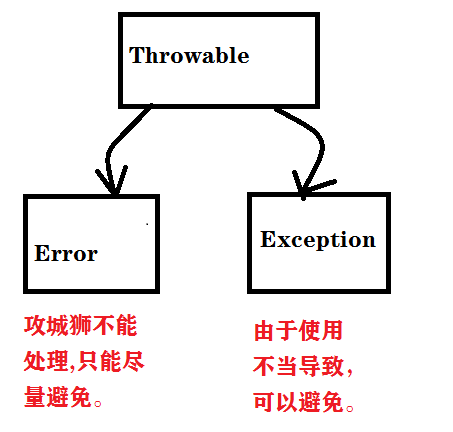
1.异常概念

异常：指的是程序在执行过程中，出现的非正常的情况，最终会导致JVM非正常停止。

异常指的并不是语法错误，语法错了，编译不通过，不会产生字节码文件，根本不能运行。



java.lang.Throwable类是java语言中所有错误或异常的超类。

1.Exception：编译期异常，进行编译（写代码）java程序出现的问题。

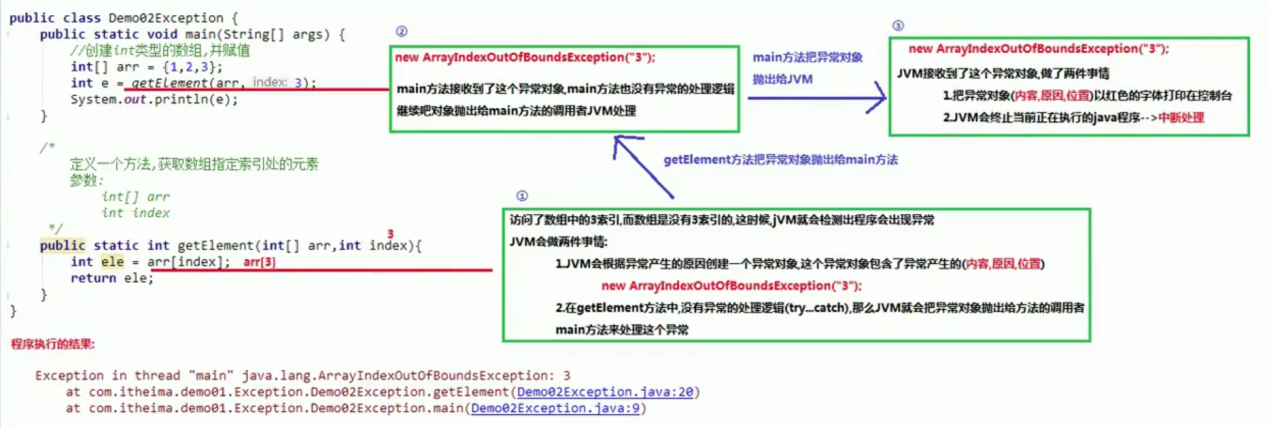
1、RuntimeException:运行期异常，java程序运行过程中出现的问题。

异常就相当于程序的一个小毛病（感冒，发烧），把异常处理掉，程序就可以继续执行。（吃点药，打点滴，继续革命工作）

2.Error：错误

错误就相当于程序得了一个无法治愈的毛病（艾滋病，癌症），必须修改源代码 。程序才能继续执行。

2.异常的产生过程解析（分析异常是怎么产生的，如何处理异常）



3.异常的处理

1.throw关键字

作用：

可以使用throw关键字在指定的**方法中抛出**指定的**异常**

使用格式：

throw **new** xxxException(“异常产生的原因”);

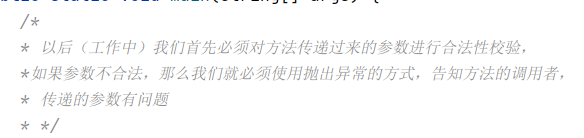
注意事项：

1、throw关键字**必须写在方法的内部**

2、throw关键字后边new的对象必须是Exception或者Exception的子类对象

3、throw关键字抛出指定的异常对象，我们就必须处理这个异常对象

throw关键字后边创建的是**RuntimeException或者RuntimeException子类对象**，我们可以不处理，默认交给JVM处理（打印异常对象，中断程序）

throw关键字后边创建的是**编译异常（写代码的时候报错）**，我们就**必须处理**这个异常，要么**throws**，要么**try…catch….**  



需要注意的是，以上抛出的两种异常都是RuntimeException,我们可以不用处理，默认交给JAVA虚拟机去处理。

2.Objects中的非空判断\_requireNonNull

Objects类中的静态方法

-public static <T> requireNonNull();查看指定引用对象是不是null



以后判断参数是否为空，可以用**Objects**的这个静态方法。



3.异常处理的第一种方式:**声明异常:throws**

--throws关键字:异常处理的第一种方式：**交给别人处理**

作用：

当方法内部抛出异常对象的时候，那么我们就必须处理这个异常对象，

可以使用throws关键字处理异常对象，会把异常对象声明抛出給方法的调用者处理（自己不处理，给JVM处理）

使用格式：

修饰符 返回值类型 **方法名**（参数列表） **throws** Aexception,Bexception…{

throw new AException(“产生原因”);

throw new BException(“产生原因”);

…

}

注意事项：

1.throws关键字必须写在**方法声明处(即方法名（参数列表）后面)**

2.throws关键字后边声明的异常必须是Exception或者是Exception的子类

3.方法内部如果抛出了多个异常对象，那么throws后边也必须声明多个异常

如果抛出的多个异常对象有子父类关系，那么直接声明父类异常即可。

4.调用了抛出异常的方法（**且异常是编译异常**），我们就必须得处理抛出的异常，要么继续使用throws声明抛出，交给方法的调用者处理，最终交给JVM，要么try…catch自己处理异常。





4.异常处理的第二种方式捕获异常try…catch(自己处理异常)

**声明异常**的**弊端**：如果异常后还有后续代码，就不会执行了（JVM中断了程序）。

格式：

try{

**可能**产生异常的代码

}catch（**定义一个异常的变量，用来接收try中抛出的异常对象**）{

异常的处理逻辑，产生异常对象之后，怎么处理异常对象

**一般在工作中，会把异常的信息记录到一个日志中（这算重点）**

}

…

catch(异常类名 变量名){}

注意：

1.try中**可能会抛出多个异常对象**，那么就可以**使用多个catch来处理这些异常对象**

2.如果try中**产生了异常**，那么就会执行catch中的一次处理逻辑，执行完毕catch中的处理逻辑，继续执行try…catch之后的代码

如果try中**没有产生异常**，那么就不会执行catch中异常的处理逻辑，执行完try中的代码，继续执行try…catch之后的代码



5.Throwable中定义了3个异常处理的办法。

- String getMessage();返回此throwable的简短描述。

- String toString();返回此throwable的详细消息字符串。

- void printStackTrace();JVM打印异常对象，默认此方法（打印的异常消息最全面）



6.finally代码块

有一些代码无论是否出现异常，我都要让其执行，就可以使用finally代码块。//比如释放资源

try{

}catch(){

}finally{

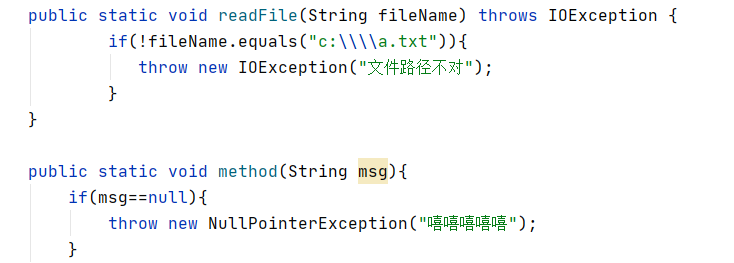
//无论是否出现异常都会执行

}

注意事项:

1.finally**不能**单独使用，必须**和try一起使用**

2.finally一般用于资源释放（资源回收），无论程序是否出现异常，最后都要资源释放（IO）





7.异常的注意事项

1、多异常的捕获处理

多个异常使用捕获又该如何处理呢?

1.多个异常分别处理。（多个try…catch）

2.多个异常一次捕获，多次处理。（一个try，多个catch）

3.多个异常一次捕获一次处理。（一个try，一个catch）

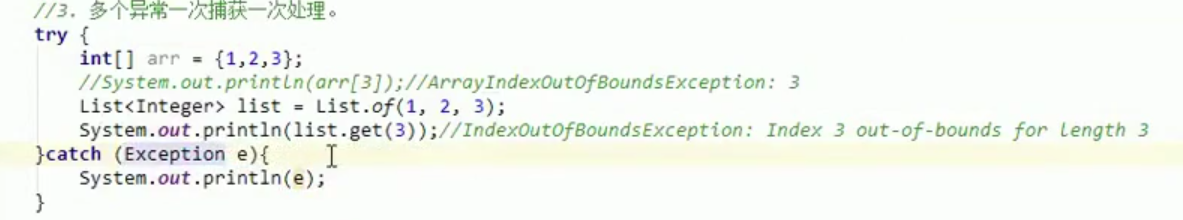
/\*\*

一个try多个catch注意事项：

catch里面定义的异常变量，如果有子父类关系，那么子类的异常变量必须写在上边，否则就会报错。







2.finally代码块中有return语句

如果finally有return语句，永远返回finally中的结果，（应避免该情况）



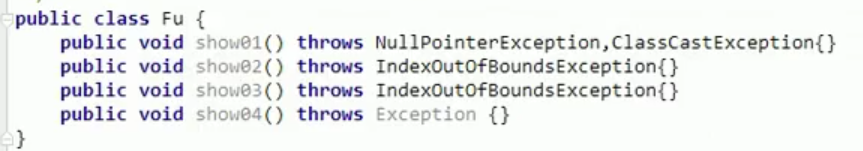
3.子父类的异常

如果父类抛出了多个异常，子类重写父类方法时，抛出和父类相同的异常或者是父类异常的子类或者不抛出异常。

父类方法**没有**抛出异常，子类重写父类该方法式**也不可抛**出异常，此时子类**产生该异常**，**只能**捕获处理，**不能**声明抛出。

注意：

父类异常是什么样，子类异常就什么样。 （记住这句就好了）





4.自定义异常类

Java提供的异常类，不够我们使用，需要自己定义一些异常类。

格式：

-public class XXXException extends Exception | RuntimeException{

添加一个空参数的构造方法。

添加一个带异常信息的构造方法。

}

注意：

1.自定义异常类一般都是以Exception结尾，说明该类是个异常类。

2.自定异常类，必须得继承Exception或者RuntimeException

继承Exception：那么自定义的异常类就是一个编译异常，如果方法内部抛出了编译期的异常，就必须处理，要么throws，要么try…catch。

继承RuntimeException：那么自定义的异常类就是一个运行期异常，无需处理，交给JVM处理（中断程序）







