Java中的异常机制

1. 什么是异常（Exception）？ **运行时期**发生的不正常情况，与编译无关。 中断产生了正常指令流的事件。编译时出的错误是语法的错误；异常与编译无关，是在程序执行的过程中出现的。
2. **Java中用类的形式对不正常情况进行了描述和封装对象，描述不正常情况的类称为异常类。 以前正常流程代码和问题处理代码相结合，现在将正常流程代码和问题处理代码相分离，目的就是为了提高阅读性。**
3. **其实异常就是java通过面向对象的思想将问题封装成了对象， 用异常类对其进行描述 。不同的问题用不同的类进行描述。 比如数组就有角标越界，空指针等等。**
4. **出现的问题会很多，意味着描述问题的类也狠多，但是都有些共性，将他们的共性进行向上抽取，就形成了异常体系。 最终，问题（或叫作不正常情况）分成了两大类 ：一是一般不可处理的，称为Error；二是可以处理的 ，称为Exception异常。无论Error还是Exception，都是问题，问题发生时，就应该及时抛出，让调用者知道并处理， Error和Exception都具有可抛出的特性，Throwable。该体系的特点就在于Throwable 以及所有的子类都具有可抛性，不是该体系的不具有可抛性 。**
5. **Error** 是有JVM抛出的严重性的问题， 这种问题一般不做针对性处理，直接修改程序。

**Exception** 是可以处理的。

1. Throwable下面主要有两个Error和Exception，**Throwable体系**中，子类的后缀都是用父类的名作为**后缀名**，阅读性比较强。
2. 一旦发生异常就可以把当前的方法结束掉。方法中抛出异常，立即结束此方法，主方法直接抛出异常给JVM，抛出异常后，不会再执行此方法中其他的语句。
3. 如果方法中可能会产生异常，则必须抛出异常（**抛出异常**利用关键字throw），并必须在此方法的声明处声明此异常或在方法体中捕捉此异常，声明异常利用**shows关键字**，如果某方法中调用了**可能会抛出异常的方法**，则调用了可能抛出异常方法的方法也需要抛出此异常或者直接进行捕捉。对异常对象的处理有两种方法：一是直接抛出（需要声明），二是直接进行捕捉并处理。
4. **可抛性到底是什么？怎么体现可抛性呢？**

**答：通过两个关键字来实现 ，throws 和 throw， 凡是可以被这两个关键字操作的类或者对象都具有可抛性。**

**throws和throw的不同点：**

**（1）throws 使用在方法声明处 ，throw使用在方法内部；**

**（2） throws后面跟的是异常类名，可以同时声明多个异常类，之间利用逗号隔开，单对于方法来说，遇到一个异常便不再继续正常执行了，throw后面跟的是异常对象（一般是匿名对象，new XXX（）），只可以抛出一个异常对象。**

1. **如果想自定义一个类为异常类，必须让此类继承异常体系，因为只有成为异常体系中的一个子类，才具有可抛性，才可以被两个关键字throw和throws 操作。定义异常类的命名规范：使定义的异常类的名字以继承的父类的名字为后缀，这是规范，不是强制的。**
2. **在方法声明处声明可能产生的异常的目的是：让调用此方法的调用者对此异常进行处理。**
3. **如果想在异常提示信息处输出想要的提示信息，在自己定义的异常类的构造方法中调用super（\*\*\*\*）；即可了，因为对于提示信息的输出方法父类已经写好了，子类直接调用父类的相应的构造方法即可了。**
4. **简单介绍异常中的几个方法：**
5. **Exception中有个叫作getMessage（）；方法，可以调用出异常提示信息 的具体内容，返回值是String类型；**
6. **Exception中有个叫作toString（）；方法，返回的是String类型，内容为异常类名和异常提示信息；**
7. **Exception中有个叫作printStackTrace（）；方法，返回值为空，直接就会输出异常类名、异常提示信息以及异常发生的位置。ＪＶＭ默认情况下，异常处理机制调用的就是异常对象的这个方法。**
8. **Log4j :就是log for java 的简写，记录的是java运行的日志信息。就可以利用重写printStackTrace方法，直接输出到文件中。**
9. **Throwable 与 Runnable**
10. **异常Exception 的分类：**
11. 编译时被检测异常Check Exception（编译器会提醒修改）（此异常可以处理，故声明出来此异常，然后对其进行处理即可）：只要是Exception及其子类都是，但是除了子类RuntimeException体系。 异常一旦出现，希望在编译时就进行检测，使该异常有对应的处理方式。 总之，这种异常都会有针对性的处理。
12. 编译时不检测异常（运行时异常） Uncheck Exception（ 编译器不会强制修改）：（此异常是不可以处理的，既然不可处理，就没有必要声明出来，即使声明处理，也没法处理）Exception的子类**RuntimeException及其子类，对于异常无论处理不处理，都可以编译通过。这种异常的发生，无法让功能继续，运算无法进行，更多的是因为调用的原因导致的，或者是引发了内部状态的改变导致的。这种异常一般不处理，编译时直接通过，在运行时，让调用者调用的程序强制停止，让调用者对代码进行修正。声明处理，利用代码处理掉此异常也没有多大意义，所以直接停止程序即可了。**
13. 异常是个类。Throwable （父类），一个子类是Error，错误，一旦出现error，虚拟机直接退出，无法运行，另外一个子类是Exception，在Exception下面又有很多子类，如RuntimeException （及其子类都属于Uncheck Exception） 。除了**RuntimeException及其子类**之外的其他的直接继承Exception的子类及其子类都是Check Exception 。
14. **对异常进行捕捉：**

try {

可能出现异常或者需要被检测的代码；

}

catch（异常类名 异常类引用变量）{//异常类引用变量是用于接收发生异常对象的

处理异常的语句。

（如果出现异常就执行这里面的语句，如果不出现异常，则不会执行catch里面的语句。）

}

finally{

\*\*\*\*\*\*\*\*\*

}

语句的作用：对于一般情况下（使用try….. catch…..finally….），则遇到异常则不能继续执行了，但是用上try….. catch…..finally….，就可以使程序继续执行，增加程序的健壮性。finally 语句的作用是无论有否异常，都会执行finally里面的语句，finally语句通常做的是一些清理扫尾工作。有无均可。

利用try….. catch…..finally….语句可以对异常进行捕捉。

1. 把可能出现异常的代码放在try{}里面；把对异常处理的代码放在catch（Exception e）{}里面；把清理资源的代码放在finally{}里面。
2. 与异常相关的关键字 throw(用来抛出异常)和throws（用来声明异常）。

异常类RuntimeException e = new RuntimeException (“异常提示信息”); //生成异常对象 。(Uncheck Exception)

throw e； // 抛出异常信息

throw的作用：当java虚拟机无法判断是否是异常时，如年龄不能为负值，此时可以认为抛出异常对象。

throws的作用：声明此方法可能会出现异常，不在此方法内进行异常处理，而是在调用此方法的地方进行异常处理。

throws ;声明一个方法本身可能会出现异常，谁调用这个方法，谁必须进行异常处理；例如：

public void setAge (int age) throws Exception {

if(age < 0){

RuntimeException e = new RuntimeException();

throw e; //抛出异常

}

}

1. 实例：

class Test5{

public static void main (String [] args){

Student student = new Student();

try{

student.setAge(-23);

}

catch(Exception e){

System.out.println(e);

}

}

}

class Student{

int age;

public void setAge(int age) throws Exception{

if (age < 0){

Exception e = new Exception("这是异常：年龄不能小于0"); //新建一个异常对象

throw e;

}

else {

System.out.println("这是正常的，无异常出现");

}

this.age = age;

}

}

1. 一个try语句多个catch语句情况：（面试时经常会遇到）

一个try语句中可能会出现多中异常，那么每一种异常都用对应的catch语句进行捕捉，但是如果catch语句并没有捕捉所有的异常，在所有的捕捉具体异常对象的catch语句后面加上一句 catch（Exception e）{\*\*\*} ，即可对多余的异常对象进行捕捉，记住一定要放在最后，否则这个catch语句，会捕捉到任意一个异常了，那么它后面的catch语句都是多余的。**所以多个catch语句时，父类的异类捕捉一定要放在子类的下面。** 建议： 在catch捕捉异常对象的时候，就捕捉最具体的异常，这样省得再使用多态的性质了。如

try {

demo1.arrayDemo(arr,-20);

}

catch (FuShuJiaoBiaoException e) {

e.printStackTrace();

}

catch (NullPointerException e) {

e.printStackTrace();

}

catch(Exception e) { //可以捕捉到任意一个异常，所以一定要放到catch的最后一个。

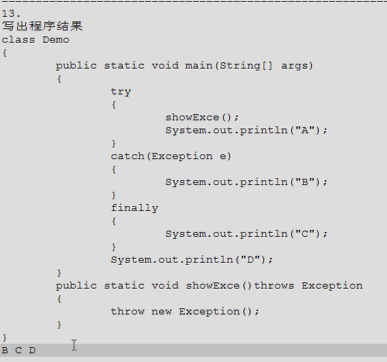
e.printStackTrace();

}

1. 异常处理的原则：
2. 方法内容如果抛出需要检测的异常，那么方法声明上必须声明异常，否则必须在使用try catch语句进行捕捉异常，否则编译失败。
3. 如果调用了声明异常的方法，那么对此调用语句要么利用try catch 语句进行捕捉或者要么在方法声明处再次声明异常throws，否则编译失败。
4. 什么时候利用throws？ 什么时候使用try catch呢？

答：如果此异常能够处理，则就利用try catch语句进行捕捉并处理，如果处理不了，就用throws 声明给调用者，让调用者进行处理。

1. 如果某段代码（try语句）会抛出多种异常，则需要利用多个catch语句进行针对性的捕捉和处理。内部有几个需要检测的异常，就需要抛出几个异常，抛出几个，就检测几个。尽量不要使用当前异常的父类异常去接收异常，因为这样会带来一些意想不到的问题。
2. 异常出现的地方，如果自己能够进行处理，就不要抛出异常，而是直接利用try catch 进行处理，在catch处理过程中可能会出现新的异常，例如处理失败，无法修复等等，此时可以在catch语句中抛出新的异常。这就称作是对异常进行封装，没有必要暴漏的问题，就不要暴漏，即使暴漏了，调用者也处理不了，可以进行对异常的转换
3. finally语句通常用于关闭或释放资源。finally语句在数据库操作时使用最多。连接数据库，查询数据，断开连接。无论查询过程正常或者出现异常，都必须断开连接。断开连接的动作就由finally语句完成。
4. try{} catch（）{} finally{}
5. 可以没有finally语句，不需要关闭或释放资源时使用；
6. 可以同时有多个catch（） {} 语句，会出现多个异常的情况下；
7. try{} finally{} 组合，用于内部无法处理异常，需要抛出异常，但是必须需要关闭或释放资源，所以加上了finally语句，若不用try finally语句，出现异常立即抛出，就不会关闭资源了。
8. 异常的注意事项：
9. 如果子类的方法重写了父类中的方法，父类的此方法抛出了异常，那么子类的此方法只能抛出此异常或者这个异常的子类异常。这是java的规则，记住即可。原因：由于父类的引用 可以接收子类的对象，若代码只可以对父类的方法的异常进行接收，又由于多态调用的是子类中的重写方法(重写方法不允许出现父类中没有的异常)，则子类抛出的异常若不是父类方法抛出异常或者其子类异常，则就无法接收，造成代码错误。
10. 若父类抛出多个异常，那么子类方法只可以抛出父类抛出异常的子集（包括父类异常的全部）。
11. 综上，子类重写父类的方法，子类重写方法只能抛出父类被重写方法的异常本身或者其子类或者其子集。如果父类的被重写的方法没有抛出异常，子类重写的方法也绝对不能抛出异常。
12. 如果子类重写方法中出现了父类中没有抛出的异常，这是子类重写方法中只能try catch ，自己进行捕捉这个异常，不能抛出，如果抛出，则编译就会出错。 **子类重写方法**不能抛出父类被重写方法所抛出异常的范围。



注意：finally 语句执行完后，相当于异常处理完毕，所以此时就可以继续往下执行。区别于下面的题目。



总结： 由于catch中有个return语句，意思是要跳出此方法，但是在跳出之前必须执行finally语句，至于finally语句后面的语句就不在执行了。

D