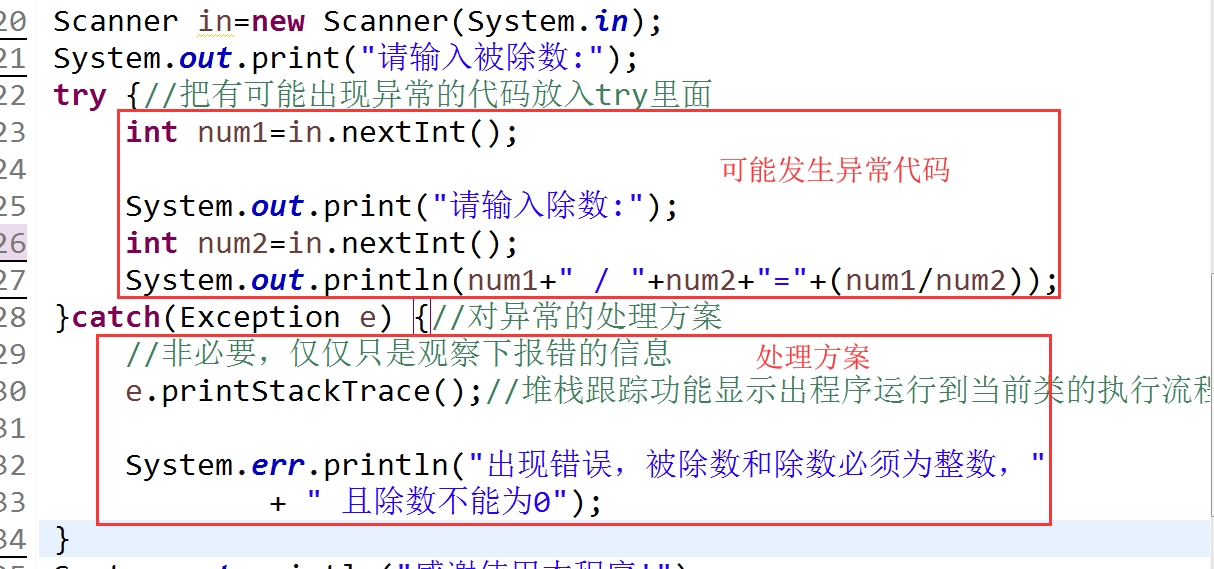
异常处理方案

}

1、在try-catch的执行过程中，try里面的代码可能会产生一种或者多种异常对象。

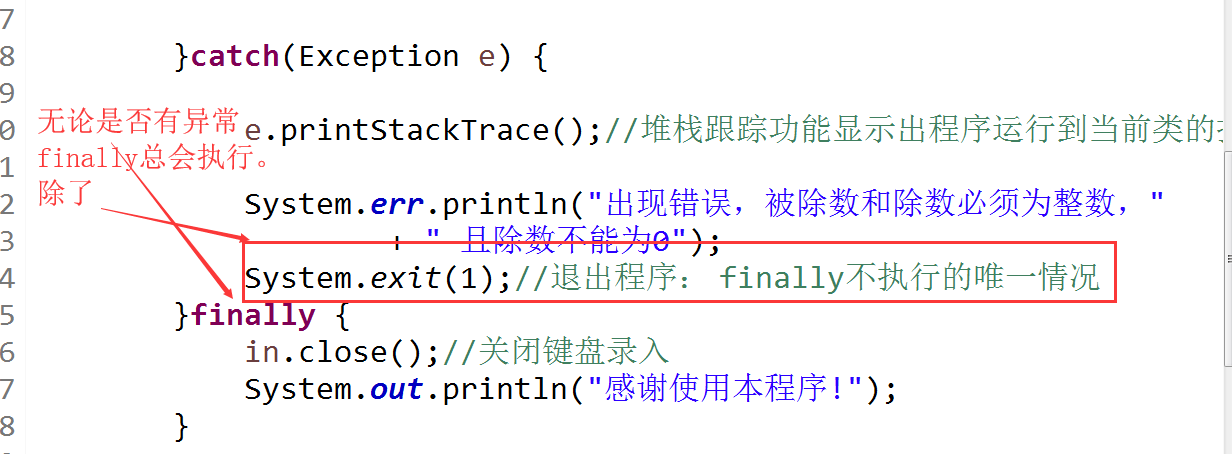
后面的catch会捕捉这些对象，并执行相应的异常处理方案

1. 如果没有异常发生，所有的catch代码块忽略不执行。

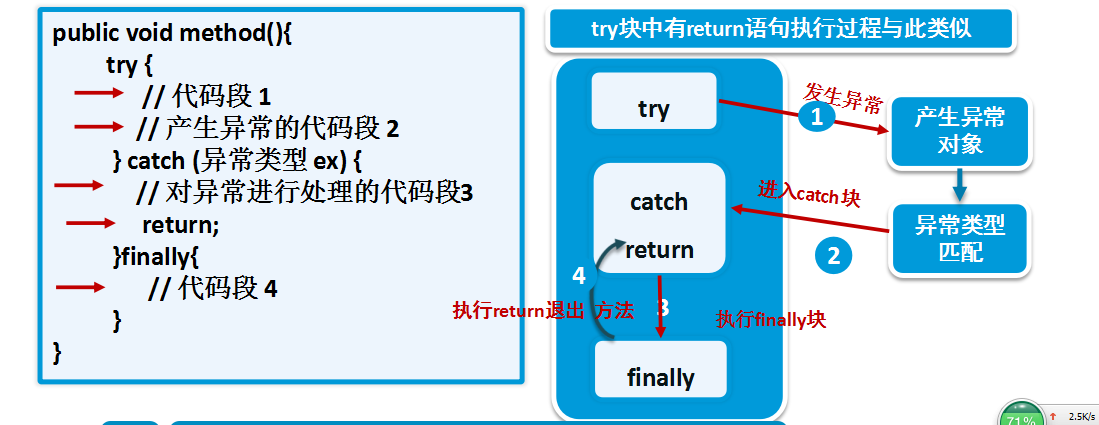


Try-catch-finally异常处理

1. finally块被定义在异常捕捉机制catch最后面
2. 能保证finally中的代码一定会被执行
3. 无论是否抛出异常，finally块一定执行，除了finally之前有System.exit（1）不会执行
4. 通常用于释放内存，关闭io流，关闭数据库链接等操作放在finally中。

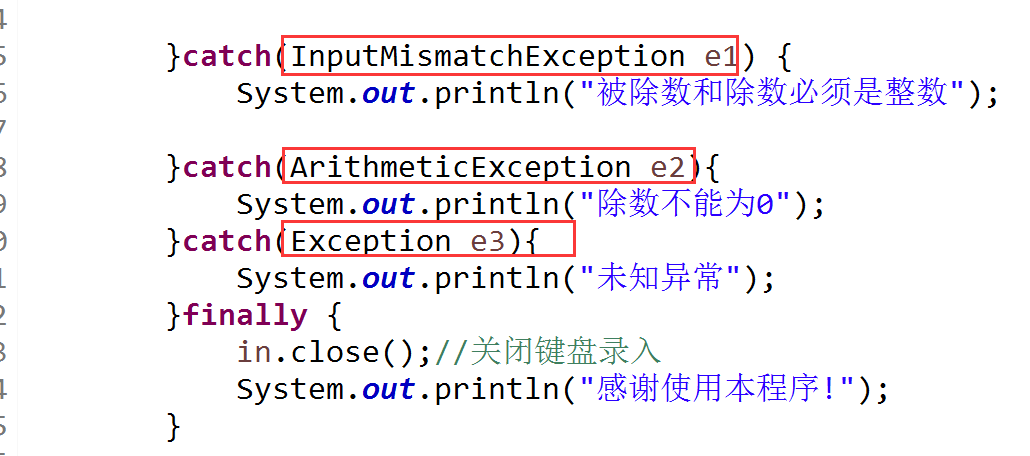


Try块和catch块中有return语句的执行顺序



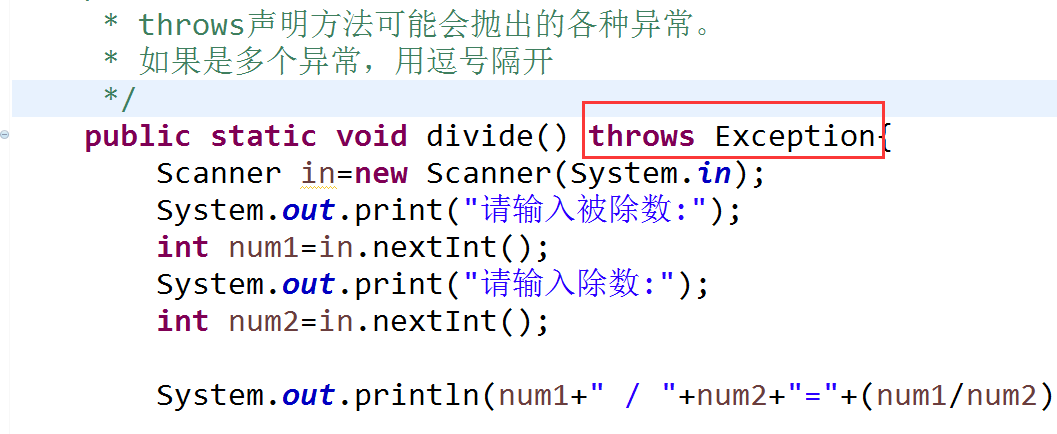
多种catch

如果多个catch的异常类型之间存在继承关系，那么父类必须在子类后面。

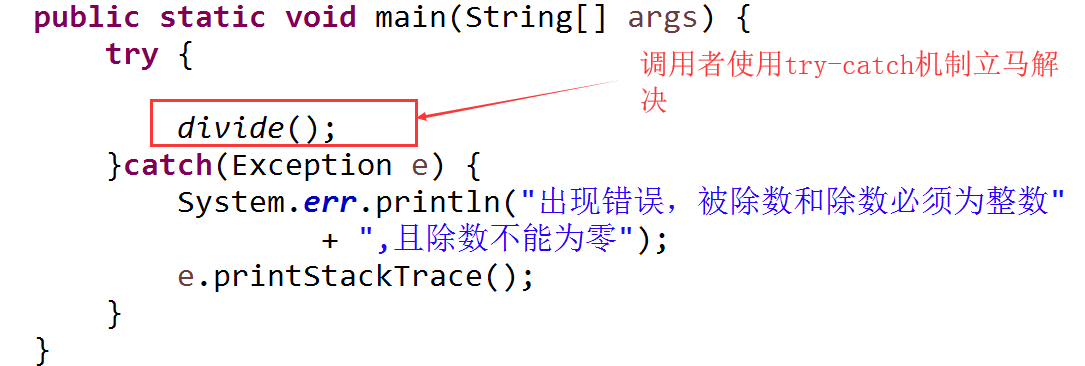


声明异常

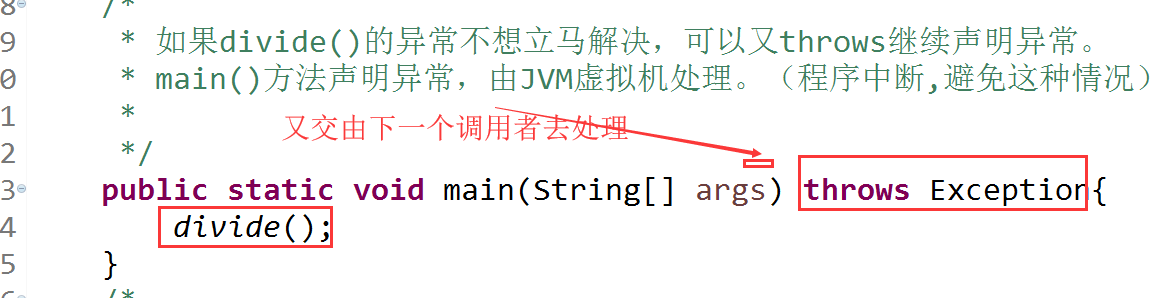
如果异常不想马上解决，而在方法上声明异常，交由调用者处理。使用throws关键字



调用者异常处理1



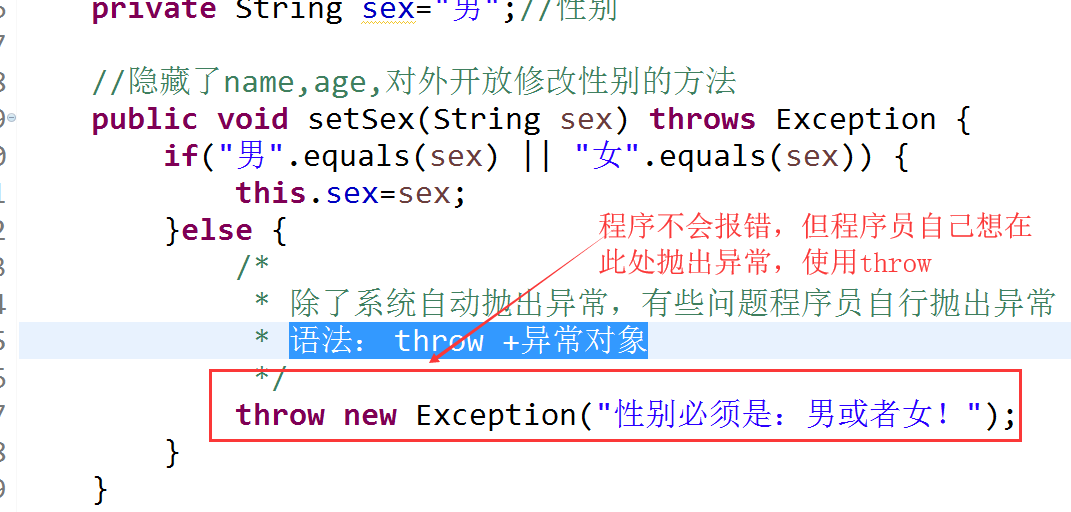
调用者异常处理2



上述案例都是JDK自动帮我们抛出异常，那么有时需要程序员自动抛出异常呢？

主动抛出异常throw

语法：throw+异常对象



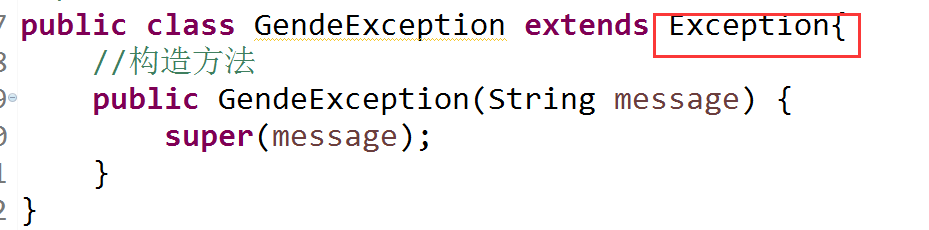
Throw和throws区别



自定义异常

如果在某些场景，jdk没有提供相对应的异常类型，程序员可以自定义异常。

自定义异常步骤

1. 继承Throwable类、继承Exception或者RumtimeException
2. 编写构造方法，调用父类的构造方法
3. 
4. 使用自定义异常案例
5. 