**每日作业卷答案**

**就业班JavaSE第5天**

# 关卡1

## 训练案例1

### 训练描述

请谈谈你对Object类的理解

### 操作步骤答案

1. Object类是所有类的根类
2. Object类中的所有方法子类都可以用
3. 所有类在创建对象的时候，最终找的父类就是Object。

## 训练案例2

### 训练描述

请阐述你对equals方法的理解

### 操作步骤答案

1. equals方法用于比较两个对象是否相同;
2. 在Object类的默认实现使用==,比较两个对象的地址是否相同
3. 当比较不是对象的地址,而是对象的属性值的时候,就需要重写equals方法;
4. 在复写Object中的equals方法时，一定要注意public boolean equals(Object obj)的参数是Object类型，在调用对象的属性时，一定要进行类型转换，在转换之前必须进行类型判断

## 训练案例3

### 训练描述

请阐述你对toString()的理解

### 操作步骤答案

1. toString() 方法返回该对象的字符串表示,在Object类中的默认实现是对象的类型+@+内存地址值。
2. 当我们要返回对象的属性值的的时候,就需要重写toString()方法
3. 直接使用输出语句输出对象名的时候,会自动调用这个对象的toString()方法

## 训练案例4

### 训练描述

请阐述异常的分类

### 操作步骤答案

1. 异常继承体系为:
   1. 最顶层父类是Throwable
   2. Error与Exception继承Throwable
   3. RuntimeException继承Exception
2. Error是系统级别的严重的错误,错误无法处理,只能修改代码.
3. Exception是编译时异常,编译时必须对编译异常进行处理,否则编译失败,如:日期格式化异常
4. RuntimeException是运行时异常,编译时无需处理,但程序运行的时候可能会报错,如:算术异常

## 训练案例5

### 训练描述

请阐述 错误(Error)与(Exception) 的区别

### 操作步骤答案

1. Error是系统级别的严重的错误,错误无法使用try...catch或throws处理,如果出现须修改代码.
2. Exception是普通的问题,可以使用try...catch或throws进行处理,处理后程序可继续运行.

## 训练案例6

### 训练描述

请说出虚拟机处理异常的方式

### 操作步骤答案

1. 把异常对象的类名,异常内容,异常出现的位置信息打印到控制台上
2. 终止程序执行

## 训练案例7

### 训练描述

请说出四种运行时异常

### 操作步骤答案

1. 索引越界异常:IndexOutOfBoundsException
2. 数组索引越界异常:ArrayIndexOutOfBoundsException
3. 字符串索引越界异常:StringIndexOutOfBoundsException
4. 空指针异常: NullPointerException
5. 算术异常:ArithmeticException

## 训练案例8

### 训练描述

请说出throw与throws的区别

### 操作步骤答案

1. throw用在方法内部,后跟一个异常对象，用来抛出一个异常对象，将这个异常对象传递到调用者处，并结束当前方法的执行
2. throws用于在方法声明上,后跟异常类名,用于告诉调用者,本方法内部可能会抛出异常,请你处理一下.

## 训练案例9

### 训练描述

请简述异常的处理方式

### 操作步骤答案

1. 异常的处理方式有两种,分别是使用throws和try...catch...finally
2. throws用在方法的声明上后接异常类名,是把异常抛给调用者进行处理
3. try...catch...finally是捕获异常,自己处理,处理完毕后面的程序可以继续运行
   1. try代码块中是可能出现异常的代码
   2. catch代码块,是遇到异常,对异常进行处理的代码
   3. finally代码块是无论是否发生异常,都必须执行的代码,用于释放资源.

## 训练案例10

### 训练描述

请用代码实现如下需求:

1. 写一个方法实现获取字符串某个索引上的字符的功能
2. 写一个测试测试方法调用上面写的方法,使用thows方式进行处理异常,并在main方法中调用这个测试方法
3. 再写一个测试测试方法调用上面写的方法,使用try...catch方式进行处理异常,并在main方法中调用这个测试方法

### 操作步骤答案

/\*

1.定义 Test10类

2.提供一个静态方法:public static char charAt(String str, int index) 用于获取字符串str,index索引处的字符

3.在方法中charAt(String,int index)

a) 当字符串为null抛出 Exception("字符串不能为null")异常,

b) 当字符串为"" 的时候抛出 Excetion("字符串不能为空") 异常;

c) 当index不在 [0,str.length()-1] 之间的时,抛出Exception("索引越界") 异常.

d) 否则返回str在index索引的字符

4.提供一个静态的方法testThrows() 用于测试charAt(String,int index)方法

a)在方法中调用charAt(String,int index)方法;

i.注意需要测试每一种情况

ii.体会: 程序一旦遇到异常后面的代码就不执行了

iii.注释运行过测试的代码,继续测试下一种情况

b)使用throws方式对异常进行处理

c)在main方法中,调用本方法

5.提供 一个静态方法testTryCatch(),用于测试charAt(String,int index)方法

a)在方法中调用charAt(String,int index)方法;

i.注意需要测试每一种情况

b)使用try...catch方式对异常进行处理

c)在main方法中,调用本方法.

\*/

**public** **class** Test10 {

// 1. 获取字符串某个索引上的字符的功能

**public** **static** **char** charAt(String str, **int** index) **throws** Exception {

// a) 当字符串为null抛出 Exception("字符串不能为null")异常,

**if**(**null** == str){

**throw** **new** Exception("字符串不能为null");

}

// b) 当字符串为"" 的时候抛出 Excetion("字符串不能为空") 异常;

**if**(str.equals("")){

**throw** **new** Exception("字符串不能为空");

}

// c) 当index不在 [0,str.length()-1] 之间的时,抛出Exception("索引越界") 异常.

**if**(index < 0 || index >= str.length()){

**throw** **new** Exception("索引越界"+index);

}

// d) 否则返回str在index索引的字符

**return** str.charAt(index);

}

// 4.提供一个静态的方法testThrows() 用于测试charAt(String,int index)方法

**public** **static** **void** testThrows() **throws** Exception{

**char** c;

// c = charAt(null, 0);

// c = charAt("", 0);

// c = charAt("abc",4);

c = *charAt*("abc", 1);

System.***out***.println(c);

}

// 5.提供 一个静态方法testTryCatch(),用于测试charAt(String,int index)方法

**public** **static** **void** testTryCatch(){

// a)在方法中调用charAt(String,int index)方法;

// i.注意需要测试每一种情况

// b)使用try...catch方式对异常进行处理

**try**{

**char** c;

// c = charAt(null, 0);

// c = charAt("", 0);

// c = charAt("abc",4);

c = *charAt*("abc", 1);

System.***out***.println(c);

}**catch**(Exception e){

System.***out***.println("捕获异常");

e.printStackTrace();

}

System.***out***.println("程序继续");

}

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

// 在main方法中,调用testThrows 方法.

// testThrows();

// 在main方法中,调用testTryCatch方法

*testTryCatch*();

}

}

## 训练案例11

### 训练描述

使用代码实现

写一个实现把字符串中数据,写入项目根目录下的content.txt文件中

### 操作步骤答案

/\*

1.定义类(Test11)

2.写一个静态方法 void write(String content),在方法中

a)定义字符缓冲输出流变量BufferedWriter bw;

b)写一个try{ }catch(IOException e){ }finally{ }代码块

c)在try{ }在代码块中

i.创建BufferedWriter对象,绑定content.txt文件,赋值给bw

ii.调用bw的write()方法,传入content

d)在catch代码块中,打印异常信息

e)在finally代码块关闭流

i.写try{}catch(IOException ex){} 代码块

ii.在try 代码块中,如果bw!=null,调用bw.close()方法

iii.在catch代码块,打印异常信息

3.在main方法中调用write(String conent)方法

\*/

**public** **class** Test11 {

// 写一个静态方法 void write(String content)

**public** **static** **void** write(String content){

// a)定义字符缓冲输出流变量BufferedWriter bw,初始值为null;

BufferedWriter bw = **null**;

// b)写一个try{ }catch(IOException e){ }finally{ }代码块

**try**{

// c)在try{ }在代码块中

// i.创建BufferedWriter对象,绑定content.txt文件,赋值给bw

bw = **new** BufferedWriter(**new** FileWriter("content.txt"));

// ii.调用bw的write()方法,传入content

bw.write(content);

}**catch**(IOException e){

// d)在catch代码块中,打印异常信息

e.printStackTrace();

}**finally**{

// e)在finally代码块关闭流

// i.写try{}catch(IOException ex){} 代码块

**try**{

// ii.在try 代码块中,如果bw!=null,调用bw.close()方法

**if**(bw != **null**){

bw.close();

}

}**catch**(IOException ex){

// iii.在catch代码块,打印异常信息

ex.printStackTrace();

}

}

}

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

*write*("我爱Java,我爱极客营程序员");

}

}

## 训练案例12

### 训练描述

请使用代码实现

每一个学生(Student)都有学号,姓名和分数,分数永远不能为为负数

如果师徒给学生赋值一个负数,抛出一个自定异常

### 操作步骤答案

/\*

1.定义异常类NoScoreException,继承RuntimeException

a)提供空参和有参构造方法

\*/

**public** **class** NoScoreException **extends** RuntimeException {

// 空参构造

**public** NoScoreException() {

**super**();

}

// 有参构造

**public** NoScoreException(String message) {

**super**(message);

}

}

/\*

2.定义学生类(Student)

a)属性:name,score

b)提供空参构造

c)提供有参构造;

i.使用setXxx方法给名称和score赋值

d)提供setter和getter方法

i.在setScore(int score)方法中

1.首先判断,如果score为负数,就抛出NoScoreException,异常信息为:分数不能为负数:xxx.

2.然后在给成员score赋值.

\*/

**public** **class** Student {

**private** String name;

**private** **int** score;

// 空参构造

**public** Student() {

**super**();

}

// c)提供有参构造;

// i.使用setXxx方法给名称和score赋值

**public** Student(String name,**int** score){

setName(name);

setScore(score);

}

// d)提供setter和getter方法

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getScore() {

**return** score;

}

// i.在setScore(int score)方法中

**public** **void** setScore(**int** score) {

// 1.首先判断,如果score为负数,就抛出NoScoreException,异常信息为:分数不能为负数:xxx.

**if**(score < 0){

**throw** **new** NoScoreException(":分数不能为负数:"+score);

}

// 2.然后在给成员score赋值.

**this**.score = score;

}

}

/\*

3.定义测试类Test12

a)提供main方法,在main方法中

i.使用满参构造方法创建Student对象,分数传入一个负数,运行程序

ii.由于一旦遇到异常,后面的代码的将不在执行,所以需要注释掉上面的代码

iii.使用空参构造创建Student对象

iv.调用setScore(int score)方法,传入一个正数,运行程序

v.调用setScore(int score)方法,传入一个负数,运行程序

\*/

**public** **class** Test12 {

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

// i.使用满参构造方法创建Student对象,分数传入一个负数,运行程序

// Student s = new Student("景甜", -10);

// ii.由于一旦遇到异常,后面的代码的将不在执行,所以需要注释掉上面的代码

// iii.使用空参构造创建Student对象

Student s = **new** Student();

// iv.调用setScore(int score)方法,传入一个正数,运行程序

s.setScore(100);

// v.调用setScore(int score)方法,传入一个负数,运行程序

s.setScore(-5);

}

}

# 关卡2

## 训练案例1

### 训练描述

请使用代码实现

写一个方法用于复制文本文件;

要求:使用try...catch...finally对异常进行处理

### 操作步骤答案

/\*

1.定义测试类Test13

a)写一个复制文件的静态方法:copyFile(String src,String dest)

i.这里的src为源文件路径,dest为目标文件路径

ii.定义BufferedReader br变量

iii.定义BufferedWriter bw 变量

iv.写try...catch...finally代码块

v.在try代码块中

1.创建BufferedReader的对象,绑定源文件,并把对象赋值给br

2.创建BufferedWriter的对象,绑定目标文件,并把对象赋值给bw

3.定义一个字符数组 chs,长度为1024,用于临时存储读取到的数据

4.定义整型变量len,用于记录读取到的有效字节个数

5.使用while循环,读一个字符数组,写一个字符数组,并刷新流

vi.在catch代码块中,打印异常信息

vii.在finally代码块

1.写一个try...catch代码块

2.在try代码块中: 如果bw不为null就关闭bw

3.在catch代码块中打印异常信息

4.在写一个try...catch代码块

5.在try代码块中,如果br不为null就关闭br

6.在catch代码中打印异常信息

b)提供main方法,在main方法中

i.在main方法,调用刚刚写copyFile方法,传入源文件路径和目标文件路径

\*/

**public** **class** Test13 {

// a)写一个复制文件的静态方法:copyFile(String src,String dest)

**public** **static** **void** copyFile(String src,String dest){

// i.这里的src为源文件路径,dest为目标文件路径

// ii.定义BufferedReader br变量

BufferedReader br = **null**;

// iii.定义BufferedWriter bw 变量

BufferedWriter bw = **null**;

// iv.写try...catch...finally代码块

**try**{

// v.在try代码块中

// 1.创建BufferedReader的对象,绑定源文件,并把对象赋值给br

br = **new** BufferedReader(**new** FileReader(src));

// 2.创建BufferedWriter的对象,绑定目标文件,并把对象赋值给bw

bw = **new** BufferedWriter(**new** FileWriter(dest));

// 3.定义一个字符数组 chs,长度为1024,用于临时存储读取到的数据

**char**[] chs = **new** **char**[1024];

// 4.定义整型变量len,用于记录读取到的有效字节个数

**int** len = 0;

// 5.使用while循环,读一个字符数组,写一个字符数组,并刷新流

**while**((len = br.read(chs))!=-1){

bw.write(chs, 0, len);

}

}**catch**(IOException e){

// vi.在catch代码块中,打印异常信息

e.printStackTrace();

}**finally**{

// vii.在finally代码块

// 1.写一个try...catch代码块

**try**{

// 2.在try代码块中: 如果bw不为null就关闭bw

**if**(bw != **null**){

bw.close();

}

}**catch**(IOException ex){

// 3.在catch代码块中打印异常信息

ex.printStackTrace();

}

// 4.在写一个try...catch代码块

**try**{

// 5.在try代码块中,如果br不为null就关闭br

**if**(br!=**null**){

br.close();

}

}**catch**(IOException ex){

// 6.在catch代码中打印异常信息

ex.printStackTrace();

}

}

}

// i.在main方法,调用刚刚写copyFile方法,传入源文件路径和目标文件路径

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

*copyFile*("content.txt", "13.txt");

}

}

## 训练案例2

### 训练描述

通过代码验证

1. 方法重写,子类只能抛比父类小的异常(父类所抛异常或父类所抛异常的子类)

a. 重写方法不能抛出与父类平级的其他异常

b. 重写方法不能抛出比父类高级的异常

c. 重写方法: 可以抛出和父类同样的异常

d. 重写方法: 可以抛出父类抛出异常的子类异常.

e. 重写方法: 可以抛出多个 "父类抛出异常的子类异常".

2. 方法重写, 父类没有抛异常,子类重写父类方法也不能抛异常?

a. 父类不抛出异常,子类也不能抛出异常

b. 如果子类内部抛出了异常(或调用抛出异常的方法)只能自己处理,不允许向外抛.

注: 这里所说的异常都是编译异常

### 操作步骤答案

/\*

1.构建异常体系

a)定义PainExecption(疼异常)类继承Exception

i.提供空参和有参构造,注意有参构造必须调用父类的有参构造

\*/

**public** **class** PainExecption **extends** Exception {

**public** PainExecption() {

**super**();

}

**public** PainExecption(String message) {

**super**(message);

}

}

/\*

b1)定义ToothPainException(牙疼异常)类继承PainException

i.提供空参和有参构造,注意有参构造必须调用父类的有参构造

\*/

**public** **class** ToothPainException **extends** PainExecption {

**public** ToothPainException() {

**super**();

}

**public** ToothPainException(String message) {

**super**(message);

}

}

/\*

b2)定义TonguePainException(舌头疼异常)类继承PainException

i.提供空参和有参构造,注意有参构造必须调用父类的有参构造

\*/

**public** **class** TonguePainException **extends** PainExecption{

**public** TonguePainException() {

**super**();

}

**public** TonguePainException(String message) {

**super**(message);

}

}

/\*

c1)定义BigToothPainException(大牙疼)继承ToothPainException(牙痛)

i.提供空参和有参构造,注意有参构造必须调用父类的有参构造

\*/

**public** **class** BigToothPainException **extends** ToothPainException {

**public** BigToothPainException() {

**super**();

}

**public** BigToothPainException(String message) {

**super**(message);

}

}

/\*

c2)定义FrontToothPainExcption(门牙疼)继承ToothPainException(牙痛)

i.提供空参和有参构造,注意有参构造必须调用父类的有参构造

\*/

**public** **class** FrontToothPainExcption **extends** ToothPainException {

**public** FrontToothPainExcption() {

**super**();

}

**public** FrontToothPainExcption(String message) {

**super**(message);

}

}

/\*

2.定义用于测试的父类Father

a)提供一个吃(eat)的成员方法,抛出牙疼异常

i.输出吃到一个石子

ii.抛出一个牙疼的异常

b)提供一个喝(drink)的成员方法不抛出异常

i.输出: 喝什么都没有问题

\*/

**public** **class** Father {

// a)提供一个吃(eat)的成员方法,抛出牙疼异常

**public** **void** eat() **throws** ToothPainException {

// i.输出吃到一个石子

System.***out***.println("吃到一个石头子");

// ii.抛出一个牙疼的异常

**throw** **new** ToothPainException();

}

// b)提供一个喝(drink)的成员方法不抛出异常

**public** **void** drink() {

// i.输出: 喝什么都没有问题

System.***out***.println("喝什么都没有问题");

}

}

/\*

3.定义用于测试的子类Son,继承Father

a)验证: 重写方法不能抛出与父类平级的其他异常

i.重写父类的eat方法,抛出与牙疼平级舌头疼异常,编译报错,说明这句话是对的

ii.注释掉这个重写的方法,验证下一个

b)验证: 重写方法不能抛出比父类高级的异常

i.重写父类的eat方法,抛出比牙疼高级疼异常,编译报错,说明这句话是对的

c)验证:重写方法: 可以抛出和父类同样的异常

i.重写父类的eat方法,也抛出牙疼异常,编译通过,说明这句话是对的

d)验证:重写方法: 可以抛出父类抛出异常的子类异常.

i.重写父类的eat方法,抛出牙疼异常的子类异常大牙疼异常,编译通过,说明这句话是对的

e)验证: 重写方法: 可以抛出多个 "父类抛出异常的子类异常".

i.重写父类的eat方法,抛出牙疼异常的两个子类异常,大牙疼和门牙疼,编译通过,说明这句是对的

f)验证:父类不抛出异常,子类也不能抛出异常(编译时异常)

i.重写父类的drink方法,抛出(牙疼,舌头疼,门牙疼)任意一个编译异常,编译都报错,说明这句话是对的

g)验证: 如果子类内部抛出了异常(或调用抛出异常的方法)只能自己处理,不允许向外抛.

i.输出喝到了100度的水

ii.抛出舌头疼异常

iii.这个时候编译报错

iv.把上面两句代码放到try代码块中

v.在catch代码块中打印异常信息

vi.编译通过.

\*/

**public** **class** Son **extends** Father {

// 1. 重写方法不能抛出与父类平级的其他异常

// public void eat() throws TonguePainException {

// }

// 2. 重写方法不能抛出比父类高级的异常

// public void eat() throws PainException {

//

// }

// // 3. 重写方法: 可以抛出和父类同样的异常

// public void eat() throws ToothPainException {

//

// }

// 4. 重写方法: 可以抛出父类抛出异常的子类异常.

// public void eat() throws BigToothPainException {

//

// }

// 5. 重写方法: 可以抛出多个 "父类抛出异常的子类异常".

// public void eat() throws BigToothPainException,FrontToothPainExcption {

// }

//// 父类不抛出异常,子类也不能抛出异常(编译时异常)

// public void drink() throws FrontToothPainExcption {

// }

// 如果子类有异常只能自己处理,不允许向外抛.

**public** **void** drink() {

**try**{

System.***out***.println("到100度的热水了");

**throw** **new** TonguePainException();

} **catch** (TonguePainException ex) {

System.***out***.println("在疼都要忍着不能说,因为你爸都不说,你也不能说");

}

}

}

## 训练案例3

### 训练描述

请用代码描述:

在一款角色扮演游戏中,每一个人都会有名字和生命值;角色的生命值不能为负数

要求:当一个人物的生命值为负数的时候需要抛出自定的异常

### 操作步骤答案

/\*

1.自定义异常类NoLifeValueExption继承RuntimeException

a)提供空参和有参构造

b)在有参构造中,需要调用父类的有参构造,把异常信息传入

\*/

**public** **class** NoLifeValueExption **extends** RuntimeException{

**public** NoLifeValueExption() {

**super**();

}

**public** NoLifeValueExption(String message) {

**super**(message);

}

}

/\*

2.定义Person类

a)属性:名称(name)和生命值(lifeValue)

b)提供空参构造

c)提供有参构造;

i.使用setXxx方法给name和lifeValue赋值

d)提供setter和getter方法

i.在setLifeValue(int lifeValue)方法中

1.首先判断,如果 lifeValue为负数,就抛出NoLifeValueException,异常信息为:生命值不能为负数:xxx.

2.然后在给成员lifeValue赋值.

\*/

**public** **class** Person {

// 名称(name)

**private** String name;

// 生命值(lifeValue)

**private** **int** lifeValue;

// 空参构造

**public** Person() {

**super**();

}

// c)提供有参构造;

**public** Person(String name, **int** lifeValue) {

**super**();

// i.使用setXxx方法给name和lifeValue赋值

setName(name);

setLifeValue(lifeValue);

}

//d)提供setter和getter方法

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getLifeValue() {

**return** lifeValue;

}

// i.在setLifeValue(int lifeValue)方法中

**public** **void** setLifeValue(**int** lifeValue) {

// 1.首先判断,如果 lifeValue为负数,就抛出NoLifeValueException,异常信息为:生命值不能为负数:xxx.

**if**(lifeValue < 0){

**throw** **new** NoLifeValueExption("生命值不能为负数:"+lifeValue);

}

// 2.然后在给成员lifeValue赋值.

**this**.lifeValue = lifeValue;

}

}

/\*

3.定义测试类Test15

a)提供main方法,在main方法中

i.使用满参构造方法创建Person对象,分数传入一个负数,运行程序

ii.由于一旦遇到异常,后面的代码的将不在执行,所以需要注释掉上面的代码

iii.使用空参构造创建Person对象

iv.调用setLifeValue(int lifeValue)方法,传入一个正数,运行程序

v.调用setLifeValue(int lifeValue)方法,传入一个负数,运行程序

\*/

**public** **class** Test15 {

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

// i.使用满参构造方法创建Person对象,分数传入一个负数,运行程序

// Person p = new Person("林思意", -100);

// ii.由于一旦遇到异常,后面的代码的将不在执行,所以需要注释掉上面的代码

// iii.使用空参构造创建Person对象

Person p = **new** Person();

// iv.调用setLifeValue(int lifeValue)方法,传入一个正数,运行程序

p.setLifeValue(1000);

// v.调用setLifeValue(int lifeValue)方法,传入一个负数,运行程序

p.setLifeValue(-100);

}

}

## 训练案例4

### 训练描述

请使用代码描述

写一个方法实现用户登陆,插,传入用户名和密码

如果用户名错误,就抛出自定义登陆异常(LoginException),异常信息为用户名不存在

如果密码错了就也抛出登陆异常,异常信息为密码错误

如果用户名和密码都对了,输出: 欢迎xxx

说明:正确用户名和密码都是admin

### 操作步骤答案

/\*

1.定义异常类LoginException继承Exception

a)提供空参和有参构造

\*/

**public** **class** LoginException **extends** Exception {

**public** LoginException() {

**super**();

}

**public** LoginException(String message) {

// 一定要调用父类的构造方法

**super**(message);

}

}

/\*

2.创建测试类Test

a)提供一个用于登陆的方法login(String name,String pwd),在放方法中

i.如果用户名错误,就抛出自定义登陆异常(LoginException),异常信息为用户名不存在

ii.如果密码错了就也抛出登陆异常,异常信息为密码错误.

iii.如果能来到下面,就说明用户和密码都是对的,输出: 欢迎xxx

b)提供main方法,在main方法

i.调用login方法,传入错误用户名,运行程序,报运行时异常,然后注释这行代码

ii.调用login方法,传入正确用户名,错误的命名,运行程序,报运行时异常,然后注释这行代码

iii.调用login方法,传入正确的用户名和密码

\*/

**public** **class** Test16 {

// a)提供一个用于登陆的方法login(String name,String pwd),在放方法中

**public** **static** **void** login(String name,String pwd) **throws** LoginException{

// i.如果用户名错误,就抛出自定义登陆异常(LoginException),异常信息为用户名不存在

**if**(!"admin".equals(name)){

**throw** **new** LoginException("用户名:"+name+"不存在");

}

// ii.如果密码错了就也抛出登陆异常,异常信息为密码错误.

**if**(!"admin".equals(pwd)){

**throw** **new** LoginException("密码错误");

}

// iii.如果能来到下面,就说明用户和密码都是对的,输出: 欢迎xxx

System.***out***.println("欢迎"+name);

}

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

**try** {

// i.调用login方法,传入错误用户名,运行程序,报运行时异常,然后注释这行代码

// login("ax", "xx");

// ii.调用login方法,传入正确用户名,错误的命名,运行程序,报运行时异常,然后注释这行代码

// login("admin", "123");

// iii.调用login方法,传入正确的用户名和密码

*login*("admin", "admin");

} **catch** (LoginException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

# 关卡3

## 训练案例1

### 训练描述

已知集合ArrayList中有, ”张三丰”,”景甜”,”林思意”,”鹿晗”;

把集合中人名写入当前项目的根目录下的names.txt中,每一个人名占一行;

要求: 使用try...catch...finally对异常进行处理

### 操作步骤

1. 创建包:com.itheima.level3\_01
2. 创建类Test17
3. 提供main方法,在main方法中
   1. 创建ArrayList集合对象list
   2. 向集合list中添加人名
   3. 定义BufferedWriter变量 br,初始化值为null
   4. 写try...catch...finally代码块
   5. 在try代码块中
      1. 创建BufferedWriter对象,绑定目标names.txt,赋值bw
      2. 遍历list集合,在遍历的过程中,写一个人名,写一个换行,再flush
   6. 在catch代码块中,打印异常信息
   7. 在finally代码块中关闭流
      1. 如果bw不为null,就调用bw.close()方法
      2. 使用try...catch处理异常

### 操作步骤答案

/\*

1.创建包:com.itheima.level3\_01

2.创建类Test17

3.提供main方法,在main方法中

a)创建ArrayList集合对象list

b)向集合list中添加人名

c)定义BufferedWriter变量 br,初始化值为null

d)写try...catch...finally代码块

e)在try代码块中

i.创建BufferedWriter对象,绑定目标names.txt,赋值bw

ii.遍历list集合,在遍历的过程中,写一个人名,写一个换行,再flush

f)在catch代码块中,打印异常信息

g)在finally代码块中关闭流

i.如果bw不为null,就调用bw.close()方法

ii.使用try...catch处理异常

\*/

**public** **class** Test17 {

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

// a)创建ArrayList集合对象list

ArrayList<String> list = **new** ArrayList<>();

// b)向集合list中添加人名

list.add("张三丰");

list.add("景甜");

list.add("林思意");

list.add("鹿晗");

// c)定义BufferedWriter变量 br,初始化值为null

BufferedWriter bw = **null**;

// d)写try...catch...finally代码块

**try**{

// e)在try代码块中

// i.创建BufferedWriter对象,绑定目标names.txt,赋值bw

bw = **new** BufferedWriter(**new** FileWriter("names.txt"));

// ii.遍历list集合,在遍历的过程中,写一个人名,写一个换行,再flush

**for**(**int** i=0; i<list.size(); i++){

String name = list.get(i);

bw.write(name);

bw.newLine();

bw.flush();

}

}**catch**(IOException e){

// f)在catch代码块中,打印异常信息

e.printStackTrace();

}**finally**{

// g)在finally代码块中关闭流

// i.如果bw不为null,就调用bw.close()方法

**try**{

**if**(bw != **null**){

bw.close();

}

}**catch**(IOException ex){

// ii.使用try...catch处理异常

ex.printStackTrace();

}

}

}

}

## 训练案例2

### 训练描述

请使用代码描述

把names.txt中的人名,读取到程序中,存储到ArrayList集合中;每一个人名为ArrayList集合中一个元素;

要求:使用try...catch...finally对异常进行处理

### 操作步骤

1. 创建包:com.itheima.level3\_02
2. 创建类Test18
3. 提供main方法,在main方法中
   1. 创ArrayList集合对象,用于存储姓名
   2. 定义BufferedReader类型变量 br,初始值为null
   3. 写try..catch...finally代码块
   4. 在try代码块中
      1. 创建BufferedReader对象,绑定names.txt文件,赋值给br
      2. 定义字符串变量line,用于存储读取到的名称
      3. 使用循环,每次读一行,添加到集合中
   5. 在catch代码块中,打印异常信息
   6. 在finnaly代码块中释放资源
      1. 如果br不为null,就调用br的close方法
      2. 处理关闭异常
   7. 打印集合list

### 操作步骤答案

/\*

1.创建包:com.itheima.level3\_02

2.创建类Test18

3.提供main方法,在main方法中

a)创ArrayList集合对象,用于存储姓名

b)定义BufferedReader类型变量 br,初始值为null

c)写try..catch...finally代码块

d)在try代码块中

i.创建BufferedReader对象,绑定names.txt文件,赋值给br

ii.定义字符串变量line,用于存储读取到的名称

iii.使用循环,每次读一行,添加到集合中

e)在catch代码块中,打印异常信息

f)在finnaly代码块中释放资源

i.如果br不为null,就调用br的close方法

ii.处理关闭异常

g)打印集合list

\*/

**public** **class** Test18 {

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

// a)创ArrayList集合对象,用于存储姓名

ArrayList<String> list = **new** ArrayList<>();

// b)定义BufferedReader类型变量 br,初始值为null

BufferedReader br = **null**;

// c)写try..catch...finally代码块

**try**{

// d)在try代码块中

// i.创建BufferedReader对象,绑定names.txt文件,赋值给br

br = **new** BufferedReader(**new** FileReader("names.txt"));

// ii.定义字符串变量line,用于存储读取到的名称

String line;

// iii.使用循环,每次读一行,添加到集合中

**while**((line = br.readLine())!=**null**){

list.add(line);

}

}**catch**(IOException e){

// e)在catch代码块中,打印异常信息

e.printStackTrace();

}**finally**{

// f)在finnaly代码块中释放资源

**try**{

// i.如果br不为null,就调用br的close方法

**if**(br!=**null**){

br.close();

}

// ii.处理关闭异常

}**catch**(IOException e){

e.printStackTrace();

}

}

// g)打印集合list

System.***out***.println(list);

}

}

## 训练案例3

### 训练描述

请使用代码描述:

每一张借记卡(DebitCard)都有账号(accountNumber):和余额(money); 余额永远不能为负数

要求:当借记卡的余额为负数的时候需要抛出自定的异常

### 操作步骤描述

1. 创建包:com.itheima.level3\_03
2. 定义异常类无效金额异常(UnValidMoneyException),继承RuntimeExpetion
   1. 提供无参构造和有参构造
3. 定义借记卡类(DebitCard)
   1. 属性: 账号(accountNumber):和余额(money);
   2. 提供空参构造
   3. 提供有参数构造
      1. 有参构造需使用setXxx方法给对应的成员变量赋值
   4. 提供setter和getter方法
      1. 在setMoney(int money)中
      2. 如果money为负数,抛出无效的金额异常UnValidMoneyException
      3. 最后给成员变量money赋值
4. 定义Test19类,提供main方法,在main方法中
   1. 使用有参构造创建,创建DebitCard对象,金额传入一个负数
   2. 运行测试,注释代码
   3. 使用空参构造创建DebitCard的对象 dc
   4. 使用setMoney方法传入一个大于0的数,运行程序
   5. 使用setMoney方法传入一个小于0的数,运行程序

### 操作步骤答案

/\*

1.创建包:com.itheima.level3\_03

2.定义异常类无效金额异常(UnValidMoneyException),继承RuntimeExpetion

a)提供无参构造和有参构造

\*/

**public** **class** UnValidMoneyException **extends** RuntimeException {

**public** UnValidMoneyException() {

**super**();

}

**public** UnValidMoneyException(String message) {

**super**(message);

}

}

/\*

3.定义借记卡类(DebitCard)

a)属性: 账号(accountNumber):和余额(money);

b)提供空参构造

c)提供有参数构造

i.有效参构造需使用setXxx方法给对应的成员变量赋值

d)提供setter和getter方法

i.在setMoney(int money)中

ii.如果money为负数,抛出无效的金额异常UnValidMoneyException

iii.最后给成员变量money赋值

\*/

**public** **class** DebitCard {

// 账号(accountNumber)

**private** String accountNumber;

// 余额(money);

**private** **int** money;

// b)提供空参构造

**public** DebitCard() {

**super**();

}

// c)提供有参数构造

**public** DebitCard(String accountNumber, **int** money) {

**super**();

// i. 使用setXxx方法给对应的成员变量赋值

setAccountNumber(accountNumber);

setMoney(money);

}

// d)提供setter和getter方法

**public** String getAccountNumber() {

**return** accountNumber;

}

**public** **void** setAccountNumber(String accountNumber) {

**this**.accountNumber = accountNumber;

}

**public** **int** getMoney() {

**return** money;

}

// i.在setMoney(int money)中

**public** **void** setMoney(**int** money) {

// ii.如果money为负数,抛出无效的金额异常UnValidMoneyException

**if**(money < 0){

**throw** **new** UnValidMoneyException("余额无效:"+money);

}

// iii.最后给成员变量money赋值

**this**.money = money;

}

}

/\*

4.定义Test19类,提供main方法,在main方法中

a)使用有参构造创建,创建DebitCard对象,金额传入一个负数

b)运行测试,注释代码

c)使用空参构造创建DebitCard的对象 dc

d)使用setMoney方法传入一个大于0的数,运行程序

e)使用setMoney方法传入一个小于0的数,运行程序

\*/

**public** **class** Test19 {

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

// a)使用有参构造创建,创建DebitCard对象,金额传入一个负数

// DebitCard dc = new DebitCard("001", -100);

// b)运行测试,注释代码

// c)使用空参构造创建DebitCard的对象 dc

DebitCard dc = **new** DebitCard();

// d)使用setMoney方法传入一个大于0的数,运行程序

dc.setMoney(100);

// e)使用setMoney方法传入一个小于0的数,运行程序

dc.setMoney(-1);

}

}