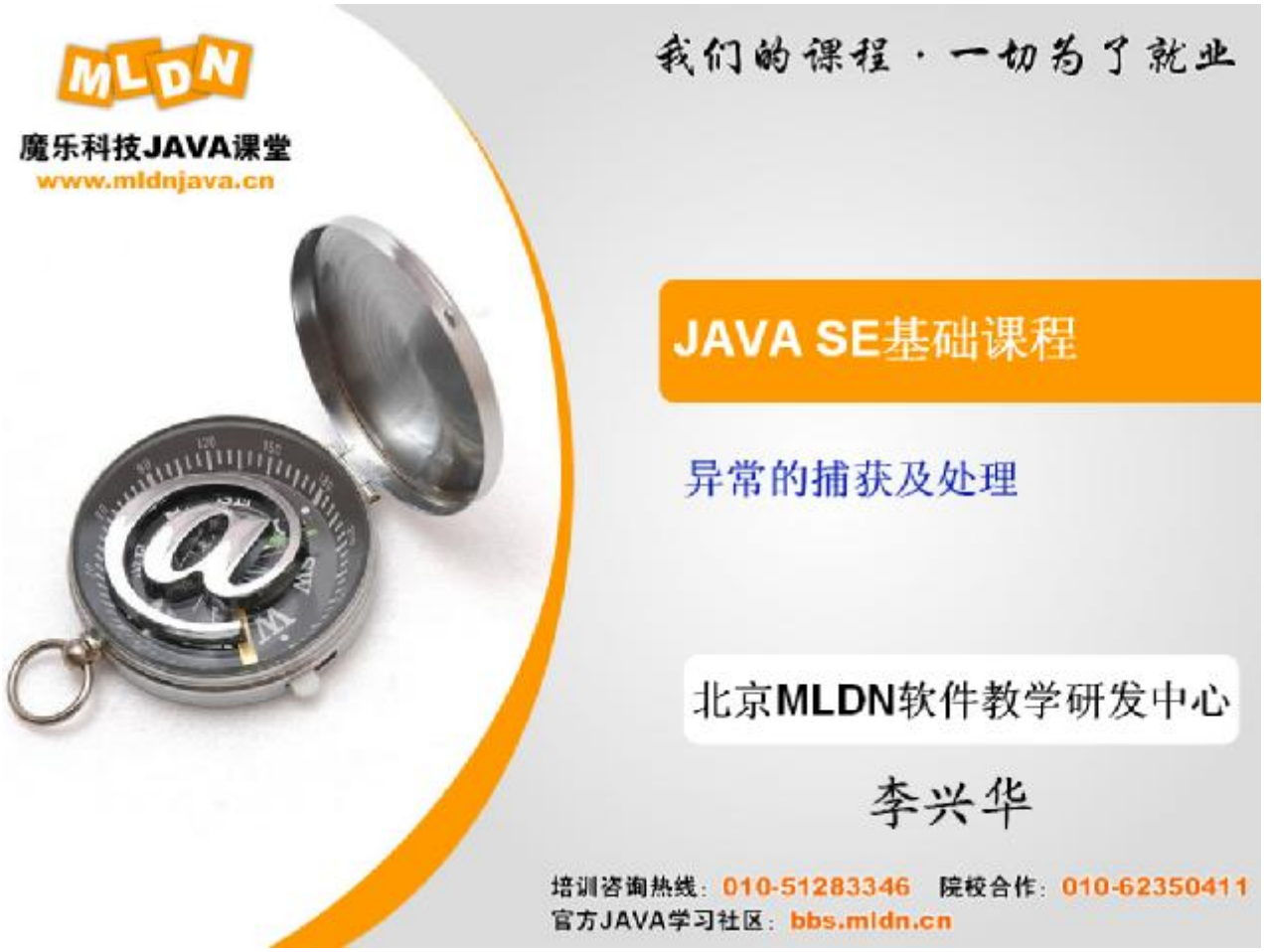
1、课程名称：异常的捕获及处理



2、知识点

2.1、上次课程的主要知识点

1、 抽象类和接口最大的特点是可以在类的基础上的进一步定义。
2、 一个类绝对不能继承一个已经实现好的类，而只能继承抽象类或实现接口。
3、 包装类：将基本数据类型包装成一个类的形式，称为包装类。
4、 匿名内部类：在抽象类和接口的基础之上发展起来的。
5、 JDK 1.5 的新特性：泛型、枚举、可变参数、foreach。

2.2、本次预计讲解的知识点

1、 异常的产生原因及处理格式

2、 异常的标准使用方式

3、 throw 和 throws 关键字的作用

3、具体内容（绝对重点）

3.1、认识异常

异常的定义：异常是导致一个程序中断的指令流，一旦出现之后程序就将立即退出，观察以下产生异常的程序：

```
public class ExpDemo01 {
    public static void main(String args[]){
        int x = 10 ;
        int y = 0 ;
        System.out.println("================ 计算开始 ================") ;
        System.out.println("计算结果是：" + (x/y)) ;
        System.out.println("================ 计算结束 ================") ;
    }
};
```

程序的运行结果：

```
================ 计算开始 ================
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ExpDemo01.main(ExpDemo01.java:6)
```

可以发现，一旦产生了异常之后，则在异常语句之后的代码将不再执行，而是直接退出程序。

为了保证程序中即使出现了异常之后，仍然可以继续执行的话，那么就需要使用异常处理语句了。

3.2、处理异常

在 Java 中使用如下的语法进行异常的处理：

```
try{
    可能出现异常的语句 ;
} catch(异常类型 异常对象){
    处理异常 ;
} catch(异常类型 异常对象){
    处理异常 ;
} ...
finally{
    异常处理的统一出口 ;
}
```

下面在以上的程序中，完成异常的处理操作，保证程序即使出现了异常也可以正确的执行完毕。

```
public class ExpDemo02 {
    public static void main(String args[]){
        int x = 10 ;
```

```
        int y = 0 ;
        System.out.println("================ 计算开始 ================") ;
        try{
            System.out.println("计算结果是：" + (x/y)) ;
            System.out.println("---------------------------------") ;
        }catch(ArithmeticException e){
            System.out.println("出现了异常：" + e) ;
        }
        System.out.println("================ 计算结束 ================") ;
    }
};
```

这个时候由于程序中增加了异常的处理机制，所以，此时的程序可以正常的执行完毕，当然，在异常的处理中也可以加入 finally 作为统一的出口操作。

```
public class ExpDemo03 {
    public static void main(String args[]){
        int x = 10 ;
        int y = 0 ;
        System.out.println("================ 计算开始 ================") ;
        try{
            System.out.println("计算结果是：" + (x/y)) ;
            System.out.println("---------------------------------") ;
        }catch(ArithmeticException e){
            System.out.println("出现了异常：" + e) ;
        }finally{
            System.out.println("不管是否有异常，都执行此语句！") ;
        }
        System.out.println("================ 计算结束 ================") ;
    }
};
```

但是以上的程序并不灵活，因为所有的操作的数字都是固定好的了，那么下面希望可以让这个数字由用户自己决定，所以，通过初始化参数传递所需要计算的数字。

```
public class ExpDemo04 {
    public static void main(String args[]){
        int x = 0 ;
        int y = 0 ;
        System.out.println("================ 计算开始 ================") ;
        try{
            x = Integer.parseInt(args[0]) ;
            y = Integer.parseInt(args[1]) ;
            System.out.println("计算结果是：" + (x/y)) ;
            System.out.println("---------------------------------") ;
        }catch(ArithmeticException e){
            e.printStackTrace() ;
        }catch(ArrayIndexOutOfBoundsException e){
```

```
            e.printStackTrace() ;
        }catch(NumberFormatException e){
            e.printStackTrace() ;
        }finally{
            System.out.println("不管是否有异常，都执行此语句！") ;
        }
        System.out.println("================ 计算结束 ================") ;
    }
};
```

此时，一个 try 语句之后跟上了多个 catch 语句，表示可以处理多种异常，但是这种语法也有问题，如果假设有很多异常并不知道的话呢？

3.3、异常的处理结构

以上的三个异常处理的操作类，以算术异常为例，观察其继承的结构：

```
java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ java.lang.RuntimeException
              └ java.lang.ArithmeticException
```

可以发现是 Exception 的子类，但是 Exception 本身又是 Throwable 子类，此类有两个子类：

- Error：表示 JVM 错误，还没有运行到程序上，所以一般不去处理。
- Exception：是在程序中发生的异常，一般都需要进行处理。

那么由于人们只关心程序中产生的异常，所以有时候会将以上的三个类都统一称为异常。

从之前学习过的对象多态性中可以发现，任何的子类对象都可以自动向父类对象转换。

那么，肯定所有的异常类的操作都可以通过 Exception 进行接收。

```
public class ExpDemo05 {
    public static void main(String args[]){
        int x = 0 ;
        int y = 0 ;
        System.out.println("================ 计算开始 ================") ;
        try{
            x = Integer.parseInt(args[0]) ;
            y = Integer.parseInt(args[1]) ;
            System.out.println("计算结果是：" + (x/y)) ;
            System.out.println("----------------------------------") ;
        }catch(ArithmeticException e){
            e.printStackTrace() ;
        }catch(ArrayIndexOutOfBoundsException e){
            e.printStackTrace() ;
        }catch(Exception e){
            e.printStackTrace() ;
        }finally{
```

```
            System.out.println("不管是否有异常，都执行此语句！") ;
        }
        System.out.println("================ 计算结束 ================") ;
    }
};
```

但是，如果一个 try 语句之中同时有多个 catch 的话一定要记住的是，捕获范围小的异常要放在捕获范围大的异常之前，一般在开发中为了简单起见，大部分的异常都直接使用 Exception 接收即可。

```
public class ExpDemo06 {
    public static void main(String args[]){
        int x = 0 ;
        int y = 0 ;
        System.out.println("================ 计算开始 ================") ;
        try{
            x = Integer.parseInt(args[0]) ;
            y = Integer.parseInt(args[1]) ;
            System.out.println("计算结果是：" + (x/y)) ;
            System.out.println("---------------------------------") ;
        }catch(Exception e){
            e.printStackTrace() ;
        }finally{
            System.out.println("不管是否有异常，都执行此语句！") ;
        }
        System.out.println("================ 计算结束 ================") ;
    }
};
```

但是，以上的代码只是在要求不严格的情况下使用的。

3.4、异常的处理流程

在异常的处理中基本上都是采用如下的过程完成的：

1、 每当一个异常产生之后，实际上都会自动生成一个异常类的实例化对象，如果此时编写了异常处理语句的话，则进行异常的处理，如果没有的话，则交给 JVM 进行处理。

2、 使用了 try 捕获异常之后，将自动与 catch 中的异常类型相匹配，如果匹配成功，则表示可以使用此 catch 处理异常，如果都没有匹配成功的，则不能处理。

3、 程序中不管是否出现了异常，如果存在了 finally 语句，都要执行此语句的代码。

3.5、throws 关键字

throws 关键字主要是用在方法的声明上，表示一个方法不处理异常，而交给被调用处进行处理。

```
class MyMath {
    public int div(int i,int j) throws Exception{
        return i / j ;
    }
```

```
};
public class ExpDemo07 {
    public static void main(String args[]){
        try{
            System.out.println(new MyMath().div(10,0)) ;
        }catch(Exception e){
            e.printStackTrace() ;
        }
    }
};
```

throws 关键字不光可以在普通的方法上使用，主方法上也可以使，如果在主方法上使用的话就表示一旦出现了异常之后，继续向上抛，表示由 JVM 进行处理了。

```
class MyMath {
    public int div(int i,int j) throws Exception{
        return i / j ;
    }
};
public class ExpDemo08 {
    public static void main(String args[]) throws Exception {
        System.out.println(new MyMath().div(10,0)) ;
    }
};
```

实际上默认情况下的所有异常都是依靠 JVM 进行处理的。

3.6、throw 关键字

throw 关键字是在程序中人为的抛出一个异常对象。

```
public class ExpDemo09 {
    public static void main(String args[]) {
        try{
            throw new Exception("抛着玩的！ ") ;
        }catch(Exception e){
            e.printStackTrace() ;
        }
    }
};
```

3.7、RuntimeException 和 Exception

如果将一个字符串变为整型数据类型，则可以使用 Integer 类中的 parseInt()方法完成。

- public static int parseInt(String s) throws NumberFormatException

本方法中存在了 throws 关键字的声明，理论上讲，在调用时必须进行异常处理，但是从实际的使用中可以发现，即使不使用 try…catch 处理也没问题。

```
public class ExpDemo10 {
    public static void main(String args[]) {
        int x = Integer.parseInt("10") ;
    }
};
```

要想解释这个问题，必须观察 NumberFormatException 异常的继承结构：

```
java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ java.lang.RuntimeException
              └ java.lang.IllegalArgumentException
                  └ java.lang.NumberFormatException
```

可以发现 NumberFormatException 是 RuntimeException 的子类，那么实际上在程序中，Java 为了异常的处理方便，定义出了一个特殊的异常类 —— RuntimeException，一旦抛出的异常是此类或者是此类的子类的话，那么可以不用进行异常处理的操作，如果不做任何异常处理的话，则一旦出现了异常之后将交给被调用处进行处理。

RuntimeException 和 Exception 的区别，Exception 必须处理，而 RuntimeException 可以不用处理。

3.8、异常处理的标准格式

之前的异常的处理语句：try…catch…finally、throw、throws 实际上在开发中是要一起使用的，以下面的程序为例。

编写一个除法的操作，要求在计算开始的时候输出“开始计算”的信息，在计算完成之后，输出“结束计算”的信息，而且只要是出现了异常一定要交给被调用处处理。

```
class MyMath {
    public int div(int i,int j) throws Exception{
        System.out.println("================ 计算开始 ================") ;
        int temp = 0 ;
        try{
            temp = i / j ;
        }catch(Exception e){
            throw e ;  // 向上抛
        }finally{
            System.out.println("================ 计算结束 ================") ;
        }
        return temp ;
    }
};
public class ExpDemo11 {
    public static void main(String args[]){
        try{
            System.out.println(new MyMath().div(10,0)) ;
        }catch(Exception e){
            e.printStackTrace() ;
        }
```

```
    }
};
```

本道程序代码结构是固定的，而且也是一个标准的处理结构。

3.9、assert 关键字（了解）

assert 关键字是在 JDK 1.4 之后增加的新关键字，表示断言。

一般 assert 都在程序中判断某条语句一定是期望的计算结果。

```
public class ExpDemo12 {
    public static void main(String args[]){
        int x = 10 ;
        assert x!=10 ;
    }
};
```

本程序中增加了断言，但是一个断言并不是直接起作用的，需要在执行一个 java 程序的时候增加“-ea”的参数。

```
D:\testjava\expdemo>java -ea ExpDemo12
Exception in thread "main" java.lang.AssertionError
        at ExpDemo12.main(ExpDemo12.java:4)
```

由于在断言出出现了错误，所以一旦加入了-ea 参数之后将在发生错误的地方进行异常的显示，但是现在显示的都是系统信息，所以，也可以为每一个断言增加自己的信息。

```
public class ExpDemo12 {
    public static void main(String args[]){
        int x = 10 ;
        assert x!=10:"x!=10" ;
    }
};
```

4、总结

1、 异常的处理就是保证程序可以正常的执行完毕

2、 异常处理的标准格式及流程

3、 Exception 和 RuntimeException 的区别