教学任务：

第七章 异常, 共 16 个slide(172-188);

目标： 1. 异常处理流程；
2. 自定义异常；

第七章： Exceptions (172-188)

Java语言按照面向对象的思想来处理异常，使得程序具有更好的可维护性。Java异常处理机制具有以下优点：

. 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。把异常情况表示成异常类，可以充分发挥类的可扩展和可重用的优势。
. 异常流程的代码和正常流程的代码分离，提高了程序的可读性，简化了程序的结构。
. 可以灵活地处理异常，如果当前方法有能力处理异常，就捕获并处理它，否则只需要抛出异常，由方法调用者来处理它。

知识点：一. 异常的基本概念

1. 异常产生的条件
   或者称为异常情况。在Java代码中哪些是异常情况呢？ 例如：
   a. 整数相除运算中，分母为0;
   b. 通过一个没有指向任何具体对象的引用去访问对象的方法；
   c. 使用数组长度作为下标访问数组元素；
   d. 将一个引用强制转化成不相干的对象；
   等等；
2. 异常会改变正常程序流程;
   异常产生后，正常的程序流程被打破了，要么程序中止，要么程序被转向异常处理(catch)的语句；
   throw new Exception():如果throw抛出的是exception异常会报错,必须处理
   如果其他exception子类型,不会显示报错.但是任然需要处理.否则运行出错
3. 当一个异常的事件发生后，该异常被虚拟机封装形成异常对象抛出。
4. 用来负责处理异常的代码被称为异常处理器
5. 通过异常处理器来捕获异常

举例：
```
class ExceptionTest {
    public static void divide(int a, int b) {
        //try {
            int result = a / b;
            System.out.println(a + "/" + b + "=" + result);
        //} catch(ArithmeticException e) {
        //    System.out.println("Sorry, error in divide");
```

```
                    //}
            }

            public static void main(String[] args) {
                divide(1,2);
                divide(10,2);
                divide(10,0);
                divide(10,5);
            }
        }
```

当遇到异常情况时,可以使用throw来自定义异常输出信息

二. try...catch语句

在Java语言中，用try...catch语句来捕获处理异常。格式如下：

```
try {
    可能会出现异常情况的代码；
} catch(异常类型 异常参数) {
    异常处理代码
} catch(异常类型 异常参数) {
    异常处理代码
}
```

程序如果遇到异常,可以使用两种方式进行处理
1.throws 抛出异常 实际并没有处理异常
  如果一致没人处理,最后交付给虚拟机处理
2.try…catch 处理异常,可能产生异常的代码位于try语句中
    处理的内容位于catch代码块中.在catch代码块中
    可以使用getMessage()来捕获throw里面抛出的自定义内容
catch语句块中
1.可以有多个Exception，但是不是每一个都执行.try中出现了那一个Exception,它才进入到哪一个catch的exception中进行处理
2.不允许出现相同Exception
3.Exception出现的位置是有顺序的,只能越往下范围越大,不能越往下范围越小,允许相同等级.

语句执行顺序

1. 如果try代码块中没有抛出异常，try代码块中语句会顺序执行完，catch代码块内容不会被执行；
2. 如果try代码块中抛出catch代码块所声明的异常类型对象，程序跳过try代码块中接下来代码，直接执行catch代码块中对应内容；
   a. 可以存在多个catch代码块，究竟执行哪个，看抛出的异常对象是否是catch代码块中异常类型；
   b. 异常只能被一个异常处理器所处理, 不能声明两个异常处理器处理相同类型的异常；
   c. 多个catch语句块所声明的异常类型不能越来越小；

3. 如果try代码块中抛出catch代码块未声明的异常类型对象，异常被抛给调用者；哪个调用了这段语句块
哪个负责处理这个异常；

举例：ch07.TryTest.java

三. finally语句: 任何情况下都必须执行的代码

由于异常会强制中断正常流程，这会使得某些不管在任何情况下都必须执行的步骤被忽略，从而影响程序的健壮性。

```
public void work() {
try {
        开门();
        工作8个小时();
        关门();
} catch(Exception e) {
        //异常处理语句
}
}
```

finally代码块能保证特定的操作总是会被执行，它的形式如下：

```
public void work() {
try {
        开门();
        工作8个小时();
} catch(Exception e) {
        //异常处理语句
} finally {
 关门();

}
}
```

当然finally代码块中代码也可位于catch语句块之后，例如：

```
public void work() {
try {
        开门();
        工作8个小时();
} catch(Exception e) {
        //异常处理语句
}

关门();

}

return:
```

举例：
情况1：try{} catch(){}finally{} return;
显然程序按顺序执行。 最后方法返回 return指向的值内容
情况2:try{ return; }catch(){} finally{} return;
当没有异常时:程序执行try块中return之前（包括return语句中的表达式运算）代码；
再执行finally块，最后执行try中return;
finally块之后的语句return，因为程序在try中已经return所以不再执行。
如果此时try中代码有异常,执行try catch finally return.
try中的return执行不到
情况3:try{ } catch(){return;} finally{} return;
程序先执行try，如果遇到异常执行catch块，
有异常：则执行catch中return之前（包括return语句中的表达式运算）代码，再执行finally语句中全部代码，
最后执行catch块中return. finally之后代码不再执行。
无异常：执行完try再finally最后return.
情况8:try{ }catch(){} finally(return;) return
此时报错,try代码块后面不能再添加语句
情况4:try{ return; }catch(){} finally{return;}
程序执行try块中return之前（包括return语句中的表达式运算）代码；
再执行finally块，因为finally块中有return所以提前退出。
情况5:try{} catch(){return;}finally{return;}
程序执行catch块中return之前（包括return语句中的表达式运算）代码；
再执行finally块，因为finally块中有return所以提前退出。
情况6:try{ return;}catch(){return;} finally{return;}
程序执行try块中return之前（包括return语句中的表达式运算）代码；
有异常：执行catch块中return之前（包括return语句中的表达式运算）代码；
则再执行finally块，因为finally块中有return所以提前退出。
无异常：则再执行finally块，因为finally块中有return所以提前退出。
错误情况:如果只有try中有return报错.必须在catch()中或者finally中或者try语句块
外面再添加一个return关键字
情况7:try{return;}catch(){return;}finally{}
程序执行try块中的return之前代码
有异常执行catch块中的return之前代码,返回catch中的return值
没有异常返回try中的return之前的代码
最终结论：任何执行try 或者catch中的return语句之前，都会先执行finally语句，如果finally存在的话。
如果finally中有return语句，那么程序就return了，所以finally中的return是一定会被return的，
编译器把finally中的return实现为一个warning

四. 异常调用栈

异常处理时所经过的一系列方法调用过程被称为异常调用栈。

1. 异常的传播

哪个调用，哪个处理；

a. 异常情况发生后，发生异常所在的方法可以处理；

b. 异常所在的方法内部没有处理，该异常将被抛给该方法调用者，调用者可以处理；

c. 如调用者没有处理，异常将被继续抛出；如一直没有对异常处理，异常将被抛至虚拟机；

2. 如果异常没有被捕获，那么异常将使你的程序将被停止。

异常产生后，如果一直没有进行捕获处理，该异常被抛给虚拟机。程序将被终止。

3. 经常会使用的异常API

getMessage：获得具体的异常出错信息，可能为null(空指针异常)

printStatckTrace()：打印异常在传播过程中所经过的一系列方法的信息，简称异常处理方法调用栈信息；在程序调试阶段，此方法可用于跟踪错误.可以显示那一行引起的错误信息.

系统默认异常输出信息.

举例：ch07.CallStackTraceTest

五. 异常层级关系

所有异常类的祖先类为java.lang.Throwable类。它有两个直接的子类：

1. Error类：表示仅靠程序本身无法恢复的严重错误，比如内存空间不足，或者Java虚拟机的方法调用栈溢出。在大多数情况下，遇到这样的错误时，建议让程序终止。

2. Exception类：表示程序本身可以处理的异常。Exception还可以分为两种：运行时异常和受检查异常。

a. 运行时异常

RuntimeException类及其子类都被称为运行时异常，这种异常的特点是Java编译器不会检查它，也就是说，当程序中

可能出现这类异常时，即使没有用try...catch语句捕获它，也没有用throws子句声明抛出它，还是会编译通过。例如divide()方法的参数b为0, 执行a/b操作时会出现ArithmeticException异常，它属于运行时异常，Java编译器不会检查它。由此可见，运行时异常是应该尽量避免的，在程序调试阶段，遇到这种异常时，正确的做法是改进程序的设计和实现方式，修改程序中的错误，从而**避免这种异常**。捕获它并且使程序恢复运行并不是明智的办法，因为即使程序恢复运行,可能会导致程序的业务逻辑错乱，导致更严重的异常，或者得到错误的运行结果

```
public int divide(int a, int b) {
    return a/b;     //当参数b为0, 抛出ArithmeticException
}
```

b. 受检查异常。

除了RuntimeException及其子类以外， 其他的Exception类及其子类都属于受检查异常(Checked Exception)。 这种异常的特点是Java编译器会检查它，也就是说，当程序中可能出现这类异常时，要么用try...catch语句捕获它，要么用throws子句声明抛出它，否则编译不会通过。

已检查异常 是指程序员已经足够小心的检查了他的代码，但是还是不能保证代码不出现异常；如，程序要访问某个文件，但访问时文件不存在，这和程序本身没有太大关系；再如，程序要进行网络连接，但执行时没有连接网线，这些问题都是已检查异常。

运行时异常 一般是由程序员没有细心检查代码，而导致的如空指针异常、数组越界、类型转换异常等都是由于程序员粗心大意造成的。这些异常是在编码过程中是能够避免的。

六. 一些未检查的异常RuntimeException

1. java.lang.ArithmeticException
   算术异常　　如：除0;
2. java.lang.NullPointerException
   空指针引用 如：没初始化一个References便使用;
3. java.lang.ArrayIndexoutofBoundsException
   数组越界　　如：调用一个有十个元素的Array的第十一个元素的内容;
4. java.lang.NumberFormatException
   数据格式异常　　如：Integer.parseInt("a");
5. java.lang.NegativeArraySizeException 数组长度为负数异常

七. 异常声明和处理

1. 使用throw声明代码会倒致异常；
2. 使用try-catch-finally语句结构处理或在方法声明上声明throws继续抛出；

异常处理语句的语法规则：

1. try代码块不能脱离catch代码块或finally代码块而单独存在。try代码块后面至少有一个catch代码块或finally代码块。

2. try代码块后面可以有零个或多个catch代码块，还可以有零个或至多一个finally代码块。如果catch代码块和finally代码块并存，finally代码块必须在catch代码块后面。

3. try代码块后面可以只跟finally代码块。

4. 在try代码块中定义的变量的作用域为try代码块，在catch代码块和finally代码块中不能访问该变量。

5. 当try代码块后面有多个catch代码块时，Java虚拟机会把实际抛出的异常对象依次和各个catch代码块声明的异常类型匹配，如果异常对象为某个异常或其子类的实例，就执行这个catch代码块，而不会再执行其他的catch代码块。

6. 如果一个方法可能出现受检查异常，要么用try...catch语句捕获，要么用throws子句声明将它抛出。

7. throw语句后面不允许紧跟其它语句，因为这些语句永远不会被执行。

八. 编写自己的异常

在特定的问题领域，可以通过扩展Exception类或RuntimeException类来创建自定义的异常。异常类包含了和异常相关的信息,这有助于负责捕获异常的catch代码块，正确地分析并处理异常。

通常情况下是程序运行状态与用户的预先定义的逻辑不符合，但程序并不能识别这种逻辑错误时需自定义异常。比如某个方法的参数只能接受0~9的数字，数字1除外，万一用户要是输入了1，我们可以自定义一个异常来处理1这个意外，从而控制程序流程等