# 填空题

1. \_\_\_\_\_\_异常\_\_\_机制是一种非常有用的辅助性程序设计方法。采用这种方法可以使得在程序设计时将程序的正常流程与错误处理分开，有利于代码的编写和维护。
2. 在Java异常处理中可以使用多个catch子句，此时包含异常类的父类Exception的catch子句的位置应该是在\_最后一个catch中\_\_\_\_\_\_\_\_\_\_。
3. 异常处理中finally块可确保无论是否发生异常，该块中代码总能被执行。finally块不执行的唯一情况是在异常处理代码中执行\_\_\_System.exit()\_\_\_\_语句退出Java虚拟机。
4. 异常是由Java应用程序抛出和处理的非严重错误，比如所需文件没有找到、零作除数，数组下标越界等，可分为两类：Checked异常和\_\_\_运行时异常\_\_\_\_。
5. 在Java中对于程序可能出现的检查时异常，要么用try…catch语句捕获并处理它，要么使用\_\_\_\_throws\_\_\_\_语句抛出，由上一级调用者来处理。
6. Java异常处理中，如果一个方法中出现了多个Checked异常，可以在方法声明中使用关键字\_\_\_\_\_throws\_\_\_\_\_\_\_\_声明抛出，各异常类型之间使用逗号分隔。
7. Java中，处理异常两大方式\_\_\_捕获 抛出\_\_\_。

# 选择题

|  |  |  |
| --- | --- | --- |
| 1. | 以下关于异常的代码的执行结果是（ C ）。（选择一项） | |
|  | public class Test {  public static void main(String args[]) {  try {  System.*out*.println("try");  return;  } catch(Exception e){  System.*out*.println("catch");  }finally {  System.*out*.println("finally");  }  }  } | |
|  |  |  |
|  | A. | try  catch  finally |
|  | B. | catch  finally |
|  | C. | try  finally |
|  | D. | try |

|  |  |  |
| --- | --- | --- |
| 2. | 在异常处理中，如释放资源、关闭文件等由（ C ）来完成。（选择一项） | |
|  |  |  |
|  | A | try子句 |
|  | B. | catch子句 |
|  | C. | finally子句 |
|  | D. | throw子句 |

|  |  |  |
| --- | --- | --- |
| 3. | 编译并运行如下Java程序，将输出（ D ）。（选择一项） | |
|  | public static void main(String[] args) {  try {  int num1 = 2;  int num2 = 0;  int result = num1 / num2;  System.*out*.println(result);  throw new NumberFormatException( );  } catch (ArrayIndexOutOfBoundsException e) {  System.*out*.print("1");  } catch (NumberFormatException e) {  System.*out*.print("2");  } catch (Exception e) {  System.*out*.print("3");  } finally {  System.*out*.print("4");  }  System.*out*.print("5");  } | |
|  |  |  |
|  | A | 134 |
|  | B. | 2345 |
|  | C. | 1345 |
|  | D. | 345 |

|  |  |  |
| --- | --- | --- |
| 4. | 阅读如下Java代码,其中错误的行是（ Ac ）。（选择二项） | |
|  | public class Student {  private String stuId;  public void setStuId(String stuId) throw Exception { // 1  if (stuId.length() != 4) { // 2  throws new Exception("学号必须为4位!"); // 3  } else {  this.stuId = stuId; //4  }  }  } | |
|  |  |  |
|  | A | 1 |
|  | B. | 2 |
|  | C. | 3 |
|  | D. | 全部正确 |

|  |  |  |
| --- | --- | --- |
| 5. | 下面选项中有关Java异常处理模型的说法错误的是（ AD ）。（选择二项） | |
|  |  |  |
|  | A | 一个try块只能有一条catch语句 |
|  | B. | 一个try块中可以不使用catch语句 |
|  | C. | catch块不能单独使用，必须始终与try块在一起 |
|  | D. | finally块可以单独使用，不是必须与try块在一起 |

|  |  |  |
| --- | --- | --- |
| 6. | 下面选项中属于运行时异常的是（c b ）。（选择二项） | |
|  |  |  |
|  | A | Exception和SexException |
|  | B. | NullPointerException和InputMismatchException |
|  | C. | ArithmeticException和ArrayIndexOutOfBoundsException |
|  | D. | ClassNotFoundException和ClassCastException |

|  |  |  |
| --- | --- | --- |
| 7. | 阅读如下Java代码，在控制台输入"-1",执行结果是（b）。（选择一项） | |
|  | public class Demo {  public static void main(String[] args) {  Scanner input = new Scanner(System.*in*);  System.*out*.print("请输入数字：");  try {  int num = input.nextInt();  if (num < 1 || num > 4) {  throw new Exception("必须在1-4之间！");  }  } catch (InputMismatchException e) {  System.*out*.println("InputMismatchException");  } catch (Exception e) {  System.*out*.println(e.getMessage());  }  }  } | |
|  |  |  |
|  | A | 输出：InputMismatchException |
|  | B. | 输出：必须在1-4之间！ |
|  | C. | 什么也没输出 |
|  | D. | 编译错误 |

# 简答题

1. Error和Exception的区别

Error：java虚拟机无法解决的重要问题，表示依靠程序本身无法恢复的严重错误比如内存溢出，虚拟机错误

Exception编程错误或偶然的外在因素导致的一般性问题，可以使用针对性的代码进行处理。

1. Checked异常和Runtime异常的区别。

编译型异常程序必须捕获或声明抛出的异常，否则会出现编译错误无法编译，

**运行时异常即使没有使用try-catch或throws进行处理，仍旧可以进行编译和运行。如果运行时发生异常，会输出异常的堆栈信息并中止程序执行。**

1. Java异常处理中，关键字try、catch、finally、throw、throws分别代表什么含义？

Try:监听异常

Catch:处理异常

Finally：无论是否有异常，都会执行

Throw:抛出异常类对象（jdk支持的异常，自定义的异常）

Throws:抛出异常类对象的类型，供调用者来处理异常

1. throws和throw的区别

作用不同：throw用于程序员自行产生并抛出异常；throws用于声明在该方法内抛出了异常

使用的位置不同：throw位于方法体内部，可以作为单独语句使用；throws必须跟在方法参数列表的后面，不能单独使用。

内容不同：throw抛出一个异常对象，且只能是一个；throws后面跟异常类，而且可以有多个。

# 编码题

* 1. 编写程序接收用户输入分数信息，如果分数在0—100之间，输出成绩。如果成绩不在该范围内，抛出异常信息，提示分数必须在0—100之间。

要求：使用自定义异常实现

package com.uek.test.exception;

import java.util.Scanner;

/\*\*

\* 1. 编写程序接收用户输入分数信息，如果分数在0—100之间，

\* 输出成绩。如果成绩不在该范围内，抛出异常信息，提示分数必须在0—100之间。

\* 要求：使用自定义异常实现

\*/

public class Test01 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.*in*);

System.*out*.println("输入成绩：");

int score = sc.nextInt();

if(score>=0&&score<=100) {

System.*out*.println(score);

}else {

try {

throw new ScoreException("分数必须在0—100之间");

} catch (ScoreException e) {

System.*out*.println(e.getMessage());

}

}

}

}

class ScoreException extends RuntimeException{

private static final long *serialVersionUID* = 1L;

public ScoreException() {}

public ScoreException(String msg) {

super(msg);

}

}

* 1. 写一个方法void isTriangle(int a,int b,int c)，判断三个参数是否能构成一个三角形， 如果不能则抛出异常IllegalArgumentException，显示异常信息 “a,b,c不能构成三角形”，如果可以构成则显示三角形三个边长，在主方法中得到命令行输入的三个整数， 调用此方法，并捕获异常。

package com.uek.test.exception;

import java.util.Scanner;

/\*\*

\* 2. 写一个方法void isTriangle(int a,int b,int c)， 判断三个参数是否能构成一个三角形，

\* 如果不能则抛出异常IllegalArgumentException， 显示异常信息 “a,b,c不能构成三角形”，如果可以构成则显示三角形三个边长，

\* 在主方法中得到命令行输入的三个整数， 调用此方法，并捕获异常。

\*/

public class Test02 {

public static void isTriangle(int a, int b, int c) {

try {

if (a < 0 || b < 0 || c < 0) {

throw new IIllegalArgumentException("三角形边长不能为负数");

}

if (a + b > c && a + c > b && b + c > a) {

System.*out*.println("三角形三边分别是：" + "a=" + a + " b=" + b + " c=" + c);

} else {

throw new IIllegalArgumentException(a + "，" + b + "，" + c + "，" + "不能构成三角形");

}

} catch (IIllegalArgumentException e) {

System.*out*.println(e.getMessage());

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.*in*);

System.*out*.println("请分别输入三角形三条边：");

int a = sc.nextInt();

int b = sc.nextInt();

int c = sc.nextInt();

*isTriangle*(a, b, c);

}

}

class IIllegalArgumentException extends RuntimeException {

private static final long *serialVersionUID* = 1L;

public IIllegalArgumentException() {

}

public IIllegalArgumentException(String msg) {

super(msg);

}

}