## 实验四 Java异常处理机制

|  |
| --- |
| 实验目的：   1. 理解异常现象及异常的抛出机制 2. 掌握异常抛出时的处理机制 3. 掌握自定义异常的方法 4. 能够掌握在方法代码中添加抛出异常的处理 5. 掌握内部类的一般使用方法 |
| 实验要求和过程   1. public class TestApp{   public static void main(String[] args) {  try {  int i = 0;  int j = 1 / i;  String myname = null;  if (myname.length() > 2)  System.out.print("1");  } catch (NullPointerException e) {  System.out.print("2");  } catch (Exception e) {  System.out.print("3");  }  }  }分析上述程序运行后的输出的结果和原因。  3  除数为0不属于NullPointerException属于Exception  2、（1）首先预测下列程序的运行结果，然后编译、执行以验证你的预测：  public class JLabTestA {  public static void main(String args[]) {  int k;  for (int i = 0; i < 4; i++) {  try {  switch (i) {  case 0:  int zero = 0;  k = 911 / zero;  break;  case 1:  int[] b = null;  k = b[0];  break;  case 2:  int[] c = new int[2];  k = c[9];  break;  case 3:  char ch = "abc".charAt(99);  break;  }  } catch (Exception e) {  System.out.println("\nTestcase #" + i + "\n");  System.out.println(e);  }  }  }  }  Testcase #0  java.lang.ArithmeticException: / by zero  Testcase #1  java.lang.NullPointerException  Testcase #2  java.lang.ArrayIndexOutOfBoundsException: Index 9 out of bounds for length 2  Testcase #3  java.lang.StringIndexOutOfBoundsException: String index out of range: 99  （2）将上述程序中catch(Exception e)更改为catch(ArithmeticException e)，预测程序运行的结果并验证之。  Testcase #0  Exception in thread "main" java.lang.ArithmeticException: / by zero  java.lang.NullPointerException  at work.JLabTestA.main(JLabTestA.java:15)  （3）将上述程序中catch(Exception e)更改为catch(NullPointerException e)，预测程序运行的结果并验证之。  Exception in thread "main" java.lang.ArithmeticException: / by zero  at work.JLabTestA.main(JLabTestA.java:11)  （4）在(3)的程序中添加一些语句，使得程序能够输出Exception Test Finished!的信息。（提示：使用finally语句）  **public** **class** JLabTestA {  **public** **static** **void** main(String args[]) {  **int** k;  **for** (**int** i = 0; i < 4; i++) {  **try** {  **switch** (i) {  **case** 0:  **int** zero = 0;  k = 911 / zero;  **break**;  **case** 1:  **int**[] b = **null**;  k = b[0];  **break**;  **case** 2:  **int**[] c = **new** **int**[2];  k = c[9];  **break**;  **case** 3:  **char** ch = "abc".charAt(99);  **break**;  }  } **catch** (NullPointerException e) {  System.***out***.println("\nTestcase #" + i + "\n");  System.***out***.println(e);  } **finally** {  System.***out***.println("Exception Test Finished!");  }  }  }  }  3、编译下列程序，观察报错信息并思考原因。  public class JLabTest{  public static void main( String args[] ){  throwException();  }  public static void throwException() throws Exception{  System.out.println( "Method throwException" );  throw new Exception();  }  }  未处理的异常类型 Exception  修改上述程序使之能正常编译，并能输出信息:Exception Catched!  **public** **class** JLabTest {  **public** **static** **void** main(String args[]) {  **try** {  *throwException*();  } **catch** (Exception e) {  System.***out***.println("Exception Catched!");  }  }  **public** **static** **void** throwException() **throws** Exception {  System.***out***.println("Method throwException");  **throw** **new** Exception();  }  }   1. （请参考上课范例）下面是一个名称为NegativeAmountException的自定义异常类，表示一个不正常的银行账目事件类。填充下面的语句，完成该类的编写**。**   **class** NegativeAmountException **\_\_extends\_\_\_ \_Exception\_\_\_**{  //NegativeAmountException异常：用消息s创建异常  NegativeAmountException(String s){  **super**(s);  }  }  **class** Account{  **double** balance;  //构造函数，余额为0；  **public** Account(){  balance = 0;  }  //构造函数，余额为n，如果初始余额小于0抛出异常  **public** Account( **double** n) **throws** NegativeAmountException{  **if**(n>0){  **this**.balance = n;  }**else** {  **\_\_\_** **throw new NegativeAmountException(**"初始余额出错"**) \_;**  }  }  //查询余额方法，返回当前余额  **public** **double** getBalance(){  **return** **this**.balance;  }  //存款方法，存款数额amount; 如果存款数目小于0抛出异常  **public** **void** deposit(**double** amount)**\_ throws** NegativeAmountException **\_\_**{  **if**(amount>=0){  balance+=amount;  }**else** {  **throw** **new** NegativeAmountException("存款出错");  }  }  //取款方法，取款数额amount; 如果取款数目小于0抛出异常  **public** **void** withdraw(**double** amount)**\_\_\_ throws** NegativeAmountException **\_** {  **if**(amount<0){  **throw** **new** NegativeAmountException("操作错误");  }**else** **if**(balance<amount){  **throw** **new** NegativeAmountException("取款出错");  }**else**{  balance-=amount;  }  }  }   1. 模仿上题中**NegativeAmountException**自定义异常的写法，根据下面要求写程序。    1. 自定义异常OnlyOneException与NoOprandException，并补充各自类的构造函数，参数用于保存异常发生时候的信息；    2. 添加main方法，从命令行参数读入两个数据，计算这两个数据的和并输出。    3. 如果参数的数目只要一个，抛出OnlyOneException异常并退出程序的执行；如果没有参数 ，抛出NoOprandException异常并退出程序的执行；   **import** java.util.Scanner;  **public** **class** Test0 {  **public** **static** **void** main(String args[]) **throws** Exception {  Scanner in = **new** Scanner(System.***in***);  String[] num = in.nextLine().split(" ");  **if** (num.length == 1)  **throw** **new** OnlyOneException("参数的数目只有一个");  **else** **if** (num.length == 0)  **throw** **new** NoOprandException("没有参数");  **else** **if** (num.length == 2) {  **int** a = Integer.*parseInt*(num[0]);  **int** b = Integer.*parseInt*(num[1]);  System.***out***.println(a + b);  }  }  }  **class** OnlyOneException **extends** Exception {  OnlyOneException(String s) {  **super**(s);  }  }  **class** NoOprandException **extends** Exception {  NoOprandException(String s) {  **super**(s);  }  }  实验总结：  了解异常处理机制，学会使用创建用户自定义异常类 |
|  |