

第 10 章

常用数学工具类

本章读者可以学到如下实例：

- ▶ 实例 110 角度和弧度的转换
- ▶ 实例 111 三角函数的使用
- ▶ 实例 112 反三角函数的使用
- ▶ 实例 113 双曲函数的使用
- ▶ 实例 114 指数与对数运算
- ▶ 实例 115 高精度整数运算
- ▶ 实例 116 高精度浮点运算
- ▶ 实例 117 七星彩号码生成器
- ▶ 实例 118 大乐透号码生成器



实例 110 角度和弧度的转换

(实例位置: 配套资源\SL\10\110)

实例说明

有两种单位可以用来度量角的大小, 即角度和弧度。通常在三角运算中使用弧度比较方便, 在表示角的大小时使用角度比较方便, 本实例将实现角度和弧度之间的转换。实例的运行效果如图 10.1 所示。

实现过程

(1) 在 Eclipse 中新建项目 110, 在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建 RadianTest 类, 在该类的 main()方法中, 使用 Math 类的方法输出 30° 、 45° 角对应的弧度值和 $\pi/6$ 、 $\pi/4$ 弧度对应的角度值。关键代码如下:

```
public class RadianTest {
    public static void main(String[] args) {
        System.out.println("30° 对应的弧度是: " + Math.toRadians(30));
        System.out.println("π/6 对应的角度是: " + Math.toDegrees(Math.PI / 6));
        System.out.println("45° 对应的弧度是: " + Math.toRadians(45));
        System.out.println("π/4 对应的角度是: " + Math.toDegrees(Math.PI / 4));
    }
}
```

技术要点

本实例主要使用了 Math 类的 toRadians() 和 toDegrees() 方法实现了角度与弧度的转换。

1. toRadians()方法

Math 类的 toRadians() 方法用于将角度转换为弧度, 该方法是一个静态方法, 可以通过类名直接调用。其语法如下:

```
public static double toRadians(double angdeg)
```

参数说明

angdeg: 用角度表示的角。

返回值: 角 angdeg 用弧度表示的值。

2. toDegrees()方法

Math 类的 toDegrees() 方法用于将弧度转换为角度, 该方法也是一个静态方法, 可以通过类名直接调用。其语法如下:

```
public static double toDegrees(double angrad)
```

参数说明

angrad: 用弧度表示的角。

返回值: 角 angrad 用角度表示的值。

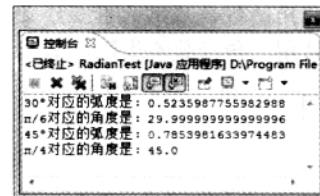


图 10.1 角度和弧度的转换



Note



实例 111 三角函数的使用

(实例位置: 配套资源\SL\10\111)



Note

实例说明

三角函数是数学的重要分支之一,很多问题使用三角函数来解决能容易很多。Math 类提供了常用三角函数的实现,本实例将演示它们的用法。实例的运行效果如图 10.2 所示。

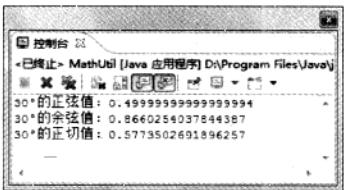


图 10.2 输出 30° 的正弦值、余弦值和正切值

指点迷津:

由于虚拟机原因,浮点运算结果并不精确,所以使用三角函数得到的结果也是不精确的。

实现过程

(1) 在 Eclipse 中新建项目 111, 在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建 MathUtil 类,在该类的 main()方法中输出了 30° 的正弦值、余弦值和正切值。关键代码如下:

```
public class MathUtil {  
    public static void main(String[] args) {  
        System.out.println("30° 的正弦值: " + Math.sin(Math.PI / 6)); //计算 30° 的正弦值  
        System.out.println("30° 的余弦值: " + Math.cos(Math.PI / 6)); //计算 30° 的余弦值  
        System.out.println("30° 的正切值: " + Math.tan(Math.PI / 6)); //计算 30° 的正切值  
    }  
}
```

技术要点

本实例主要使用了 Math 类的 sin()、cos() 和 tan() 方法实现了求指定角度的正弦值、余弦值和正切值。

1. sin()方法

Math 类的 sin()方法用于求指定角的正弦值,该方法是一个静态方法,可以通过类名直接调用。其语法如下:

```
public static double sin(double a)
```

参数说明

a: 用弧度表示的角。

返回值: 角 a 的正弦值。



2. cos()方法

Math 类的 cos()方法用于求指定角的余弦值，该方法是一个静态方法，可以通过类名直接调用。其语法如下：

```
public static double cos(double a)
```

参数说明

a：用弧度表示的角。

返回值：角 a 的余弦值。

3. tan()方法

Math 类的 tan()方法用于求指定角的正切值，该方法是一个静态方法，可以通过类名直接调用。其语法如下：

```
public static double tan(double a)
```

参数说明

a：用弧度表示的角。

返回值：角 a 的正切值。



Note

实例 112 反三角函数的使用

(实例位置：配套资源\SL\10\112)

实例说明

反三角函数通常用于获得某些三角函数值所对应的弧度，进而转换为角度，以满足生产和生活中不同角度的应用。Math 类提供了常用反三角函数的实现，本实例将演示反三角函数的用法。实例的运行效果如图 10.3 所示。



图 10.3 输出指定值的反正弦值、反余弦值和反正切值

指点迷津：

反三角函数返回的值是以弧度为单位的角，该值是一个近似值，图 10.3 中的 0.5、0.866 和 0.5774 分别是 30° 对应弧度值（ 30° 对应的弧度近似值为 0.524）的正弦值、余弦值和正切值的近似值。

实现过程

- (1) 在 Eclipse 中新建项目 112，在项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建 MathUtil 类，在该类的 main()方法中，分别输出 0.5、0.866 和 0.5774 的反正弦值、反余弦值和反正切值。关键代码如下：

```
public class MathUtil {
    public static void main(String[] args) {
        System.out.println("0.5 的反正弦值：" + Math.asin(0.5));           //计算 0.5 的反正弦值
        System.out.println("0.866 的反余弦值：" + Math.acos(0.866));       //计算 0.866 的反余弦值
        System.out.println("0.5774 的反正切值：" + Math.tan(0.5774));      //计算 0.5774 的反正切值
    }
}
```



```

        System.out.println("0.866 的反余弦值: " + Math.acos(0.866)); //计算 0.866 的反余弦值
        System.out.println("0.5774 的反正切值: " + Math.atan(0.5774)); //计算 0.5774 的反正切值
    }
}

```



Note

技术要点

本实例主要使用了 Math 类的 `asin()`、`acos()` 和 `atan()` 方法实现了求指定值的反正弦值、反余弦值和反正切值。

1. `asin()`方法

Math 类的 `asin()` 方法用于求指定值的反正弦值，该方法是一个静态方法，可以通过类名直接调用。其语法如下：

```
public static double asin(double a)
```

参数说明

a: 要返回其反正弦的值。

返回值：值 a 的反正弦值。

2. `acos()`方法

Math 类的 `acos()` 方法用于求指定值的反余弦值，该方法是一个静态方法，可以通过类名直接调用。其语法如下：

```
public static double acos(double a)
```

参数说明

a: 要返回其反余弦的值。

返回值：值 a 的反余弦值。

3. `atan()`方法

Math 类的 `atan()` 方法用于求指定值的反正切值，该方法是一个静态方法，可以通过类名直接调用。其语法如下：

```
public static double atan(double a)
```

参数说明

a: 要返回其反正切的值。

返回值：值 a 的反正切值。

实例 113 双曲函数的使用

(实例位置：配套资源\SL\10\113)

实例说明

双曲函数在物理学中有重要应用，如阻尼落体、导电电容等。Math 类提供了常用双曲函数的实现，本实例将演示它们的用法。实例的运行效果如图 10.4 所示。

实现过程

(1) 在 Eclipse 中新建项目 113，在项目中创建 `com.mingrisoft` 包。



图 10.4 输出 30 的指定双曲函数值



(2) 在 com.mingrisoft 包中创建 MathUtil 类, 在该类的 main()方法中输出 30 的双曲正弦值、双曲余弦值和双曲正切值。代码如下:

```
public class MathUtil {
    public static void main(String[] args) {
        System.out.println("30 的双曲正弦值: " + Math.sinh(30)); //计算 30 的双曲正弦值
        System.out.println("30 的双曲余弦值: " + Math.cosh(30)); //计算 30 的双曲余弦值
        System.out.println("30 的双曲正切值: " + Math.tanh(30)); //计算 30 的双曲正切值
    }
}
```

**Note**

技术要点

本实例主要使用了 Math 类的 sinh()、cosh() 和 tanh() 方法实现了求指定值的双曲正弦值、双曲余弦值和双曲正切值。

1. sinh()方法

Math 类的 sinh() 方法用于求指定值的双曲正弦值, 该方法是一个静态方法, 可以通过类名直接调用。其语法如下:

```
public static double sinh(double a)
```

参数说明

a: 要返回其双曲正弦的数值。

返回值: 值 a 的双曲正弦值。

2. cosh()方法

Math 类的 cosh() 方法用于求指定值的双曲余弦值, 该方法是一个静态方法, 可以通过类名直接调用。其语法如下:

```
public static double cosh(double a)
```

参数说明

a: 要返回其双曲余弦的数值。

返回值: 值 a 的双曲余弦值。

3. tanh()方法

Math 类的 tanh() 方法用于求指定值的双曲正切值, 该方法是一个静态方法, 可以通过类名直接调用。其语法如下:

```
public static double tanh(double a)
```

参数说明

a: 要返回其双曲正切的数值。

返回值: 值 a 的双曲正切值。

实例 114 指数与对数运算

(实例位置: 配套资源\SL\10\114)

实例说明

指数与对数运算是初等函数的重点, 它们在数学分析中有重要的应用。Math 类提供了常用指数与对数运算的实现, 本实例将演示它们的用法。实例的运行效果如图 10.5 所示。



图 10.5 指数与对数运算的结果

实现过程

- (1) 在 Eclipse 中新建项目 114，在项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建 MathUtil 类，在该类的 main()方法中，实现求指数、对数和方根的运算。关键代码如下：

```
public class MathUtil {
    public static void main(String[] args) {
        System.out.println("8 的立方根是: " + Math.cbrt(8));
        System.out.println("e 的 8 次方是: " + Math.exp(8));
        System.out.println("e 的 9 次方是: " + Math.expm1(8));
        System.out.println("8 的自然对数是: " + Math.log(8));
        System.out.println("8 的 10 为底的对数是: " + Math.log10(8));
        System.out.println("9 的自然对数是: " + Math.log1p(8));
        System.out.println("2 的 3 次方是: " + Math.pow(2, 3));
        System.out.println("8 的平方根是: " + Math.sqrt(8));
    }
}
```

技术要点

本实例主要使用了 Math 类的 cbrt()、exp()、expml()、log()、log10()、log1p()、pow() 和 sqrt() 方法实现了求指定值的方根、对数和平方根等运算。

实例 115 高精度整数运算

(实例位置：配套资源\SL\10\115)

实例说明

为了弥补虚拟机在高精度计算方面的不足，Java 推出了 BigInteger 类，它可以用来完成任意精度的整数运算。本实例将演示其基本的四则运算。实例的运行效果如图 10.6 所示。

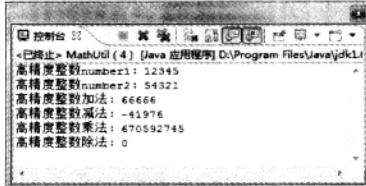


图 10.6 高精度整数运算



指点迷津：

使用该类虽然能大幅度提高运算的精度，但是牺牲的却是性能，因此对于普通运算不推荐使用。

实现过程

- (1) 在 Eclipse 中新建项目 115，在项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建 MathUtil 类，在 main()方法中创建两个 BigInteger 对象，并演示基本的四则运算。关键代码如下：

```
import java.math.BigInteger;
public class MathUtil {
    public static void main(String[] args) {
        BigInteger number1 = new BigInteger("12345");           //声明高精度整数 number1
        BigInteger number2 = new BigInteger("54321");           //声明高精度整数 number2
        BigInteger addition = number1.add(number2);             //计算 number1 加 number2
        BigInteger subtraction = number1.subtract(number2);      //计算 number1 减 number2
        BigInteger multiplication = number1.multiply(number2); //计算 number1 乘 number2
        BigInteger division = number1.divide(number2);          //计算 number1 除 number2
        System.out.println("高精度整数 number1: " + number1);
        System.out.println("高精度整数 number2: " + number2);
        System.out.println("高精度整数加法: " + addition);
        System.out.println("高精度整数减法: " + subtraction);
        System.out.println("高精度整数乘法: " + multiplication);
        System.out.println("高精度整数除法: " + division);
    }
}
```

指点迷津：

BigInteger 没有 int、long 等类型的构造方法，所以本实例使用字符串来构造高精度整数。

技术要点

BigInteger 类可以表示不可变的、任意精度的整数。所有操作中，都以二进制补码形式表示 BigInteger（如 Java 的基本整数类型）。BigInteger 提供了所有 Java 的基本整数操作符的对应物，并提供了 java.lang.Math 的所有相关方法。另外，BigInteger 还提供了以下运算：模算术、GCD 计算、质数测试、素数生成、位操作以及一些其他操作。本实例使用 BigInteger 类的 add()、subtract()、multiply() 和 divide() 方法实现了加、减、乘、除四则运算。

实例 116 高精度浮点运算

（实例位置：配套资源\SL\10\116）

实例说明

为了弥补虚拟机在高精度计算方面的不足，Java 还提供了 BigDecimal 类，它可以用来完



Note



Note

成任意精度的浮点运算。本实例将演示其基本的四则运算。实例的运行效果如图 10.7 所示。

```

<已停止> MathUtil (5) [Java 应用程序 D:\Program Files\Java\jdk1.6.0_17\bin\javaw.exe] (2011-5-19 下午 01:32:46)
高精度浮点数 number1: 1.234999999999999307220832633902318775653839111328125
高精度浮点数 number2: 5.432100000000000150635059981141239406783447265625
高精度浮点数加法: 6.666600000000000813571432445314712822437286376953125
高精度浮点数减法: -4.1976000000000002199129767177510075271129608154296875
高精度浮点数乘法: 6.7059274499999980963410041780928191515195520884985600575155587077524
8930867112297564745953408203125
高精度浮点数除法: 0.2272601756226873394246669017768510128696700540567039

```

图 10.7 高精度浮点数运算

指点迷津：

使用 `BigDecimal` 类虽然可以大幅度提高运算的精度，但是牺牲的却是系统的性能，因此对于普通运算也不推荐使用该类。

实现过程

- (1) 在 Eclipse 中新建项目 116，在项目中创建 `com.mingrisoft` 包。
- (2) 在 `com.mingrisoft` 包中创建 `MathUtil` 类，在 `main()` 方法中创建两个 `BigDecimal` 对象，并演示基本的四则运算。关键代码如下：

```

import java.math.BigDecimal;
import java.math.RoundingMode;
public class MathUtil {
    public static void main(String[] args) {
        BigDecimal number1 = new BigDecimal(1.2345);           // 声明高精度浮点数 number1
        BigDecimal number2 = new BigDecimal(5.4321);           // 声明高精度浮点数 number2
        BigDecimal addition = number1.add(number2);           // 计算 number1 加 number2
        BigDecimal subtraction = number1.subtract(number2);   // 计算 number1 减 number2
        BigDecimal multiplication = number1.multiply(number2); // 计算 number1 乘 number2
        // 以四舍五入的方式获得高精度除法运算的结果
        BigDecimal division = number1.divide(number2, RoundingMode.HALF_UP);
        System.out.println("高精度浮点数 number1: " + number1);
        System.out.println("高精度浮点数 number2: " + number2);
        System.out.println("高精度浮点数加法: " + addition);
        System.out.println("高精度浮点数减法: " + subtraction);
        System.out.println("高精度浮点数乘法: " + multiplication);
        System.out.println("高精度浮点数除法: " + division);
    }
}

```

技术要点

`BigDecimal` 表示不可变的、任意精度的有符号十进制数。`BigDecimal` 由任意精度的整数非标度值和 32 位的整数标度（scale）组成。如果为零或正数，则标度是小数点后的位数；如果为负数，则将该数的非标度值乘以 10 的负 scale 次幂。因此，`BigDecimal` 表示的数值是 $(\text{unscaledValue} \times 10^{-\text{scale}})$ 。本实例使用 `BigInteger` 类的 `add()`、`subtract()`、`multiply()` 和 `divide()` 方法实现了加、减、乘、除四则运算。



实例 117 七星彩号码生成器

(实例位置: 配套资源\SL\10\117)



Note

实例说明

七星彩是中国体彩推出的一种彩票,其基本玩法是:从0~9十个数字中随机选择一个,一共选择7次,组成一个7位数。如果完全和中奖号码相同则中一等奖。本实例将实现一个七星彩号码生成器,实例的运行效果如图10.8所示。

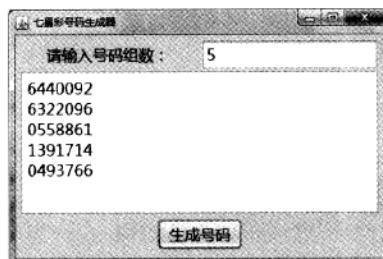


图 10.8 七星彩号码生成器

实现过程

(1) 在 Eclipse 中新建项目 117, 在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建继承 JFrame 的窗体类 SevenStar, 在该窗体中添加相应控件, 其中获得用户输入号码组数的 JTextField 的名称为 textField, 显示生成号码的 JTextArea 的名称为 textArea, “生成号码”按钮的名称为 button。

(3) 在“生成号码”按钮的事件中, 根据用户输入的组数, 实现随机生成七星彩号码的功能。关键代码如下:

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        int times = Integer.parseInt(textField.getText()); //获得用户输入的需要生成的中奖号码个数
        //省略提示购买数量太多的代码
        StringBuilder sb = new StringBuilder(); //利用 StringBuilder 类保存彩票中奖号码
        for (int i = 0; i < times; i++) {
            int number = new Random().nextInt((int) Math.pow(10, 7)); //生成随机数
            String luckNumber = "" + number;
            while (luckNumber.length() < 7) {
                luckNumber = "0" + luckNumber; //如果随机数长度不够 7 位用 0 补齐
            }
            sb.append(luckNumber + "\n");
        }
        textArea.setText(sb.toString()); //显示生成的中奖号码
    }
});
```



技术要点

通过 Random 类的实例生成伪随机数流。该类提供了常用的伪随机数生成方法，类型包括 boolean、int、long、double 等。本实例使用 setSeed()方法设置随机数种子值，使用 nextInt()方法获得一个小于参数值的随机整数。



Note

1. setSeed()方法

Random 类的 setSeed()方法使用单个 long 种子，设置此随机数生成器的种子值。其语法如下：

```
public void setSeed(long seed)
```

参数说明

seed：为随机数生成器设置的种子值。

2. nextInt()方法

Random 类的 nextInt()方法用于返回一个伪随机数，它是取自此随机数生成器序列在 0（包括）和指定值（不包括）之间均匀分布的 int 值。其语法如下：

```
public int nextInt(int n)
```

参数说明

n：要返回的随机数的范围。必须为正数。

返回值：下一个伪随机数，它是取自此随机数生成器序列中 0（包括）和 n（不包括）之间均匀分布的 int 值。

实例 118 大乐透号码生成器

(实例位置：配套资源\SL\10\118)

实例说明

大乐透是中国体彩推出的一种彩票，其基本玩法是：从 1~35 随机选取不重复的 5 个数字，从 1~12 随机选取不重复的两个数字组成一个七位数。如果完全和中奖号码相同则中一等奖。本实例将实现一个大乐透号码生成器，实例的运行效果如图 10.9 所示。

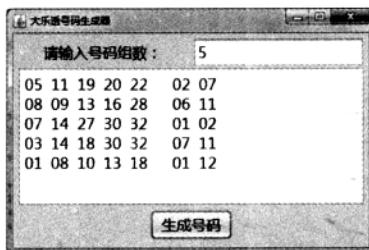


图 10.9 大乐透号码生成器

实现过程

(1) 在 Eclipse 中新建项目 118，在项目中创建 com.mingrisoft 包。



(2) 在 com.mingrisoft 包中创建继承 JFrame 的窗体类 SuperFun，在该窗体中添加相应控件，其中获得用户输入号码组数的 JTextField 的名称为 textField，显示生成号码的 JTextArea 的名称为 textArea，“生成号码”按钮的名称为 button。

(3) 在窗体类 SuperFun 中，创建生成前段号码的 getStartNumber()方法。关键代码如下：

```
public List<String> getStartNumber() {
    List<String> list = new ArrayList<String>(); // 创建前段号码集合
    String luckyNumber = "";
    for (int i = 1; i < 36; i++) { // 初始化前段号码集合
        if (i < 10) {
            list.add("0" + i + " ");
        } else {
            list.add(" " + i + " ");
        }
    }
    int roundIndex = 0;
    List<String> luckylist = new ArrayList<String>(); // 保存前段号码的 List 集合
    for (int j = 0; j < 5; j++) {
        int amount = list.size(); // 获取前段号码的个数
        Random r = new Random(); // 创建并实例化 Random 的对象
        roundIndex = r.nextInt(amount); // 获取一个 0~amount-1 的随机数
        luckyNumber = list.get(roundIndex); // 获取幸运数字
        luckylist.add(luckyNumber); // 添加到 luckylist 中
        list.remove(roundIndex); // 移除刚刚产生的号码
    }
    Collections.sort(luckylist); // 对前段号码进行排序
    return luckylist;
}
```

(4) 在窗体类 SuperFun 中，创建生成后段号码的 getEndNumber()方法。关键代码如下：

```
public List<String> getEndNumber() {
    List<String> list = new ArrayList<String>(); // 创建后段号码集合
    String luckyNumber = "";
    for (int i = 1; i < 13; i++) { // 初始化后段号码集合
        if (i < 10) {
            list.add("0" + i + " ");
        } else {
            list.add(" " + i + " ");
        }
    }
    int roundIndex = 0;
    List<String> luckylist = new ArrayList<String>(); // 保存后段号码的 List 集合
    for (int j = 0; j < 2; j++) {
        int amount = list.size(); // 获取后段号码的个数
        Random r = new Random(); // 创建并实例化 Random 的对象
        roundIndex = r.nextInt(amount); // 获取一个 0~amount-1 的随机数
        luckyNumber = list.get(roundIndex); // 获取幸运数字
        luckylist.add(luckyNumber); // 添加到 luckylist 中
        list.remove(roundIndex); // 移除刚刚产生的号码
    }
}
```





```
Collections.sort(luckylist); //对后段号码进行排序  
return luckylist;  
}
```

(5) 在“生成号码”按钮的事件中，根据用户输入的组数，实现随机生成大乐透号码的功能。关键代码如下：

```
Note button.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        int times = Integer.parseInt(textField.getText()); //获得用户输入的需要生成的中奖号码个数  
        //省略提示购买数量太多的代码  
        StringBuilder sb = new StringBuilder(); //创建字符串生成器对象  
        for (int i = 0; i < times; i++) {  
            List<String> startList = getStartNumber(); //获得前段号码的集合  
            List<String> endList = getEndNumber(); //获得后段号码的集合  
            for (int m = 0; m < startList.size(); m++) {  
                sb.append(startList.get(m)); //在字符串生成器中添加前段号码  
            }  
            sb.append("      ");  
            for (int n = 0; n < endList.size(); n++) {  
                sb.append(endList.get(n)); //在字符串生成器中添加后段号码  
            }  
            sb.append("\n");  
        }  
        textArea.setText(sb.toString()); //在文本域中显示号码  
    }  
});
```

技术要点

本实例使用 List 集合存储号码，然后从该 List 集合中随机获得号码，由于大乐透的每一组号码中的前段号码或后段号码是不允许重复的，因此，从 List 集合中随机获得号码后，就将所获得的号码从该 List 集合中移除，这样可以避免获得的号码重复。

第11章

错误处理

本章读者可以学到如下实例：

- ▶ 实例 119 算数异常
- ▶ 实例 120 数组下标越界异常
- ▶ 实例 121 空指针异常
- ▶ 实例 122 类未发现异常
- ▶ 实例 123 非法访问异常
- ▶ 实例 124 文件未发现异常
- ▶ 实例 125 数据库操作异常
- ▶ 实例 126 方法中抛出异常
- ▶ 实例 127 方法上抛出异常
- ▶ 实例 128 自定义异常类
- ▶ 实例 129 捕获单个异常
- ▶ 实例 130 捕获多个异常



实例 119 算数异常

(实例位置: 配套资源\SLAF119)



Note

实例说明

算数异常即 `ArithmaticException`, 是指整数被 0 除产生的异常。在 Java 中, 如果一个整数被 0 除, 那么将抛出 `ArithmaticException` 异常, 但是浮点数被 0 除, 将不引发算数异常, 这与数学中不同。本实例将演示出现算数异常的情况, 并进行处理。实例的运行效果如图 11.1 所示。



图 11.1 算数异常

实现过程

- (1) 在 Eclipse 中创建项目 119, 并在该项目中创建 `com.mingisoft` 包。
- (2) 在 `com.mingisoft` 包中创建类文件, 名称为 `ExceptionTest`。在该类的主方法中, 演示出现算数异常的情况。这里将第一条可能出现异常的语句应用 `try...catch` 语句捕获, 并输出异常信息; 第二条可能出现异常的语句不做处理。关键代码如下:

```
public class ExceptionTest {
    public static void main(String[] args) {
        System.out.println("-1.0 / 0 = " + (-1.0 / 0));           //演示负浮点数除 0
        System.out.println("+1.0 / 0 = " + (+1.0 / 0));         //演示正浮点数除 0
        try {
            System.out.println("-1 / 0 = " + (-1 / 0));          //演示负整数除 0
        } catch(Exception e) {
            System.out.println("抛出异常: " + e.getMessage());
        }
        System.out.println("+1 / 0 = " + (+1 / 0));           //演示正整数除 0
        System.out.println("输出结束。");
    }
}
```

指点迷津:

由于第二条可能出现异常的语句没有捕获, 在发生异常时, 程序终止了, 所以最后一条语句并没有被输出。

技术要点

本实例的技术要点就是何时抛出算数异常, 以及如何处理。在 Java 程序运行, 系统检查到整数被 0 除的情况下, 它将构造一个新的异常对象, 然后引发该异常。这时将导致出现异常语



句后面的语句不会被执行，也就是终止程序的执行。为了让后面的语句继续执行，可以应用 try...catch 语句捕获该异常，并进行处理，这样将不影响后面语句的执行。

指点迷津：

在 Java 的异常处理机制中，有一个默认处理异常的程序。当程序出现异常时，默认处理程序将显示一个描述异常的字符串，打印异常发生处的堆栈轨迹，并终止程序。



Note

实例 120 数组下标越界异常

(实例位置：配套资源\SL\11\120)

实例说明

数组下标越界异常即 `ArrayIndexOutOfBoundsException`，当访问的数组元素的下标值大于数组的最大下标值时发生，也就是数组元素的下标值大于等于数组的长度时发生。本实例将演示出现数组下标越界异常（`ArrayIndexOutOfBoundsException`）的情况。实例的运行效果如图 11.2 所示。



图 11.2 数组下标越界异常

实现过程

- (1) 在 Eclipse 中创建项目 120，并在该项目中创建 `com.mingrisoft` 包。
- (2) 在 `com.mingrisoft` 包中创建类文件，名称为 `ArrayExceptionTest`。在该类的 `main()` 方法中，首先声明一个长度为 5 的整型数组，并应用 `Arrays` 对象的 `fill()` 方法，将数组中所有元素赋值为 8，然后应用 `for` 循环遍历输出所有数组元素。关键代码如下：

```

public class ArrayExceptionTest {
    public static void main(String[] args) {
        int[] array = new int[5];           // 声明一个长度为 5 的整型数组
        Arrays.fill(array, 8);             // 将新声明数组的所有元素赋值为 8
        for (int i = 0; i < 6; i++) {       // 遍历输出所有数组元素
            System.out.println("array[" + i + "] = " + array[i]);
        }
    }
}
    
```

指点迷津：

上面代码运行时，当数组元素的下标值大于 4 时，将抛出 `ArrayIndexOutOfBoundsException` 异常，并终止程序的执行。



技术要点

本实例应用的主要技术就是数组下标越界异常。数组下标越界异常属于运行时错误，在程序编译阶段并不会产生错误，只有在程序运行时才有可能出现。例如，一个包含 6 个元素的数组，它的最大下标值应该是 5，如果在程序中访问 array[6]，将抛出数组下标越界异常。

Note

指点迷津：

如果要遍历数组中的全部元素，则推荐使用 foreach 循环，它可以避免数组的下标越界。如果要使用数组的下标，则需要记住数组的下标是从 0 开始计算的。如果需要使用数组的长度，则推荐使用 length 属性。另外使用 ArrayList 类也可以避免这些问题。

实例 121 空指针异常

(实例位置：配套资源\SL\11\121)

实例说明

空指针异常即 NullPointerException，当应用程序试图在需要对象的地方使用 null 时，将抛出该异常。本实例将演示出现空指针异常的情况。实例的运行效果如图 11.3 所示。

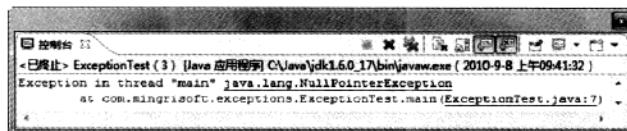


图 11.3 空指针异常

实现过程

(1) 在 Eclipse 中创建项目 l21，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建一个 Java 类，名称为 NullPointerE。在该类的 main()方法中定义一个字符串，并将其设置为 null，然后调用 toLowerCase()方法将字符串转换成小写并输出。关键代码如下：

```
public class NullPointerE {  
    @SuppressWarnings("null")  
    public static void main(String[] args) {  
        String string = null; // 将字符串设置为 null  
        System.out.println(string.toLowerCase()); // 将字符串转换成小写  
    }  
}
```

指点迷津：

上面的代码在执行时，由于使用了 null 值，所以会抛出 NullPointerException 异常。

技术要点

本实例应用的主要技术点就是空指针异常（NullPointerException）。当应用程序试图在需要



对象的地方使用 null 时，抛出 NullPointerException 异常。这种情况包括：

- 调用 null 对象的实例方法。
- 访问或修改 null 对象的字段。
- 将 null 作为一个数组，获得其长度。
- 将 null 作为一个数组，访问或修改其元素值。
- 将 null 作为 Throwable 值抛出。



Note

实例 122 类未发现异常

(实例位置：配套资源\SL\11\122)

实例说明

类未发现异常即 ClassNotFoundException，表示类没有找到。通常情况下，在进行数据连接时，需要借助第三方的数据库驱动包才能完成。如果代码中指定了所需的 Jar 包，而在程序中并没有提供所需的 Jar 包，在运行时将产生类未发现异常。本实例将演示抛出类未发现异常的情况。实例的运行效果如图 11.4 所示。

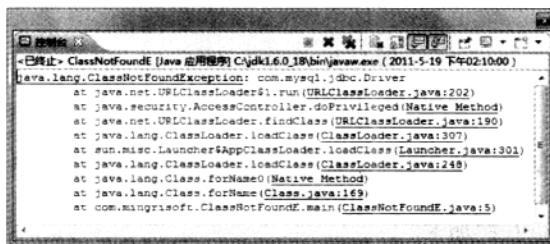


图 11.4 产生的类未发现异常

实现过程

- (1) 在 Eclipse 中创建项目 122，并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中新建一个 Java 类，名称为 ClassNotFoundE，在该类的 main() 方法中加载 MySQL 数据库的驱动，并应用 try…catch 语句捕获异常。关键代码如下：

```

public class ClassNotFoundE {
    public static void main(String[] args) {
        try {
            Class.forName("com.mysql.jdbc.Driver");           //加载 MySQL 驱动程序
        } catch (ClassNotFoundException e) {                  //捕获异常
            e.printStackTrace();                           //打印堆栈信息
        }
    }
}

```

技术要点

本实例应用的主要技术点就是类未发现异常。当应用程序试图使用以下方法通过字符串名加载类时，将抛出 ClassNotFoundException 异常。例如：

- Class 类中的 forName()方法。



- ClassLoader 类中的 findSystemClass()方法。
- ClassLoader 类中的 loadClass()方法。



Note

实例说明

非法访问异常即 `IllegalAccessException`，当不允许访问某类时发生。本实例将演示出现非法访问异常的情况。实例的运行效果如图 11.5 所示。

```
非法访问异常: java.lang.IllegalAccessException: Class com.mingrisoft.IllegalAccessE can not access a member of class java.lang.String with modifiers "private"
at sun.reflect.Reflection.ensureMemberAccess(Reflection.java:65)
at java.lang.reflect.Field.doSecurityCheck(Field.java:960)
at java.lang.reflect.Field.getFieldAccessor(Field.java:896)
at java.lang.reflect.Field.getInt(Field.java:199)
at com.mingrisoft.IllegalAccessE.main(IllegalAccessE.java:12)
```

图 11.5 非法访问异常

实现过程

- (1) 在 Eclipse 中创建项目 123，并在该项目中创建 `com.mingrisoft` 包。
- (2) 在 `com.mingrisoft` 包中编写类 `IllegalAccessE`，在该类的 `main()` 方法中，输出了可能发生异常的运行结果。关键代码如下：

```
import java.lang.reflect.Field;
public class IllegalAccessE {
    public static void main(String[] args) {
        Class<?> clazz = String.class; //获得代表 String 类的类对象
        Field[] fields = clazz.getDeclaredFields(); //获得 String 类的所有域
        for (Field field : fields) { //遍历所有域
            if (field.getName().equals("hash")) { //如果域的名字是 hash
                try {
                    System.out.println(field.getInt("hash")); //输出 hash 的值
                } catch (IllegalArgumentException e) { //捕获 IllegalArgumentException 异常
                    e.printStackTrace();
                } catch (IllegalAccessException e) { //捕获 IllegalAccessException 异常
                    e.printStackTrace();
                }
            }
        }
    }
}
```

技术要点

本实例应用的主要技术点就是非法访问异常，即 `IllegalAccessException`。当应用程序试图



反射性地创建一个实例（而不是数组）、设置或获取一个字段，或者调用一个方法时，并且当前正在执行的方法无法访问指定类、字段、方法或构造方法的定义时抛出的异常。

实例 124 文件未发现异常

（实例位置：配套资源\SL\11\124）



Note

实例说明

文件未发现异常即 `FileNotFoundException`，当要访问的文件找不到时发生。本实例将演示出现文件未发现异常的情况。实例的运行效果如图 11.6 所示。

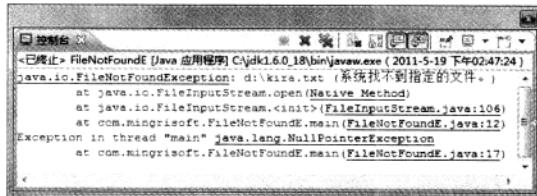


图 11.6 文件未发现异常

实现过程

- (1) 在 Eclipse 中创建项目 124，并在该项目中创建 `com.mingrisoft` 包。
- (2) 在 `com.mingrisoft` 包中编写名称为 `FileNotFoundException` 的类，在该类的 `main()` 方法中，捕获可能发生的异常并输出。关键代码如下：

```
public class FileNotFoundException {
    public static void main(String[] args) {
        FileInputStream fis = null; // 创建一个文件输入流对象
        try {
            File file = new File("d:\\kira.txt"); // 创建一个文件对象
            fis = new FileInputStream(file); // 初始化文件输入流对象
        } catch (FileNotFoundException e) { // 捕获异常
            e.printStackTrace();
        } finally {
            try {
                fis.close(); // 释放资源
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

指点迷津：

上面的代码在执行时，当 D 盘根目录下不存在 `kira.txt` 文件时，将会抛出 `FileNotFoundException` 异常。



技术要点

本实例应用的主要技术点就是文件未发现异常 (FileNotFoundException)。该异常当程序试图打开指定路径名表示的文件失败时抛出。在不存在具有指定路径名的文件时，此异常将由 FileInputStream、 FileOutputStream 和 RandomAccessFile 构造方法抛出。如果该文件存在，但是由于某些原因不可访问，例如试图打开一个只读文件进行写入，则此时这些构造方法仍然会抛出该异常。



Note

实例 125 数据库操作异常

(实例位置：配套资源\SL\11\125)

实例说明

数据库操作异常即 SQLException，通常发生在出现数据库访问错误时。本实例将演示出现数据库操作异常的情况。实例的运行效果如图 11.7 所示。

```

*已停止* SQL [Java 应用程序] C:\jdk1.6.0_18\bin\java.exe (2011-5-19 下午03:30:02)
java.sql.SQLException: Access denied for user 'mr'@'localhost' (using password: NO)
    at com.mysql.jdbc.SQLError.createSQLException (SQLError.java:1055)
    at com.mysql.jdbc.SQLError.createSQLException (SQLError.java:956)
    at com.mysql.jdbc.MysqlIO.checkErrorPacket (MysqlIO.java:3491)
    at com.mysql.jdbc.MysqlIO.checkErrorPacket (MysqlIO.java:3423)
    at com.mysql.jdbc.MysqlIO.secureAuth411 (MysqlIO.java:910)
    at com.mysql.jdbc.MysqlIO.secureAuth41 (MysqlIO.java:3923)
    at com.mysql.jdbc.MysqlIO.doHandshake (MysqlIO.java:1273)
    at com.mysql.jdbc.ConnectionImpl.createNewIO (ConnectionImpl.java:2031)
    at com.mysql.jdbc.ConnectionImpl.<init> (ConnectionImpl.java:716)
    at com.mysql.jdbc.JDBC4Connection.<init> (JDBC4Connection.java:46)
    at sun.reflect.NativeConstructorAccessorImpl.newInstance0 (Native Method)
    at sun.reflect.DelegatingConstructorAccessorImpl.newInstance (DelegatingConstructorAccessorImpl.java:27)
    at java.lang.reflect.Constructor.newInstance (Constructor.java:513)
    at com.mysql.jdbc.Util.handleNewInstance (Util.java:406)
    at com.mysql.jdbc.ConnectionImpl.getInstance (ConnectionImpl.java:302)
    at com.mysql.jdbc.NonRegisteringDriver.connect (NonRegisteringDriver.java:282)
    at java.sql.DriverManager.getConnection (DriverManager.java:582)
    at java.sql.DriverManager.getConnection (DriverManager.java:185)
    at com.mingrisoft.SQL.main (SQL.java:15)
Exception in thread "main" java.lang.NullPointerException
    at com.mingrisoft.SQL.main (SQL.java:22)


```

图 11.7 数据库操作异常

指点迷津：

在运行本实例时，需要将 MySQL 数据库的驱动包配置到构建路径中，否则将不能抛出 SQLException 异常，而是抛出 ClassNotFoundException 异常。

实现过程

- (1) 在 Eclipse 中创建项目 125，并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中编写类 ExceptionTest，在该类的 main()方法中，编写数据库连接的代码，并且捕获可能抛出的异常。关键代码如下：

```

public class ExceptionTest {
    public static void main(String[] args) {
        String URL = "jdbc:mysql://localhost:3306/db_database"; //MySQL 数据库的 URL
    }
}

```



```
String DRIVER = "com.mysql.jdbc.Driver"; //MySQL 数据库的驱动
String USERNAME = "mr"; //数据库的用户名
Connection connection = null;
try {
    Class.forName(DRIVER); //加载驱动
    connection = DriverManager.getConnection(URL, USERNAME, ""); //建立连接
} catch (SQLException e) { //捕获 SQLException 异常
    e.printStackTrace();
} catch (ClassNotFoundException e) { //捕获 ClassNotFoundException 异常
    e.printStackTrace();
} finally {
    try {
        connection.close(); //释放资源
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}
```



Note

技术要点

本实例应用的主要技术点就是数据库操作异常（`SQLException`）。该异常用于提供关于数据库访问错误或其他错误信息。提供的具体信息如下：

- 描述错误的字符串。
 - “SQLstate” 字符串，该字符串遵守 XOPEN SQLstate 约定或 SQL:2003 约定。
 - 特定于每个供应商的整数错误代码。
 - 到下一个 `Exception` 的链接。
 - 因果关系，如果存在任何导致此 `SQLException` 的原因。

实例 126 方法中抛出异常

(实例位置: 配套资源\SL\11\126)

实例说明

在项目开发中，通常是自顶向下进行的。在完成项目的整体设计后，需要对每个接口和类进行编写。如果一个类使用了其他类还没有实现的方法，则可以在实现其他类方法时让其抛出 `UnsupportedOperationException` 异常，以便在以后进行修改完成。实例的运行效果如图 11.8 所示。

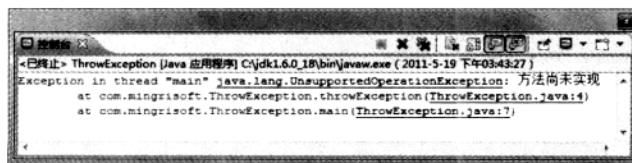


图 11.8 方法中抛出异常



实现过程



Note

- (1) 在 Eclipse 中创建项目 126，并在该项目中创建 com.mingrisoft 包。
(2) 在 com.mingrisoft 包中编写类 ThrowException，在该类中定义两个方法，一个是 throwException()方法，用于抛出异常；另一个是 main()方法，用于进行测试。关键代码如下：

```
public class ThrowException {  
    public static void throwException() {  
        throw new UnsupportedOperationException("方法尚未实现"); //抛出异常  
    }  
    public static void main(String[] args) {  
        ThrowException.throwException(); //调用抛出异常的方法  
    }  
}
```

技术要点

本实例应用的主要技术点就是使用 throw 关键字在方法中抛出异常。使用 throw 关键字可以在方法体中抛出异常。该异常既可以是系统预定义异常，又可以是用户自定义异常。其格式如下：
throw 异常对象；

throw 关键字可以抛出一个异常对象，并且仅可以应用在方法体中。

脚下留神：

请读者不要和 throws 关键字混淆。

实例 127 方法上抛出异常

(实例位置：配套资源\SL\11\127)

实例说明

在方法的执行过程中，如果存在可能遇到引发问题的因素，则应该在定义方法时加以说明。例如读取文件的方法可能遇到文件不存在的情况，此时需要在方法声明时抛出文件不存在异常。本实例将演示如何在方法上抛出异常。实例的运行效果如图 11.9 所示。

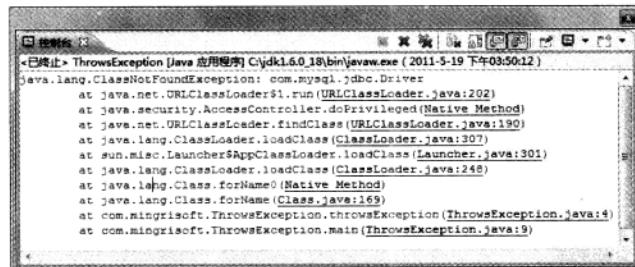


图 11.9 方法上抛出异常

实现过程

- (1) 在 Eclipse 中创建项目 127，并在该项目中创建 com.mingrisoft 包。



(2) 在 com.mingrisoft 包中编写类 ThrowsException，并且在该类中定义两个方法，一个是 throwException()方法，用于抛出异常；另一个是 main()方法，用于进行测试。关键代码如下：

```
public class ThrowsException {
    public static void throwsException() throws ClassNotFoundException { //抛出异常
        Class.forName("com.mysql.jdbc.Driver");
    }

    public static void main(String[] args) {
        try {
            ThrowsException	throwsException(); //捕获异常
        } catch (ClassNotFoundException e) { //调用抛出异常的方法
            e.printStackTrace();
        }
    }
}
```



Note

技术要点

本实例应用的主要技术点是使用 throws 关键字抛出异常。使用 throws 关键字可以在方法体外抛出异常。该异常既可以是系统预定义异常，又可以是用户自定义异常。下面以 FileInputStream 类的构造方法为例进行讲解，其声明如下：

```
public FileInputStream(String name) throws FileNotFoundException
```

该方法在定义时就抛出了 FileNotFoundException 异常，因此如果其他的类使用该方法，则必须捕获或者继续抛出该异常。throws 关键字可以声明抛出多个异常，并且仅可以应用在方法体外。

脚下留神：

请读者不要和 throw 关键字混淆。

实例 128 自定义异常类

(实例位置：配套资源\SL\11\128)

实例说明

在 Java SE API 中，已经定义了几十种异常类，它们基本包括了常用的异常类型。然而，实际开发中有可能遇到 Java SE API 中没有提供的情况，这时就需要自定义异常类了。本实例将演示如何自定义一个除零异常类。实例的运行效果如图 11.10 所示。

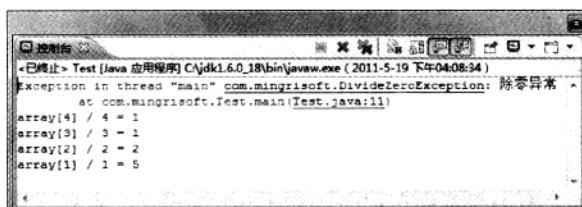


图 11.10 自定义除零异常类



实现过程



Note

- (1) 在 Eclipse 中创建项目 128，并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中编写类 DivideZeroException，该类继承自 ArithmeticException 并提供了两个构造方法。关键代码如下：

```
public class DivideZeroException extends ArithmeticException { //自定义异常类
    public DivideZeroException() {
    }
    public DivideZeroException(String msg) {
        super(msg);
    }
}
```

- (3) 在 com.mingrisoft 包中再编写类 Test 进行测试，在 main()方法中，抛出自定义的异常。关键代码如下：

```
public class Test {
    public static void main(String[] args) {
        int[] array = new int[5]; // 定义长度为 5 的数组
        Arrays.fill(array, 5); // 将数组中的元素赋值为 5
        for (int i = 4; i > -1; i--) {
            if (i == 0) { // 如果除 0
                throw new DivideZeroException("除零异常"); // 如果除零就抛出有异常信息
            }
            System.out.println("array[" + i + "] / " + i + " = " + array[i] / i);
        }
    }
}
```

技术要点

编写一个自定义异常类非常简单，只需要继承 Exception 或者 Exception 的子类。一个最简单的自定义异常类代码如下：

```
public class MRSsoft extends Exception{}
```

在自定义类时，推荐提供两个构造方法：一个无参数构造方法和一个字符串参数构造方法。它可以为调试提供更加详细的信息。

实例 129 捕获单个异常

(实例位置：配套资源\SL\11\129)

实例说明

当遇到异常时，除了可以将异常抛出，还可以将其捕获。抛出异常虽然简单，但是有时却不得不使用捕获来处理异常。如果程序遇到异常而没有捕获，则程序会直接退出。这在大多数情况下是不能被接受的，至少需要保存程序当前状态才能退出。本实例将演示如何捕获单个异常。实例的运行效果如图 11.11 所示。

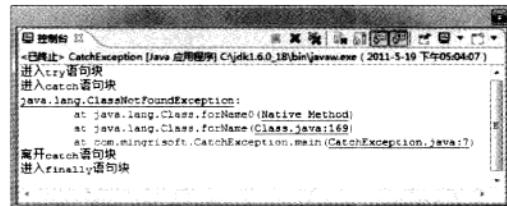


图 11.11 捕获单个异常

实现过程

- (1) 在 Eclipse 中创建项目 129，并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中编写类 CatchException，在该类的 main()方法中应用 try…catch…finally 语句捕获单个异常。关键代码如下：

```
public class CatchException {
    public static void main(String[] args) {
        try { // 定义 try 语句块
            System.out.println("进入 try 语句块");
            @SuppressWarnings("unused")
            Class<?> clazz = Class.forName("");
            System.out.println("离开 try 语句块");
        } catch (ClassNotFoundException e) { // 定义 catch 语句块
            System.out.println("进入 catch 语句块");
            e.printStackTrace();
            System.out.println("离开 catch 语句块");
        } finally { // 定义 finally 语句块
            System.out.println("进入 finally 语句块");
        }
    }
}
```

技术要点

本实例主要的技术点是捕获单个异常。Java 中捕获异常是通过 try…catch…finally 语句来完成的。其中 try 语句块是必需的，catch 和 finally 语句块可以选择一个或者两个。try 语句块用来放置可能出现问题的语句，catch 语句块用来放置异常发生后执行的代码，finally 语句块用来放置无论是否发生异常都需要执行的代码。

指点迷津：

捕获异常是一个高开销的操作，因此 try 语句块中的语句应该尽量少。

实例 130 捕获多个异常

(实例位置：配套资源\SL\11\130)

实例说明

在程序中，有时可能会遇到出现多个异常的情况，这时就需要分别捕获这些异常。本实例



将演示如何捕获多个异常。实例的运行效果如图 11.12 所示。

实现过程



(1) 在 Eclipse 中创建项目 130，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中编写类 CatchExceptions，在该类的 main() 方法中，应用 try…catch…finally 语句捕获多个异常。关键代码如下：

```
public class CatchExceptions {  
    private static String URL = "jdbc:mysql://localhost:3306/db_database"; //数据库 URL  
    private static String DRIVER = "com.mysql.jdbc.Driver"; //数据库驱动  
    private static String USERNAME = "mr"; //用户名  
    private static String PASSWORD = "mingri"; //密码  
    private static Connection conn;  
    public static Connection getConnection() {  
        try {  
            Class.forName(DRIVER); //加载驱动程序  
            conn = DriverManager.getConnection(URL, USERNAME, PASSWORD); //建立连接  
            return conn;  
        } catch (ClassNotFoundException e) { //捕获类未发现异常  
            e.printStackTrace();  
        } catch (SQLException e) { //捕获 SQL 异常  
            e.printStackTrace();  
        }  
        return null;  
    }  
    public static void main(String[] args) {  
        CatchExceptions.getConnection();  
    }  
}
```

指点迷津：

在上面的代码中首先捕获 ClassNotFoundException 异常，然后是 SQLException 异常。

技术要点

本实例主要的技术点是捕获多个异常。在 Java 中捕获异常是通过 try…catch…finally 语句来完成的。其中 try 语句块是必需的，catch 和 finally 语句块可以选择一个或者两个。try 语句块用来放置可能出现问题的语句，如果在 try 语句块中可能出现多个异常，则最好提供多个 catch 语句块来进行捕获，这样可以针对不同的异常提供不同的处理方案。如果 try 语句块中出现的异常和第一个 catch 语句块捕获的异常不匹配，JVM 将比较第二个 catch 语句块，依此类推，直到出现匹配的为止。如果没有找到匹配的，异常对象将抛给调用该方法的方法。