1 内部类

1.1 内部类的概述

将类定义在另一个类的内部则成为内部类。其实就是类定义的位置发生了变化。

```
class  Outer
{

    int num = 10;

    class Inner{
        //属性
        int age = 10;
        //内部类定义函数
        public void print(){
            System.out.println("内部类...");
        }

    }

    public void show(){
        System.out.println("外部类....")
    }

}
```

在一个类中，定义在类中的叫成员变量，定义在函数中的叫成员函数，那么根据类定义的位置也可以分为**成员内部类**和**局部内部类**。

备注：内部类生产的 class 文件为 “外部类$内部类”，为了标明该内部类是属于具体哪个外部类的。

1.2 成员内部类的访问方式

1. 内部类可以直接访问外部类的成员属性。(孙悟空相当于内部类飞到牛魔王的肚子里面去)。
2. 外部类需要访问内部类的成员属性时需要创建内部类的对象。
    1. 在外部类的成员函数中创建内部类的对象，通过内部类对象对象直接访问内部类的成员。
    2. 在其他类中直接创建内部类的对象。

        Outer.Inner inner = new Outer().new Inner();

```
class  Outer
{

     int num = 10;

     class Inner{
          //内部类定义函数
          public void print(){
               //内部类可以直接访问外部类
               System.out.println(num);
          }

     }

     public void show(){
          System.out.println("外部类....");
     }

}
```

外部类访问内部类的属性

```
class  Outer
{

    int num = 10;

    class Inner{

         int count = 20;

         //内部类定义函数
         public void print(){
              //内部类可以直接访问外部类
              System.out.println(num);
         }

    }

    public void show(){
         System.out.println("访问内部类的属性"+count);
    }


    public static void main(String[] args){
         new Outer().show();
    }

}
```

```
位置: 类 Outer
                System.out.println("访问内部类的属性"+count);
                                                    ^
1 错误

G:\0226\day02>
```

编译异常分析：外部类需要访问内部类的属性时，需要创建内部类的对象访问。

```
class  Outer
{

    int num = 10;

    class Inner{

        int count = 20;

        //内部类定义函数
        public void print(){
            //内部类可以直接访问外部类
            System.out.println(num);
        }

    }

    public void show(){
        System.out.println("访问内部类的属性"+new Inner().count);
    }


    public static void main(String[] args){
        new Outer().show();
    }

}
```

有 A 类和 B 类，当 A 类想要直接访问 B 类中的成员，而 B 类又需要建立 A 类的对象来 A 类中的成员。这时，就将 A 类定义成 B 类的内部类。比喻：孙悟空和铁扇公主。孙悟空到了公主肚子中，就成了内部类（其实是持有外部类的对象引用）。

疑问： 什么时候使用内部类呢？

当我们分析事物时，发现事物的内部还有具体的事物，这时则应该定义内部类了。

比如人体是一个类，人体有心脏，心脏的功能在直接访问人体的其他内容。这时就将心脏定义在人体类中，作为内部类存在。

内部类的优势：成员内部类作为外部类的成员，那么可以访问外部类的任意成员。

```
class Body{

    class Heart{

        public Heart getHeart(){
            return new Heart();
        }
    }

}
```

1.3 成员内部类访问细节

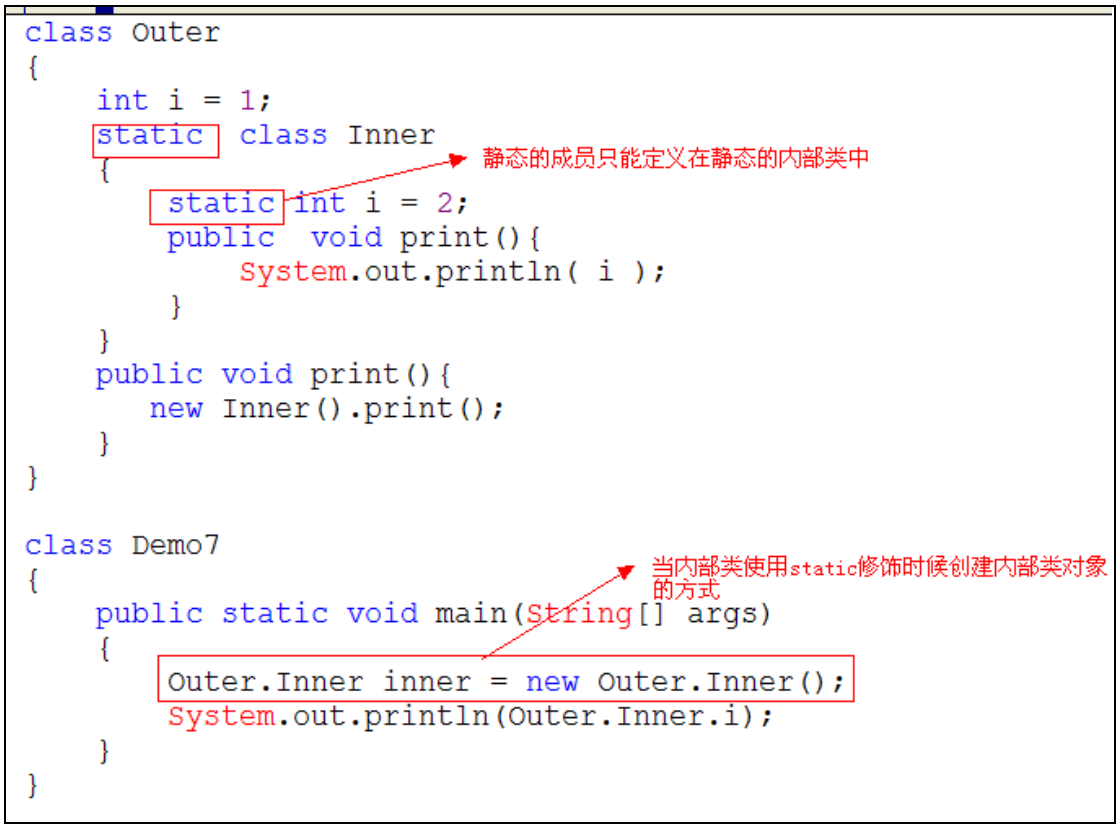
```java
class Outer
{
    int i = 1;
    private class Inner
    {
        int i = 2;
        public  void print(){
            System.out.println( i );
        }
    }
    public void print(){
       new Inner().print();
    }
}

class Demo7
{
    public static void main(String[] args)
    {
        Outer.Inner inner = new Outer().new Inner();
        inner.print();
    }
}
```

私有的类，只能在当前Outer类可见。

```
G:\0226\day06>javac Demo7.java
Demo7.java:21: Outer.Inner 可以在 Outer 中访问 private
                Outer.Inner inner = new Outer().new Inner();
                      ^
Demo7.java:21: Outer.Inner 可以在 Outer 中访问 private
                Outer.Inner inner = new Outer().new Inner();
                                                ^
```

私有的成员内部类不能在其他类中直接创建内部类对象来访问。



如果内部类中包含有静态成员，那么 java 规定内部类必须声明为静态的访问静态内部类的形式：Outer.Inner in = new Outer.Inner();

总结：成员内部类(成员属性、成员方法)特点：

1. **私有的成员内部类**
   特点：不能在其他类中直接创建内部类对象来访问
2. **静态的成员内部类**
   特点：如果内部类中包含有静态成员，那么 java 规定内部类必须声明为静的访问静态内部类的形式：
   Outer.Inner in = new Outer.Inner();

```
class Outer{

        int num = 10;

    class Inner{

        int num = 20;

        public void show(){

            System.out.println("show num "+num);   //num=20
        }

    }
}

class InnerClassDemo3
{
    public static void main(String[] args)
    {
        Outer.Inner inner = new Outer().new Inner();
        inner.show();
    }
}
```

疑问： 目前打印的 num 是 20，如果想打印 10 的话，应该怎么做？

解答：这时候其实在 show 方法中已经存在了两个 this 对象，一个是外部类对象，一个是内部类对象，所以要在 this 前面加上类名标明对应的 this。

1.4 局部内部类

局部内部类概述：包含在外部类的函数中的内部类称之为局部内部类。

访问：可以在包含局部内部类的方法中直接创建局部内部类的对象调用局部内部类的成员。

注意：局部内部类只能访问所在函数的 fanal 属性。

```
/*
内部类定义在类中的局部位置.

内部类定义局部位置上，只能访问该局部中的被final修饰的常量

*/
class Outer
{
    int num = 4;
    class MyIn
    {
    }
    public void  method(final int y)
    {
        final int x = 3;
        class Inner
        {
            void show()
            {
                System.out.println("show run -"+Outer.this.num);
                System.out.println("x="+x);
                System.out.println("y="+y);
            }
        }

        new Inner().show();
    }
}
class InnerClassDemo4
{
    public static void main(String[] args)
    {
        new Outer().method(5);
    }
}
```

1.5 匿名内部类

匿名内部类：就是没有类名字的内部类。
匿名内部类作用：简化内部类书写。
匿名内部类的前提:必须继承一个父类或者是实现一个接口。
匿名内部类的格式：

new 父类或者接口(){ 执行代码….};

内部类的写法：

```
class Outer{
	class Inner
	{
		public void show(){
			System.out.println("内部类的 show 方法");
		}
	}
	public void print(){
		new Inner().show();
	}
}
```

匿名内部类调用 show 方法:

```
abstract class Inner{
    abstract void show();
}


class Outer{
    /*
    class Inner
    {
        public void show(){
            System.out.println("内部类的show方法");
        }
    }
    */
    public void print(){
        new Inner(){
         public void show(){
                System.out.println("内部类的show方法");
            }
        }.show();
    }
}
```

案例：在外部类调用 show1、show2 方法。内部类的实现方法/

```
class Outer
{
    class  Inner
    {
        public void show1(){
            System.out.println("show1");
        }

        public void show2(){
            System.out.println("show2");
        }
    }
    //要求在print方法中调用show1和show2两个方法
    public void print(){
        Inner inner = new Inner();
        inner.show1();
        inner.show2();
    }

}
```

使用匿名内部类实现：

```java
interface Inner{
    public void show1();
    public void show2();
}


class Outer
{
    Inner inner = new Inner()
    {
        public void show1(){
            System.out.println("show1");
        }

        public void show2(){
            System.out.println("show2");
        }
    }
    //要求在print方法中调用show1和show2两个方法
    public void print(){
        inner.show1();
        inner.show2();
    }

}
```

注意细节：

1.使用匿名内部类时，如果需要调用匿名内部类的两个方法或者两个方法以上。可以使用变量指向该对象。

2 异常

2.1 现实生活的病

现实生活中万物在发展和变化会出现各种各样不正常的现象。

1：例如：人的成长过程中会生病。

|—病

|—不可治愈(癌症晚期)

|—可治愈

|—小病自行解决(上火,牙痛)

|—去医院(感冒,发烧)

2.2 java 异常体系图

现实生活中的很多病况从面向对象的角度考虑也是一类事物，可以定义为类。
java 中可以通过类对这一类不正常的现象进行描述，并封装为对象。

1. java 的异常体系包含在 java.lang 这个包默认不需要导入。
2. java 异常体系

```
|——Throwable　（实现类描述 java 的错误和异常）
    |——Error　（错误）一般不通过代码去处理。
    |——Exceprion　（异常）
        |——RuntimeException　（运行时异常）
        |——非运行时异常
```

常见的 Error

```
/*
    java虚拟机默认管理了64M的内存,一下数组需要1G的内存
    这样子会造成内存溢出
*/
byte[] buf = new byte[1024*1024*1024]; //1k -->1M--->1G
System.out.println(buf);
```

```
G:\0226\day05>java Demo4
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at Demo4.main(Demo4.java:5)
```

错误原因：内存溢出。需要的内存已经超出了 java 虚拟机管理的内存范围。

```
Exception in thread "main" java.lang.NoClassDefFoundError: Demo5
Caused by: java.lang.ClassNotFoundException: Demo5
        at java.net.URLClassLoader$1.run(URLClassLoader.java:200)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:188)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:307)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:301)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:252)
        at java.lang.ClassLoader.loadClassInternal(ClassLoader.java:320)
Could not find the main class: Demo5.  Program will exit.
```

错误原因：找不到类文件。

错误(Error)：

它指的是一个合理的应用程序不能截获的严重的问题。大多数都是反常的情况。错误是 JVM 的一个故障(虽然它可以是任何系统级的服务)。所以，错误是很难处理的，一般的开发人员(当然不是你)是无法处理这些错误的。比如内存溢出.

3. 异常体系图的对应

```java
class ExceptionDemo1
{

    public void div(int a , int b){
        int c = a/b;
        System.out.println("~~~~~"+c);
    }


    public static void main(String[] args)
    {
        ExceptionDemo1 demo1 = new ExceptionDemo1();
        demo1.div(4,0);
    }
}
```

```
toString: java.lang.ArithmeticException: / by zero
java.lang.ArithmeticException: / by zero
        at ExceptionDemo4.div(ExceptionDemo4.java:4)
        at ExceptionDemo4.main(ExceptionDemo4.java:13)
程序出现了异常....
程序正常输出...
```

```
G:\0226\day02>java ExceptionDemo
Exception in thread "main" java.lang.NoClassDefFoundError: ExceptionDemo
Caused by: java.lang.ClassNotFoundException: ExceptionDemo
        at java.net.URLClassLoader$1.run(Unknown Source)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(Unknown Source)
        at java.lang.ClassLoader.loadClass(Unknown Source)
        at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)
        at java.lang.ClassLoader.loadClass(Unknown Source)
        at java.lang.ClassLoader.loadClassInternal(Unknown Source)
Could not find the main class: ExceptionDemo.  Program will exit.
```

2.3 Throwable 类

1. toString() 输出该异常的类名。
2. getMessage() 输出异常的信息，需要通过构造方法传入异常信息（例如病态信息）。
3. printStackTrace() 打印栈信息。

人生病：流鼻涕，感冒，呼吸道感染，肺炎。。。最后体现的是肺炎。

医生要处理需要获知这些信息。从外到里处理。最后找病源

```java
/*
 Throwable类

 printStackTrace() 打印栈信息

 肺炎
 上呼吸道感染
 发烧
 流鼻涕感冒
 小感冒
 */
```

```
class Demo6 {

    public static void main(String[] args) {

        // Throwable able=new Throwable();
        Throwable able = new Throwable("想吐。。。");
        System.out.println(able.toString()); // 输出该异常的类名
        System.out.println(able.getMessage()); // 输出异常的信息
        able.printStackTrace(); // 打印栈信息
    }
}
```

2.4 程序中的异常处理

1. 当除数是非 0，除法运算完毕，程序继续执行。
2. 当除数是 0，程序发生异常，并且除法运算之后的代码停止运行。因为程序发生异常需要进行处理。

```
class Demo7 {

    public static void main(String[] args) {
        div(2, 0);
        System.out.println("over");
    }

    public static void div(int x, int y) {
        //该行代码的y值可能是0,程序会出现异常并停止
        System.out.println(x / y);
        System.out.println("除法运算");
    }
}
//ArithmeticException
```

疑问： 出现异常如何处理？

2.4.1 自行处理

1. try{//可能发生异常的代码 }catch(异常类 变量名){//处理}。
2. 案例除法运算的异常处理。
3. 如果没有进行 try catch 处理，出现异常程序就停止。进行处理后，程序会继续执行。

```
class Demo7 {

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

```
        div(2, 0);
        System.out.println("over");
    }

    public static void div(int x, int y) {

        try {
            System.out.println(x / y); // 可能出现异常的语句，放入try中。
        } catch (ArithmeticException e) { // 进行异常匹配，
            //异常信息
            System.out.println(e.toString());
            System.out.println(e.getMessage());
             e.printStackTrace();
            System.out.println("除数不能为0");
        }
        System.out.println("除法运算");
    }
}
```

多个异常

1. 案例 print 方法，形参中增加数组。
2. 在 print 方法中操作数组会发生新的异常 ArrayIndexOutOfBoundsException，NullPointerException），如何处理？
    1. 使用将可能发生异常的代码放入 try 语句中，添加多个 catch 语句即可。
    2. 可以处理多种异常，但是同时只能处理一种异常。
    3. try 中除 0 异常和数组角标越界同时出现，只会处理一种。

```
public class Demo8 {

    public static void main(String[] args) {

        System.out.println();
        int[] arr = { 1, 2 };
        arr = null;

        // print (1, 0, arr);
        print (1, 2, arr);

        System.out.println("over");
    }

    public static void print(int x, int y, int[] arr) {

        try {
```

```
            System.out.println(arr[1]);
            System.out.println(x / y);
        } catch (ArithmeticException e) {
            e.toString();
            e.getMessage();
            e.printStackTrace();
            System.out.println("算术异常。。。");
        } catch (ArrayIndexOutOfBoundsException e) {
            e.toString();
            e.getMessage();
            e.printStackTrace();
            System.out.println("数组角标越界。。。");
        } catch (NullPointerException e) {
            e.toString();
            e.getMessage();
            e.printStackTrace();
            System.out.println("空指针异常。。。");
        }
        System.out.println("函数执行完毕");
    }
}
```

总结

1. 程序中有多个语句可能发生异常，可以都放在 try 语句中。并匹配对个 catch 语句处理。
2. 如果异常被 catch 匹配上，接着执行 try{}catch(){} 后的语句。没有匹配上程序停止。
3. try 中多个异常同时出现，只会处理第一条出现异常的一句，剩余的异常不再处理。
4. 使用多态机制处理异常。
    1. 程序中多态语句出现不同异常，出现了多个 catch 语句。简化处理（相当于急诊）。
    2. 使用多态，使用这些异常的父类进行接收。（父类引用接收子类对象）

```
public static void div(int x, int y, int[] arr, Father f) {

        try {
            System.out.println(arr[1]); // 数组越界
            System.out.println(x / y); // 除零
            Son s = (Son) f; // 类型转换

        } catch (Exception e) {
            e.toString();
            e.getMessage();
            e.printStackTrace();
```

```
            System.out.println("出错啦");
        }
        System.out.println("函数执行完毕");
    }
```

多个 catch 语句之间的执行顺序。

1. 是进行顺序执行，从上到下。
2. 如果多个 catch 内的异常有子父类关系。
    1. 子类异常在上，父类在最下。编译通过运行没有问题
    2. 父类异常在上，子类在下，编译不通过。(因为父类可以将子类的异常处理，子类的 catch 处理不到)。
    3. 多个异常要按照子类和父类顺序进行 catch

```
class Father {

}

class Son extends Father {

}

public class Demo8 {

    public static void main(String[] args) {

        System.out.println();
        int[] arr = { 1, 2 };
        arr = null;
        Father f = new Father();
        div(1, 0, arr, f);

        System.out.println("over");
    }

    public static void div(int x, int y, int[] arr, Father f) {

        try {
            System.out.println(arr[1]);
            System.out.println(x / y);
            Son s = (Son) f;

        } catch (ArithmeticException e) {
            e.toString();
            e.getMessage();
            e.printStackTrace();
```

```
            System.out.println("算术异常。。。");
        } catch (ArrayIndexOutOfBoundsException e) {
            e.toString();
            e.getMessage();
            e.printStackTrace();
            System.out.println("数组角标越界。。。");
        } catch (NullPointerException e) {
            e.toString();
            e.getMessage();
            e.printStackTrace();
            System.out.println("空指针异常。。。");
        } catch (Exception e) {
            e.toString();
            e.getMessage();
            e.printStackTrace();
            System.out.println("出错啦");
        }
        System.out.println("函数执行完毕");
    }
}
```

总结

处理异常应该 catch 异常具体的子类，可以处理的更具体，不要为了简化代码使用异常的父类。

疑惑：感觉异常没有作用.

```
/*
    应用场景：从配置文件中读取数据库的用户名和密码
*/
import java.io.*;
class Demo9
{
    public static void main(String[] args) throws Exception
    {
        //System.out.println("Hello World!");
        read();
    }


    public static void read() throws Exception{
        FileInputStream in = null;
        try{                            //该文件路径是由用户输入
            in = new FileInputStream("E:\\a.txt"); //获取文件的输入流
        }catch(Exception    e){
            //如果找不到用户输出的文件,那么读取系统默认的配置文件
            in = new FileInputStream("E:\\b.txt");
        }
        int data = 0;
        while((data=in.read())!=-1){ //不断读取文件
            System.out.println((char)data);
        }
    }
```

2.4.2 抛出处理

定义一个功能，进行除法运算例如（div(int x,int y)）如果除数为 0，进行处理。功能内部不想处理，或者处理不了。就抛出使用 throw new Exception("除数不能为 0")；进行抛出。抛出后需要在函数上进行声明，告知调用函数者，我有异常，你需要处理如果函数上不进行 throws 声明，编译会报错。例如：未报告的异常 java.lang.Exception；必须对其进行捕捉或声明以便抛出 throw new Exception("除数不能为 0")；

```
public static void div(int x, int y) throws Exception { // 声明异常，通知方法调用者。

        if (y == 0) {
        throw new Exception("除数为0"); // throw关键字后面接受的是具体的异常的对象
        }
        System.out.println(x / y);
        System.out.println("除法运算");
    }
```

5：main 方法中调用除法功能

调用到了一个可能会出现异常的函数，需要进行处理。

1：如果调用者没有处理会编译失败。

如何处理声明了异常的函数。

1：try{}catch(){}

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

        try {
            div(2, 0);
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("over");

    }

    public static void div(int x, int y) throws Exception { // 声明异常，通知方法调用者。

        if (y == 0) {
          throw new Exception("除数为0"); // throw关键字后面接受的是具体的异常的对象
        }
```

```
        System.out.println(x / y);
        System.out.println("除法运算");
    }
}
```

2：继续抛出 throws

```
class Demo9 {

    public static void main(String[] args) throws Exception {
        div(2, 0);
        System.out.println("over");
    }

     public static void div(int x, int y) throws Exception { // 声明异常，通知方法调用者。
    if (y == 0) {
        throw new Exception("除数为0"); // throw关键字后面接受的是具体的异常的对象
    }

        System.out.println(x / y);
        System.out.println("除法运算");
    }
}
```

throw 和 throws 的区别

1. 相同：都是用于做异常的抛出处理的。
2. 不同点：
    1. 使用的位置：throws 使用在函数上，throw 使用在函数内
    2. 后面接受的内容的个数不同：
        1. throws 后跟的是异常类，可以跟多个，用逗号隔开。
        2. throw 后跟异常对象。

```
//throws 处理
public static void main(String[] args) throws InterruptedException {
        Object obj = new Object();
        obj.wait();

    }
```

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

        //try catch 处理
        Object obj = new Object();
```

```
        try {
            obj.wait();
        } catch (InterruptedException e) {

            e.printStackTrace();
        }

    }
```

总结

1. try 语句不能单独存在，可以和 catch、finally 组成 try...catch...finally、try...catch、try...finally 三种结构。
2. catch 语句可以有一个或多个，finally 语句最多一个，try、catch、finally 这三个关键字均不能单独使用。
3. try、catch、finally 三个代码块中变量的作用域分别独立而不能相互访问。如果要在三个块中都可以访问，则需要将变量定义到这些块的外面。
4. 多个 catch 块时候，Java 虚拟机会匹配其中一个异常类或其子类，就执行这个 catch 块，而不会再执行别的 catch 块。（子类在上，父类在下）。
5. throw 语句后不允许有紧跟其他语句，因为这些没有机会执行。
6. 如果一个方法调用了另外一个声明抛出异常的方法，那么这个方法要么处理异常，要么声明抛出。

2.4.3 自定义异常

问题：现实中会出现新的病，就需要新的描述。
分析： java 的面向对象思想将程序中出现的特有问题进行封装。
案例： 定义功能模拟凌波登录。(例如：lb(String ip))需要接收 ip 地址

1. 当没有 ip 地址时，需要进行异常处理。
   1. 当 ip 地址为 null 是需要 throw new Exception("无法获取 ip");
   2. 但 Exception 是个上层父类，这里应该抛出更具体的子类。
   3. 可以自定义异常
2. 自定义描述没有 IP 地址的异常（NoIpException）。
   1. 和 sun 的异常体系产生关系。继承 Exception 类，自定义异常类名也要规范，结尾加上 Exception,便于阅读

```
/*
 自定义异常
 */
class NoIpException extends Exception {
```

```
    NoIpException() {

    }

    NoIpException(String message) {
        super(message);
    }

}

class Demo10 {

    public static void main(String[] args) throws NoIpException {

        System.out.println();
        String ip = "192.168.10.252";
        ip = null;
        try {
            Lb(ip);
        } catch (NoIpException e) {
            System.out.println("程序结束");
        }

    }

    /*
     *
     * 凌波教学
     */
    public static void Lb(String ip) throws NoIpException {
        if (ip == null) {
            // throw new Exception("没插网线吧，小白");
            throw new NoIpException("没插网线吧，小白");
        }

        System.out.println("醒醒了，开始上课了。");
    }
}
```

案例：模拟吃饭没带钱的问题

1. 定义吃饭功能，需要钱。（例如：eat(double money)）
2. 如果钱不够是不能吃放，有异常。
3. 自定义 NoMoneyException();继承 Exception 提供有参无参构造，调用父类有参构造初始化。at 方法进行判断，小于 10 块，throw NoMoneyException("钱不够

");

4. eat 方法进行声明，throws NoMoneyException
5. 如果钱不够老板要处理。调用者进行处理。try{}catch(){} 。

```
class NoMoneyException extends Exception {

    NoMoneyException() {

    }

    NoMoneyException(String message) {
        super(message);
    }
}

class Demo11 {

    public static void main(String[] args) {

        System.out.println();
        try {
            eat(0);
        } catch (NoMoneyException e) {
            System.out.println("跟我干活吧。");
        }
    }



    public static void eat(double money) throws NoMoneyException {
        if (money < 10) {
            throw new NoMoneyException("钱不够");
        }
        System.out.println("吃桂林米粉");
    }
}
```

2.5 运行时异常和非运行时异常

2.5.1 RuntimeException

RunntimeException 的子类：

ClassCastException

多态中，可以使用 Instanceof 判断，进行规避

ArithmeticException
进行 if 判断，如果除数为 0，进行 return
NullPointerException
进行 if 判断，是否为 null
ArrayIndexOutOfBoundsException
使用数组 length 属性，避免越界
这些异常时可以通过程序员的良好编程习惯进行避免的
1：遇到运行时异常无需进行处理，直接找到出现问题的代码，进行规避。
2：就像人上火一样牙疼一样，找到原因，自行解决即可
3：该种异常编译器不会检查程序员是否处理该异常
4：如果是运行时异常，那么没有必要在函数上进行声明。
6：案例
1：除法运算功能（div(int x,int y)）
2:if 判断如果除数为 0,throw new ArithmeticException();
3：函数声明 throws ArithmeticException
4：main 方法调用 div,不进行处理
5：编译通过，运行正常
6：如果除数为 0，报异常，程序停止。
7：如果是运行时异常，那么没有必要在函数上进行声明。
1：Object 类中的 wait()方法，内部 throw 了 2 个异常
IllegalMonitorStateException
InterruptedException
1：只声明了一个(throws) IllegalMonitorStateException 是运行是异常没有声明。

```
class Demo12 {

    public static void main(String[] args){
        div(2, 1);
    }

    public static void div(int x, int y) {
        if (y == 0) {
            throw new ArithmeticException();
        }
        System.out.println(x / y);
    }
}
```

2.5.2 非运行时异常(受检异常)

如果出现了非运行时异常必须进行处理 throw 或者 try{}catch(){}处理，否则编译器报错。

1；IOException　使用要导入包 import java.io.IOException;

2：ClassNotFoundException

2；例如人食物中毒，必须进行处理，要去医院进行处理。

3：案例

1：定义一测试方法抛出并声明 ClassNotFoundException（test()）

2：main 方法调用 test

3：编译报错

1：未报告的异常 java.lang.ClassNotFoundException;必须对其进行捕捉或声明以便抛出

```
public void isFile(String path){
        try
        {
            /*
            根据文件的路径生成一个文件对象，如果根据该路径找不到相应的文件，
            则没法生成文件对象。
            */
            File file = new File(path);
            //读取文件的输入流
            FileInputStream input = new FileInputStream(file);
            //读取文件
            input.read();
        }
        catch (NullPointerException e)
        {
            System.out.println("读取默认的文件路径..");
        }

    }
```

4：Sun 的 API 文档中的函数上声明异常，那么该异常是非运行是异常，调用者必须处理。

5：自定义异常一般情况下声明为非运行时异常

2：函数的重写和异常

1：运行时异常

1:案例定义 Father 类,定义功能抛出运行是异常,例如(test() throw ClassCastException)

2：定义 Son 类，继承 Father 类，定义 test 方法，没有声明异常

3：使用多态创建子类对象，调用 test 方法

4：执行子类方法

1：函数发生了重写，因为是运行时异常，在父类的 test 方法中，可以声明 throws 也可以不声明 throws

```
class Father {
    void test() throws ClassCastException { // 运行时异常
```

```
        System.out.println("父类");
        throw new ClassCastException();
    }
}

class Son extends Father {
    void test() {
        System.out.println("子类");
    }
}
class Demo14 {

    public static void main(String[] args) {
        Father f = new Son();
        f.test();
    }
}
```

2：非运行时异常

1：定义父类的 test2 方法，抛出非运行时异常，例如抛出 ClassNotFoundException

1：此时父类 test2 方法必须声明异常，因为是非运行时异常

2：Son 类定义 test2 方法，抛出和父类一样的异常，声明异常

3：使用多态创建子类对象，调用 test 方法，调用 test2 方法，

1：声明非运行时异常的方法，在调用时需要处理，所以在 main 方法调用时 throws

2：实现了重写，执行子类的 test2 方法

3：总结子类重写父类方法可以抛出和父类一样的异常，或者不抛出异常。

```
//  1 子类覆盖父类方法父类方法抛出异常，子类的覆盖方法可以不抛出异常
class Father {
    void test() throws ClassNotFoundException { // 非运行时异常
        System.out.println("父类");
        throw new ClassNotFoundException();
    }
}

class Son extends Father {
    void test() {
        System.out.println("子类");
        // 父类方法有异常，子类没有。
    }
}
```

```
class Demo14 {

    public static void main(String[] args) throws ClassNotFoundException
{
        Father f = new Son();
        f.test();

    }
}
```

4：子类抛出并声明比父类大的异常例如子类 test2 方法抛出 Exception

1：编译失败，无法覆盖

2：子类不能抛出父类异常的父类。

3：总结子类不能抛出比父类的异常更大的异常。

```
//2：子类覆盖父类方法不能比父类抛出更大异常
class Father {
    void test() throws Exception {
        // 非运行时异常
        System.out.println("父类");
        throw new Exception();
    }
}

class Son extends Father {
    void test() throws ClassNotFoundException { // 非运行时异常
        System.out.println("子类");
        throw new ClassNotFoundException();
    }
}
class Demo14 {

    public static void main(String[] args) throws Exception {
        Father f = new Son();
        f.test();

    }
}
```

3：总结

1：子类覆盖父类方法是，父类方法抛出异常，子类的覆盖方法可以不抛出异常，或者抛出父类方法的异常，或者该父类方法异常的子类。

2：父类方法抛出了多个异常，子类覆盖方法时，只能抛出父类异常的子集

3：父类没有抛出异常子类不可抛出异常

1：子类发生非运行时异常，需要进行 try{}catch 的 (){}处理，不能

抛出。

4：子类不能比父类抛出更多的异常

2.6 finally

1：实现方式一：

try{ // 可能发生异常的代码 } catch(异常类的类型 e){ // 当发生指定异常的时候的处理代码 }catch...

比较适合用于专门的处理异常的代码，不适合释放资源的代码。

2：实现方式二：

try{ } catch(){} finally{ // 释放资源的代码 }

finally 块是程序在正常情况下或异常情况下都会运行的。

比较适合用于既要处理异常又有资源释放的代码

3：实现方式三

try{ }finally{ // 释放资源 }

比较适合处理的都是运行时异常且有资源释放的代码。

4：finally:关键字主要用于释放系统资源。

1：在处理异常的时候该语句块只能有一个。

2：无论程序正常还是异常，都执行 finally。

5：finally 是否永远都执行？

1：只有一种情况，但是如果 JVM 退出了 System.exit(0)，finally 就不执行。

2：return 都不能停止 finally 的执行过程。

6：案例使用流

1：使用 FileInputStream 加载文件。

导包 import java.io.FileInputStream;

2：FileNotFoundException

导入包 import java.io.FileNotFoundException;

3：IOException

import java.io.IOException;

```
public class FinallyDemo {
    // 本例子使用finally 关闭系统资源。
    public static void main(String[] args) {

        FileInputStream fin = null;
        try {
            System.out.println("1创建io流可能出现异常");
            fin = new FileInputStream("aabc.txt"); // 加载硬盘的文本文件到内存，通过流
            // System.out.println(fin);
        } catch (FileNotFoundException e) {
```

```
            System.out.println("2没有找到abc.txt 文件");
            System.out.println("3catch 了");
            // System.exit(0);
            // return;
        }
        // finally
        finally {
            System.out.println("4fianlly执行");
            if (fin != null) { // 如果流对象为null 流对象就不存在，没有必要关闭资源
                try {
                    fin.close();
                } catch (IOException e) {
                    e.printStackTrace();
                    System.out.println("close 异常");
                }

            }
            System.out.println("5finally over");
        }
        System.out.println("6mainover");
    }
}

// 2：无论程序正常还是异常，都执行finally。 但是遇到System.exit(0); jvm退出。
// finally用于必须执行的代码， try{} catch(){}finally{}
// try{}finally{}
```

3 作业

1. 为什么要将一个类定义成内部类？
2. 匿名内部类的使用和细节(面试题)
3. 异常的思想和体系特点？
4. throws 和 throw 的如何使用？
5. 什么时候 try 什么时候 throws？
6. 编译时被检测异常和运行时异常的区别？
7. 异常的所有细节？
8. finally 的应用？
9. 包的作用，名称空间的定义和理解？
10. jar 包的基本使用。只要将类和包都存储到 jar 中，方便于使用。只要将 jar 配置到 classpath 路径下。