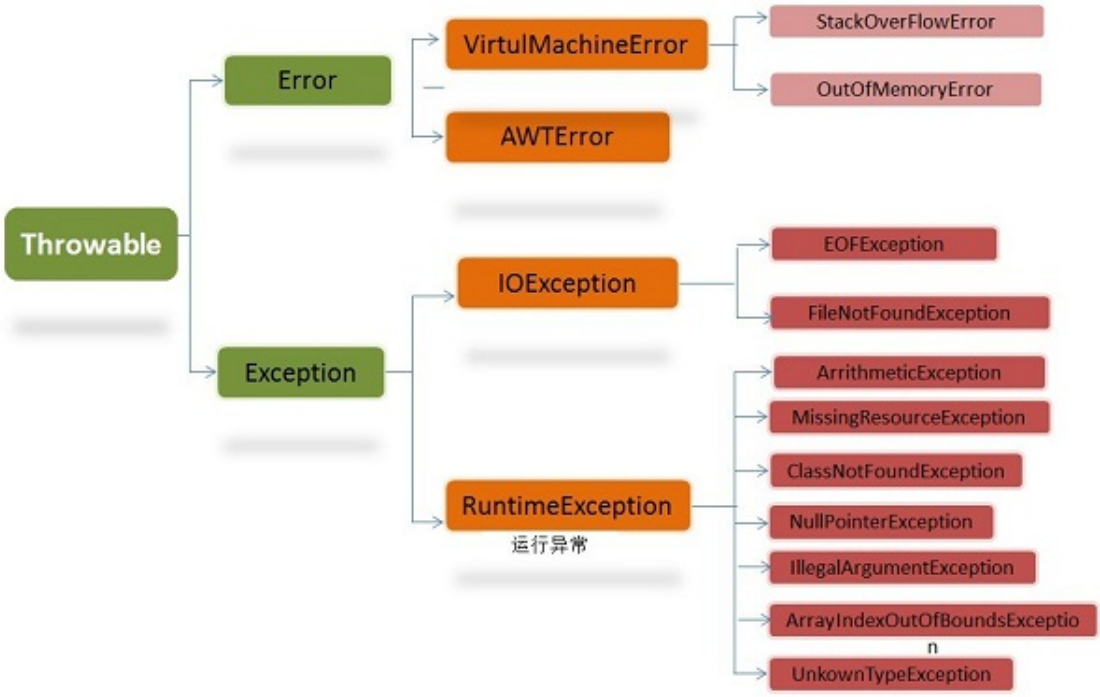
# 异常

## Throwable 类

> Throwable 类是Java语言中所有错误和异常的顶级父类，直接子类为 Error 和 Exception




## 构造方法

```
// 无参构造：构造一个新的 throwable 对象，其详细信息为null
public Throwable()

// 有参构造：使用指定的详细信息(message)构造一个新的 throwable 对象
public Throwable(String message)
```

## 成员方法

```
// 返回此 throwable 对象的详细信息字符串
public String getMessage()

// 返回一个简要信息描述
public String toString()

// 打印此 throwable 对象及其详细信息字符串到标准错误流(控制台)
public void printStackTrace()
```

### 案例代码

```
public class Test {
    public static void main(String[] args) {
        Throwable throwable1 = new Throwable();

```

```
5
   System.out.println(throwable1.getMessage())
   ;
6
   System.out.println(throwable1.toString());
7          throwable1.printStackTrace();
8
9          Throwable throwable2 = new
   Throwable("错误信息！！！");
10
11
   System.out.println(throwable2.getMessage())
   ;
12
   System.out.println(throwable2.toString());
13         throwable2.printStackTrace();
14     }
15 }
```

## Error

> Error 类是 Throwable 的子类，它指出了一个合理的应用程序不应该试图捕捉的严重问题
>
> 是程序中无法处理的错误，表示运行应用程序中出现了严重的错误。此类错误一般表示代码运行时JVM出现问题。通常有Virtual MachineError（虚拟机运行错误）、NoClassDefFoundError（类定义错误）等。比如说当jvm耗完可用内存时，将出现

> OutOfMemoryError。此类错误发生时，JVM将终止线程。非代码性错误。因此，当此类错误发生时，应用不应该去处理此类错误。

【注意】Error 结尾的是严重问题，无法解决

## Exception

> Exception 类及其子类是一种 Throwable 的子类，指示了一个合理的应用程序可能想要捕获的条件。

【注意】Exception 结尾的是我们可以处理的，一般我们需要关注的是 RuntimeException

# 六种常见的异常

ClassCastException(类转换异常)、IndexOutOfBoundsException(数组越界异常)、NullPointerException(空指针异常)、ArrayStoreException(数据存储异常，操作数组时类型不一致)、IOException(IO异常)、IllegalAccessException(安全权限异常，没有访问权限)

# 异常处理

> 异常处理分为两种：捕获和抛出
>
> **有能力处理就进行捕获，没有就抛出**

# 捕获

```
格式：
    try {
        // 有可能出现问题的代码，存在一定隐患的代码
    } catch (异常类型 变量名) {
        // 对应当前异常类型的处理方式
    } finally {
        // 无论是否捕获，都会执行，常用于释放资源
    }
```

## 案例代码

```
public class Test {
    public static void main(String[] args)
{
        int num1 = 0;
        int num2 = 20;

        int ret = 0;

        /*
         * 除数不能为0
         */
        try {
            ret = num2 / num1;
        } catch (ArithmeticException e) {
            e.printStackTrace();
        }

```

```
17              System.out.println(ret);
18          }
19      }
```

结果

```
1   java.lang.ArithmeticException: / by zero
2       at
    code.exception.Test.main(Test.java:11)
3   0
```

- 体会1：使用try-catch-finally处理编译时异常，是让程序在编译时就不再报错，但是运行时仍然有可能报错。相当于我们使用try-catch将一个编译时可能出现的异常，延迟到运行时出现。
- 体会2：在开发中，运行时异常比较常见，此时一般不用try-catch去处理，因为处理和不处理都是一个报错，最好办法是去修改代码。针对编译时异常，我们一定要考虑异常处理。

【注意】

```
1、代码中出现异常，JVM会终止代码运行，如果使用try catch捕获处理异常，JVM会认为当前代码中不存在异常，可以继续运行。

2、try - catch代码块中声明的都是局部变量，需要提前声明

3、try - catch捕获处理异常，可以处理多种异常情况

4、代码中存在多种隐患，存在多个异常情况，try - catch捕获有且只能处理第一个出现异常的代码，因为JVM从异常代码开始直接进入异常捕获阶段

5、Exception作为Java中所有异常的超类，在捕获异常处理时如果直接使用Exception进行捕获处理，无法具体到对某一个异常来处理

6、Exception可以作为try - catch 最后一个，用于处理其他异常捕获之后没有对症方式遗留问题
```

## 抛出

```
关键字：
    throw
        在方法内特定条件下抛出指定异常，后面跟异常类型的对象

    throws
        在【方法声明】位置，告知调用者，当前方法有哪些异常抛出，后面跟异常的类型

    用于处理非当前方法操作问题，导致出现的异常，一般情况下是用于处理方法运行过程中因为参数传入，参数处理，运算结果导致的问题，抛出异常。

    throw是一个稍微高级的参数合法性判断
```

## 案例代码

```
public class TestThrows {
    public static void main(String[] args) {

        try {
            test(1, 0);
        } catch (ArithmeticException e) {
            e.printStackTrace();
        }
    }

    /**
     * 测试方法，打印两个数的差
```

```
13          *
14          * @param num1 第一个参数，被除数
15          * @param num2 第二个参数，除数
16          * @throws ArithmeticException 如果除数为
    0，抛出异常
17          */
18         public static void test(int num1, int
    num2) throws ArithmeticException {
19
20             if (0 == num2) {
21                 throw new
    ArithmeticException("除数不能为0");
22             }
23
24             System.out.println(num1 / num2);
25         }
26     }
```

结果

```
1   java.lang.ArithmeticException: 除数不能为0
2       at
    code.exception.TestThrows.test(TestThrows.ja
    va:23)
3       at
    code.exception.TestThrows.main(TestThrows.ja
    va:7)
```

- 体会：tyr-catch-fianlly:真正的将异常给处理掉了。throws的方式只是将异常抛给了方法的调用者，并没

有真正将异常处理掉。

【注意】

```
1、throw 和 throws 必须同时出现，并且体现在注释上

2、代码如果运行到throw抛出异常，之后的代码不再运行，之后的代码是成为无法抵达的代码

3、代码中存在多种隐患，按照隐含的情况，分门别类处理，不能在同一个条件内抛出两个异常。并且在方法的声明位置，throws之后，不同的异常，使用逗号隔开

4、当调用带有异常抛出的方法时，对于方法抛出的异常，有两种处理方式，可以捕获处理，也可以抛出处理。
```

# 异常分类

## 运行时异常

```
RuntimeException：代码运行过程中出现的异常，没有强制处理的必要性，因为JVM会处理RuntimeException异常，即报错
NullPointerException（空指针异常）：指针指向的对象为空（null）
ArrayIndexOutOfBoundException（数组角标越界异常）StringIndexOutOfBoundException（字符串越界异常）...
ClassCastException（类型转换异常）
```

可能在java虚拟机正常工作时抛出的异常。java提供了两种异常机制。一种是运行时异常(RuntimeExepction)，一种是检查式异常(checked execption)。

检查式异常：我们经常遇到的IO异常及sql异常就属于检查式异常。对于这种异常，java编译器要求我们必须对出现的这些异常进行catch 所以 面对这种异常不管我们是否愿意，只能自己去写一堆catch来捕捉这些异常。

运行时异常：我们可以不处理。当出现这样的异常时，总是由虚拟机接管。比如：我们从来没有人去处理过NullPointerException异常，它就是运行时异常，并且这种异常还是最常见的异常之一。

## 编译时异常

```
1  强制要求处理，不管是捕获处理还是抛出处理，都需要进行操作，如果未处理就会报错
2  IOException(IO异常)
3      FileNotFoundException
4  ClassNotFoundException
```

# 自定义异常

> 继承自 Exception 或者 RuntimeException，只需要提供无参构造和一个带参构造即可
>
> 【注意】如果继承自Exception，调用抛出了此异常的方法需要手动对其捕获或者抛出

## 自定义异常类

```
public class MyException extends Exception {
    public MyException() {}

    public MyException(String message) {
        super(message);
    }
}
```

## 测试

```
public class Test {
    public static void main(String[] args)
{
        try {
            buy(false);
        } catch (MyException e) {
            e.printStackTrace();
        }
    }

    /**
     * 买方法，当没有女朋友的时候，抛出异常
     * @param hasGirlFriend boolean类型，是否
有女朋友
     * @throws MyException 自定义单身狗异常
     */
    public static void buy(boolean
hasGirlFriend) throws MyException {
```

```
16          if (false == hasGirlFriend) {
17              throw new MyException("你还没有女
    朋友");
18          }
19
20          System.out.println("买一送一");
21      }
22  }
```

结果

```
1  code.myexception.MyException: 你还没有女朋友
2      at
   code.myexception.Test.buy(Test.java:19)
3      at
   code.myexception.Test.main(Test.java:6)
```

## 注意

```
1  1、父类的方法有异常抛出，子类的重写方法在抛出异常的时
   候必须要小于等于父的异常，如果这个方法抛出的异常范围
   大于父类SuperClass所抛出的异常的话,那么在display
   方法中对异常的catch处理就会catch不到这个异常.
2
3  2、父类的方法没有异常抛出，子类的重写方法不能有异常抛
   出
4
5  3、父类的方法抛出多个异常，子类的重写方法必须比父少或
   者小
```

# 总结

1、异常的存在是非常有必要的，可以帮助我们定位和解决问题

2、异常的处理方式：捕获和抛出

3、自定义异常一定要有无参构造和有参构造

4、后期会遇到大量的异常，一般使用快捷键就能搞定

Eclipse： Ctrl + 1

Idea： Alt + Enter