

《Java 程序设计》实验报告九

学生姓名：李季鸿

班级：2021 级计本（3）班

学号：20213002624

实验地点：线上

指导教师：张春元

实验日期：2023-05-08

共 2 学时

实验环境：Win10+JDK1.8+IntelliJ IDEA 2022.1.1

1. 实验目的

学习常用类和 GUI 编程，巩固面向对象编程思想。

2. 实验内容

- （1）用集成化开发工具完成实验教材 P72 实验 3 内容。
- （2）用集成化开发工具完成实验教材 P74 实验 4 内容。
- （3）用集成化开发工具完成实验教材 P81 实验 1 内容。
- （4）用集成化开发工具完成实验教材 P82 实验 2 内容。

3. 实验过程

报告撰写具体要求：截屏显示或直接写出实验 1 至实验 4 的源码和运行结果。

实验内容（1）：

```
package _8_.shiyant1;
/**
 * \* Created with IntelliJ IDEA.
 * \* @ProjectName: java_study_codes
 * \* @FileName: CompareDate
 * \* @author: li-jihong
 * \* Date: 2023-05-08 1:26
 */

import java.util.Calendar;
import java.util.Date;
import java.util.Scanner;

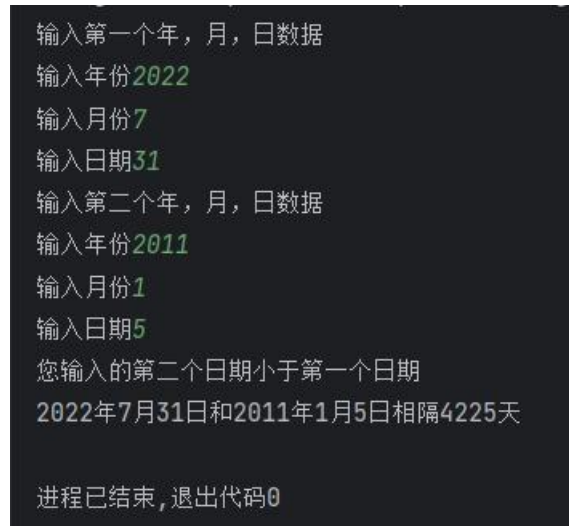
public class CompareDate {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("输入第一个年，月，日数据");
        // System.out.println("输入第一个年，月，日，时，分，秒数据");
        System.out.print("输入年份");
        short yearOne = scanner.nextShort();
        System.out.print("输入月份");
        byte monthOne = scanner.nextByte();
        System.out.print("输入日期");
        byte dayOne = scanner.nextByte();
```

```

//      System.out.print("输入时");
//      byte hourOne = scanner.nextByte();
//      System.out.print("输入分");
//      byte minuteOne = scanner.nextByte();
//      System.out.print("输入秒");
//      byte secondOne = scanner.nextByte();
System.out.println("输入第二个年，月，日数据");
//      System.out.println("输入第二个年，月，日，时，分，秒数据");
System.out.print("输入年份");
short yearTwo = scanner.nextShort();
System.out.print("输入月份");
byte monthTwo = scanner.nextByte();
System.out.print("输入日期");
byte dayTwo = scanner.nextByte();
//      System.out.print("输入时");
//      byte hourTwo = scanner.nextByte();
//      System.out.print("输入分");
//      byte minuteTwo = scanner.nextByte();
//      System.out.print("输入秒");
//      byte secondTwo = scanner.nextByte();
Calendar calendar = Calendar.getInstance(); //初始化日历对象
calendar.set(yearOne, monthOne - 1, dayOne); //将 calendar 的时间设置为 yearOne
年 monthOne 月 dayOne 日
//      calendar.set(yearOne, monthOne - 1, dayOne, hourOne, minuteOne, secondOne); //
将 calendar 的时间设置为 yearOne 年 monthOne 月 dayOne 日
long timeOne = calendar.getTimeInMillis(); //calendar 表示的时间转换成毫秒
calendar.set(yearTwo, monthTwo - 1, dayTwo);
//      calendar.set(yearTwo, monthTwo - 1, dayTwo, hourTwo, minuteTwo, secondTwo);
long timeTwo = calendar.getTimeInMillis();
Date date1 = new Date(timeOne); // 用 timeOne 做参数构造 date1
Date date2 = new Date(timeTwo);
if (date2.equals(date1))
    System.out.println("两个日期的年、月、日完全相同");
else if (date2.after(date1))
    System.out.println("您输入的第二个日期大于第一个日期");
else if (date2.before(date1))
    System.out.println("您输入的第二个日期小于第一个日期");
long days = Math.abs(timeTwo - timeOne) / (1000 * 60 * 60 * 24); // 使用
timeTwo,timeOne 计算两个日期相隔天数
System.out.println(yearOne + "年" + monthOne + "月" + dayOne + "日和"
    + yearTwo + "年" + monthTwo + "月" + dayTwo + "日相隔" + days + "天");
//      System.out.println(yearOne + "年" + monthOne + "月" + dayOne + "日" + hourOne +
"时" + minuteOne + "分" + secondOne + "秒和"

```

```
//          + yearTwo + "年" + monthTwo + "月" + dayTwo + "日" + hourTwo + "时" +
minuteTwo + "分" + secondTwo + "秒"
//          + "相隔" + days + "天");
    }
}
```



实验 1 运行截图

实验 1 后的练习:

(1)

```
package _8_.shiyuan1;
/**
 * \* Created with IntelliJ IDEA.
 * \* @ProjectName: java_study_codes
 * \* @FileName: CompareDate
 * \* @author: li-jihong
 * \* Date: 2023-05-08 1:26
 */

import java.util.Calendar;
import java.util.Date;
import java.util.Scanner;

public class CompareDate {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
//        System.out.println("输入第一个年, 月, 日数据");
        System.out.println("输入第一个年, 月, 日, 时, 分, 秒数据");
        System.out.print("输入年份");
        short yearOne = scanner.nextShort();
        System.out.print("输入月份");
        byte monthOne = scanner.nextByte();
```

```

System.out.print("输入日期");
byte dayOne = scanner.nextByte();
System.out.print("输入时");
byte hourOne = scanner.nextByte();
System.out.print("输入分");
byte minuteOne = scanner.nextByte();
System.out.print("输入秒");
byte secondOne = scanner.nextByte();
//    System.out.println("输入第二个年，月，日数据");
System.out.println("输入第二个年，月，日，时，分，秒数据");
System.out.print("输入年份");
short yearTwo = scanner.nextShort();
System.out.print("输入月份");
byte monthTwo = scanner.nextByte();
System.out.print("输入日期");
byte dayTwo = scanner.nextByte();
System.out.print("输入时");
byte hourTwo = scanner.nextByte();
System.out.print("输入分");
byte minuteTwo = scanner.nextByte();
System.out.print("输入秒");
byte secondTwo = scanner.nextByte();
Calendar calendar = Calendar.getInstance(); //初始化日历对象
//    calendar.set(yearOne, monthOne - 1, dayOne); //将 calendar 的时间设置为 yearOne
//    年 monthOne 月 dayOne 日
//    calendar.set(yearOne, monthOne - 1, dayOne, hourOne, minuteOne, secondOne); //将
//    calendar 的时间设置为 yearOne 年 monthOne 月 dayOne 日
//    long timeOne = calendar.getTimeInMillis(); //calendar 表示的时间转换成毫秒
//    calendar.set(yearTwo, monthTwo - 1, dayTwo);
//    calendar.set(yearTwo, monthTwo - 1, dayTwo, hourTwo, minuteTwo, secondTwo);
//    long timeTwo = calendar.getTimeInMillis();
//    Date date1 = new Date(timeOne); // 用 timeOne 做参数构造 date1
//    Date date2 = new Date(timeTwo);
//    if (date2.equals(date1))
//        System.out.println("两个日期的年、月、日完全相同");
//    else if (date2.after(date1))
//        System.out.println("您输入的第二个日期大于第一个日期");
//    else if (date2.before(date1))
//        System.out.println("您输入的第二个日期小于第一个日期");
//    long days = Math.abs(timeTwo - timeOne) / (1000 * 60 * 60 * 24); // 使用
//    timeTwo,timeOne 计算两个日期相隔天数
//    System.out.println(yearOne + "年" + monthOne + "月" + dayOne + "日和"
//    + yearTwo + "年" + monthTwo + "月" + dayTwo + "日相隔" + days + "天");

```

```

        System.out.println(yearOne + "年" + monthOne + "月" + dayOne + "日" + hourOne + "
时" + minuteOne + "分" + secondOne + "秒和"
            + yearTwo + "年" + monthTwo + "月" + dayTwo + "日" + hourTwo + "时" +
minuteTwo + "分" + secondTwo + "秒"
            + "相隔" + days + "天");
    }
}

```

```

输入第一个年，月，日，时，分，秒数据
输入年份2022
输入月份7
输入日期31
输入时12
输入分0
输入秒0
输入第二个年，月，日，时，分，秒数据
输入年份2011
输入月份1
输入日期5
输入时12
输入分0
输入秒0
您输入的第二个日期小于第一个日期
2022年7月31日12时0分0秒和2011年1月5日12时0分0秒相隔4225天
进程已结束,退出代码0

```

实验 1（1）运行截图

(2)

```

package _8_.shiyant1;

import java.text.DecimalFormat;
import java.util.Calendar;
import java.util.Scanner;

/**
 * \* Created with IntelliJ IDEA.
 * \* @ProjectName: java_study_codes
 * \* @FileName: InterestCalculator
 * \* @author: li-jihong
 * \* Date: 2023-05-08 1:38
 */

```

```

public class InterestCalculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("请输入存款金额: ");
        double amount = scanner.nextDouble();
        System.out.println("请输入起始时间: ");
        System.out.print("输入年份: ");
        short startYear = scanner.nextShort();
        System.out.print("输入月份: ");
        byte startMonth = scanner.nextByte();
        System.out.print("输入日期: ");
        byte startDay = scanner.nextByte();
        Calendar startDate = Calendar.getInstance();
        startDate.set(startYear, startMonth - 1, startDay);
        System.out.println("请输入结束时间: ");
        System.out.print("输入年份: ");
        short endYear = scanner.nextShort();
        System.out.print("输入月份: ");
        byte endMonth = scanner.nextByte();
        System.out.print("输入日期: ");
        byte endDay = scanner.nextByte();
        Calendar endDate = Calendar.getInstance();
        endDate.set(endYear, endMonth - 1, endDay);
        if (endDate.before(startDate)) {
            System.out.println("结束时间不能早于起始时间，请重新输入结束时间: ");
            System.out.print("输入年份: ");
            endYear = scanner.nextShort();
            System.out.print("输入月份: ");
            endMonth = scanner.nextByte();
            System.out.print("输入日期: ");
            endDay = scanner.nextByte();
            endDate.set(endYear, endMonth - 1, endDay);
        }
        long diffDays = (endDate.getTimeInMillis() - startDate.getTimeInMillis()) / (24 * 60 *
60 * 1000);
        double dailyRate = 0.01 / 365;
        double interest = amount * dailyRate * diffDays;
        DecimalFormat decimalFormat = new DecimalFormat("#.0000");
        System.out.println("存款金额: " + amount + " 元，存款天数: " + diffDays + " 天，
日利率: " + decimalFormat.format(dailyRate * 100) + "%，利息为: " +
decimalFormat.format(interest) + " 元");
    }
}

```

```
请输入存款金额: 10000
请输入起始时间:
输入年份: 2023
输入月份: 5
输入日期: 8
请输入结束时间:
输入年份: 2023
输入月份: 4
输入日期: 8
结束时间不能早于起始时间, 请重新输入结束时间:
输入年份: 2023
输入月份: 6
输入日期: 8
存款金额: 10000.0 元, 存款天数: 31 天, 日利率: .0027%, 利息为: 8.4932 元

进程已结束,退出代码0
```

实验 1 (2) 运行截图

在这个程序中, 我们首先使用 `Scanner` 类从标准输入读取存款金额和起止时间。然后, 我们使用 `Calendar` 类来计算起止时间之间的天数。最后, 我们使用利率计算公式计算出利息并输出结果。请注意, 此示例程序假设利率为固定值, 而实际上, 利率通常会随时间和市场条件而变化。此外, 此程序也没有考虑复利的情况。因此, 在实际应用中, 应该根据需要进行相应的调整。

根据测试结果, 可以发现程序存在两个问题:

输入结束时间时, 月份为 2, 日期为 2, 显然结束时间在开始时间之前, 但程序没有对这种情况进行判断和处理。

输出的利息结果只保留了小数点后两位, 但实际上银行计算利息的精度更高, 应该保留更多小数位。

针对上述问题进行了修正和优化, 改进后的程序在输入结束时间时增加了判断, 如果结束时间早于起始时间, 会提示用户重新输入结束时间。同时, 将计算利息时的利率改为每日万分之一, 这是银行常用的利率计算方式。在输出利息结果时, 将利息保留了小数点后四位, 提高了计算精度。

实验内容 (2):

```
package _8_.shiyang2;

/**
 * \* Created with IntelliJ IDEA.
 * \* @ProjectName: java_study_codes
 * \* @FileName: HandleBigInteger
 * \* @author: li-jihong
 * \* Date: 2023-05-08 1:49
 */
```

```

import java.math.BigInteger;

public class HandleBigInteger {
    public static void main(String[] args) {
        BigInteger n1 = new BigInteger("987654321987654321987654321"),
            n2 = new BigInteger("123456789123456789123456789"),
            result = null;

        result = n1.add(n2);
        System.out.println("和:" + result.toString());
        result = n1.subtract(n2);
        System.out.println("差:" + result.toString());
        result = n1.multiply(n2);
        System.out.println("积:" + result.toString());
        result = n1.divide(n2);
        System.out.println("商:" + result.toString());
        BigInteger m = new BigInteger("1968957"),
            COUNT = new BigInteger("0"),
            ONE = new BigInteger("1"),
            Two = new BigInteger("2");
        System.out.println(m.toString() + "因子有:");
        for (BigInteger i = Two; i.compareTo(m) < 0; i = i.add(ONE)) {
            if ((n1 remainder(i).compareTo(BigInteger.ZERO)) == 0) {
                COUNT = COUNT.add(ONE);
                System.out.print(" " + i.toString());
            }
        }
        System.out.println("");
        System.out.println(m.toString() + "一共有" + COUNT.toString() + "个因子");
    }
}

```

```

和:1111111111111111111111111111110
差:864197532864197532864197532
积:121932631356500531591068431581771069347203169112635269
商:8
1968957因子有:
 3 9 17 27 51 153 289 459 757 867 2271 2601 6813 7803 12869 20439 38607 115821 218773 347463 379721 656319 1139163
1968957一共有23个因子
进程已结束,退出代码0

```

实验 2 运行截图

实验 2 后的练习:

(1)

```
package _8_.shiyang2;
```

```
/**
```



```

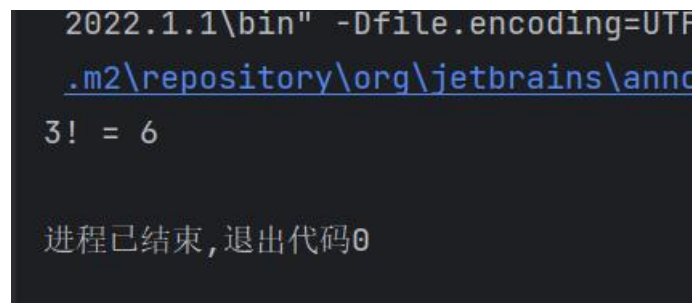
/** Created with IntelliJ IDEA.
 * \* @ProjectName: java_study_codes
 * \* @FileName: Factorial
 * \* @author: li-jihong
 * \* Date: 2023-05-08 1:56
 */

import java.math.BigInteger;

public class Factorial {
    public static BigInteger factorial(int n) {
        BigInteger result = BigInteger.ONE;
        for (int i = 1; i <= n; i++) {
            result = result.multiply(BigInteger.valueOf(i));
        }
        return result;
    }

    public static void main(String[] args) {
        int n = 3;
        BigInteger result = factorial(n);
        System.out.println(n + "! = " + result);
    }
}

```



实验 2（1）运行截图

在这个程序中，我们使用了 Java 中的 `BigInteger` 类来进行大整数的计算。在 `factorial` 方法中，我们使用一个 `BigInteger` 对象 `result` 来保存计算结果，初始值为 1。然后，我们用一个 `for` 循环，从 1 到 `n` 遍历每个数字，将 `result` 乘以当前数字，最后返回 `result` 即可得到 `n` 的阶乘。在 `main` 方法中，我们调用 `factorial` 方法来计算 3 的阶乘，并将结果输出到控制台上。

（2）

```

package _8_.shiyang2;

/**
 * \* Created with IntelliJ IDEA.
 * \* @ProjectName: java_study_codes

```

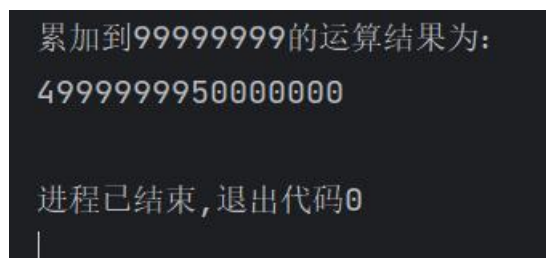
```

/** @FileName: sum
/** @author: li-jihong
/** Date: 2023-05-08 2:01
*/

import java.math.BigInteger;

public class Sum {
    public static void main(String args[]) {
        BigInteger n = new BigInteger("0");
        BigInteger x = new BigInteger("99999999");
        BigInteger ONE = new BigInteger("1");
        System.out.println("累加到 99999999 的运算结果为: ");
        for (BigInteger i = ONE; i.compareTo(x) <= 0; i = i.add(ONE)) {
            n = n.add(i);
        }
        System.out.println(n.toString());
    }
}

```



```

累加到99999999的运算结果为:
4999999950000000

进程已结束,退出代码0

```

实验 2（2）运行截图

实验内容（3）:

MainClass.java:

```

package _8_.shiyans3;

/**
/** Created with IntelliJ IDEA.
/** @ProjectName: java_study_codes
/** @FileName: MainClass
/** @author: li-jihong
/** Date: 2023-05-08 2:07
*/

public class MainClass {
    public static void main(String[] args) {
        ComputerFrame frame;
        frame = new ComputerFrame();
        frame.setTitle("算术测试");
    }
}

```

```

        frame.setBounds(100, 100, 650, 180);
    }
}

ComputerFrame.java:
package _8_.shiyang3;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * \* Created with IntelliJ IDEA.
 * \* @ProjectName: java_study_codes
 * \* @FileName: ComputerFrame
 * \* @author: li-jihong
 * \* Date: 2023-05-08 2:06
 */
public class ComputerFrame extends JFrame {
    JMenuBar menuBar;
    JMenu choiceGrade;
    JMenuItem grade1, grade2;
    JTextField textOne, textTwo, textResult;
    JButton getProblem, giveAnswer;
    JLabel operatorLabel, message;
    Teacher teacherZhang;
    ComputerFrame() {
        teacherZhang = new Teacher();
        teacherZhang.setMaxInteger(20);
        setLayout(new FlowLayout());
        menuBar = new JMenuBar();
        choiceGrade = new JMenu("选择级别");
        grade1 = new JMenuItem("幼儿级别");
        grade2 = new JMenuItem("儿童级别");
        grade1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                teacherZhang.setMaxInteger(10);
            }
        });
        grade2.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                teacherZhang.setMaxInteger(50);
            }
        });
    }
}

```

```

});
choiceGrade.add(grade1);
choiceGrade.add(grade2);
menuBar.add(choiceGrade);
setJMenuBar(menuBar);
textOne = new JTextField(5);
textTwo = new JTextField(5);
textResult = new JTextField(5);
operatorLabel = new JLabel("+");
operatorLabel.setFont(new Font("Arial", Font.BOLD, 20));
message = new JLabel("你还没有回答呢");
getProblem = new JButton("获取题目");
giveAnswer = new JButton("确认答案");
add(getProblem);
add(textOne);
add(operatorLabel);
add(textTwo);
add(new JLabel("="));
add(textResult);
add(giveAnswer);
add(message);
textResult.requestFocus();
textOne.setEditable(false);
textTwo.setEditable(false);
getProblem.setActionCommand("getProblem");
textResult.setActionCommand("answer");
giveAnswer.setActionCommand("answer");
teacherZhang.setJTextField(textOne, textTwo, textResult);
teacherZhang.setJLabel(operatorLabel, message);
getProblem.addActionListener(teacherZhang);
giveAnswer.addActionListener(teacherZhang);
textResult.addActionListener(teacherZhang);
setVisible(true);
validate();
setDefaultCloseOperation(DISPOSE_ON_CLOSE);
}
}

```

Teacher.java:

```

package _8_.shiyang3;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

```

```

import java.util.Random;

/**
 * \* Created with IntelliJ IDEA.
 * \* @ProjectName: java_study_codes
 * \* @FileName: Teacher
 * \* @author: li-jihong
 * \* Date: 2023-05-08 2:06
 */
public class Teacher implements ActionListener {
    int numberOne, numberTwo;
    String operator = "";
    boolean isRight;
    Random random;
    int maxInteger;
    JTextField textOne, textTwo, textResult;
    JLabel operatorLabel, message;

    Teacher() {
        random = new Random();
    }

    public void setMaxInteger(int n) {
        maxInteger = n;
    }

    public void actionPerformed(ActionEvent e) {
        String str = e.getActionCommand();
        if (str.equals("getProblem")) {
            numberOne = random.nextInt(maxInteger) + 1;
            numberTwo = random.nextInt(maxInteger) + 1;
            double d = Math.random();
            if (d >= 0.5)
                operator = "+";
            else
                operator = "-";
            textOne.setText("" + numberOne);
            textTwo.setText("" + numberTwo);
            operatorLabel.setText(operator);
            message.setText("请回答");
            textResult.setText(null);
        } else if (str.equals("answer")) {
            String answer = textResult.getText();

```

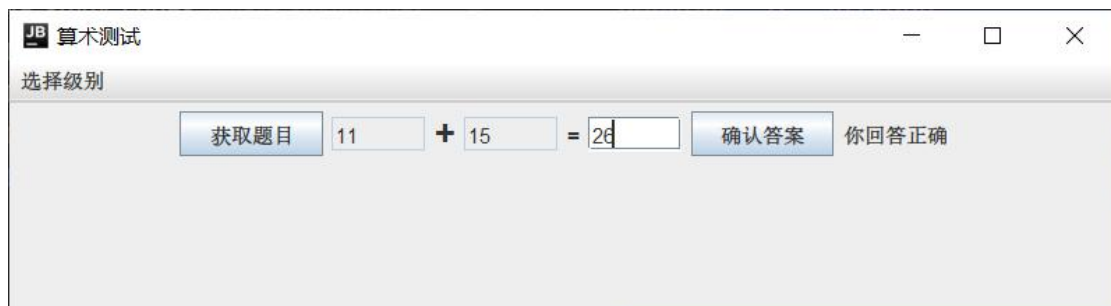
```

try {
    int result = Integer.parseInt(answer);
    if (operator.equals("+")) {
        if (result == numberOne + numberTwo)
            message.setText("你回答正确");
        else
            message.setText("你回答错误");
    } else if (operator.equals("-")) {
        if (result == numberOne - numberTwo)
            message.setText("你回答正确");
        else
            message.setText("你回答错误");
    }
} catch (NumberFormatException ex) {
    message.setText("请输入数字字符");
}
}

public void setJTextField(JTextField... t) {
    textOne = t[0];
    textTwo = t[1];
    textResult = t[2];
}

public void setJLabel(JLabel... label) {
    operatorLabel = label[0];
    message = label[1];
}
}

```



实验 3 运行截图

实验 3 后的练习：

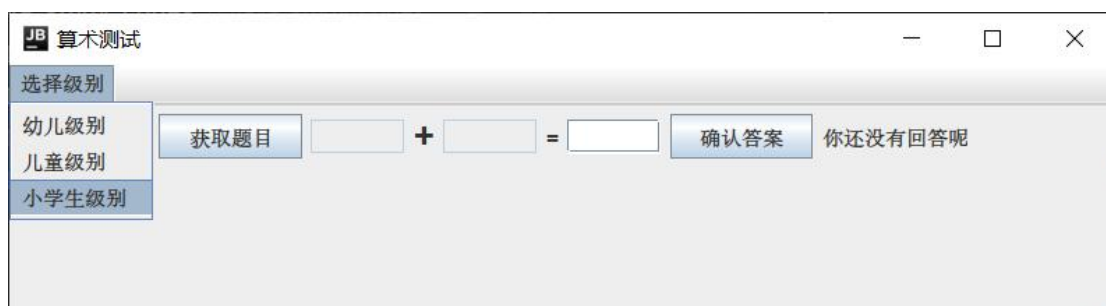
(1) 模仿本实验代码，再增加“小学生”级别。

“小学生”级别相关代码：

```

JMenuItem grade3;
grade3 = new JMenuItem("小学生级别");
grade3.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        teacherZhang.setMaxInteger(100);
    }
});
choiceGrade.add(grade3);

```



实验 3（1）运行截图

思路：在ComputerFrame类中创建JMenuItem的对象grade3，并将其作为“小学生级别”来呈现，而后设置其可以出现的最大数据为100，最后将其加入choiceGrade对象中。

（2）给上述程序增加测试乘法的功能。

需要修改代码如下：（蓝色字体为添加部分）

```

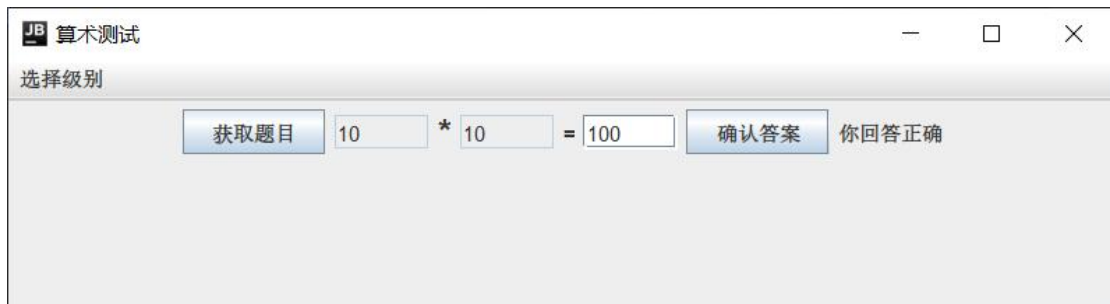
public void actionPerformed(ActionEvent e) {
    String str = e.getActionCommand();
    if(str.equals("getProblem")) {
        numberOne = random.nextInt(maxInteger)+1;//1至maxInteger之间
        numberTwo=random.nextInt(maxInteger)+1;
        double d=Math.random(); // 获取(0,1)之间的随机数
        if(d>=0.7)
            operator="+";
        else if(d<=0.3)
            operator="-";
        else
            operator="*";
        textOne.setText(""+numberOne);
        textTwo.setText(""+numberTwo);
        operatorLabel.setText(operator);
        message.setText("请回答");
        textResult.setText(null);
    }
    else if(str.equals("answer")) {
        String answer=textResult.getText();

```

```

try{    int result=Integer.parseInt(answer);
        if(operator.equals("+")){
            if(result==numberOne+numberTwo)
                message.setText("你回答正确");
            else
                message.setText("你回答错误");
        }
        else if(operator.equals("-")){
            if(result==numberOne-numberTwo)
                message.setText("你回答正确");
            else
                message.setText("你回答错误");
        }
        else if(operator.equals("*")){
            if(result==numberOne*numberTwo)
                message.setText("你回答正确");
            else
                message.setText("你回答错误");
        }
    }
    catch(NumberFormatException ex) {
        message.setText("请输入数字字符");
    }
}
}

```



实验 3（2）运行截图

经过上述测试，每项功能都比较健全了，但是发现，点击右上角的关闭按钮，窗口关闭后代码仍然不停止，故重写了 `windowClosing` 方法，当窗口关闭时，调用 `System.exit(0)` 来结束程序。然后，我们将这个实例添加到了窗口上，以监听窗口关闭事件。代码如下所示：

```
package _8_shiyan3;
```

```
/**
```

```
 * \* Created with IntelliJ IDEA.
```

```
 * \* @ProjectName: java_study_codes
```

```
 * \* @FileName: MainClass
```



```

    /** @author: li-jihong
    /** Date: 2023-05-08 2:07
    */
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFrame;

public class MainClass {
    public static void main(String[] args) {
        ComputerFrame frame = new ComputerFrame();
        frame.setTitle("算术测试");
        frame.setBounds(100, 100, 650, 180);

        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }
}

```



关闭窗口结束任务.
mp4

实验内容（4）：

GiveCalendar.java:

```
package _8_.shiyuan4;
```

```
import java.time.LocalDate;
```

```

/**
 * \* Created with IntelliJ IDEA.
 * \* @ProjectName: java_study_codes
 * \* @FileName: GiveCalendar
 * \* @author: li-jihong
 * \* Date: 2023-05-08 2:28
 */
public class GiveCalendar {
    /**
     * @param date 日期
     * @return dataArrays 这个月的所有天
     */
    public static LocalDate[] getCalendar(LocalDate date) {

```

```

        date = date.withDayOfMonth(1);
        //date 从 1 开始
        int days = date.lengthOfMonth();
        LocalDate[] dataArrays = new LocalDate[days];
        for (int i = 0; i < days; i++) {
            dataArrays[i] = date.plusDays(i);
        }
        return dataArrays;
    }
}

CalendarPanel.java:
package _8_.shiyang4;

import javax.swing.*;
import java.awt.*;
import java.time.DayOfWeek;
import java.time.LocalDate;

/**
 * \* Created with IntelliJ IDEA.
 * \* @ProjectName: java_study_codes
 * \* @FileName: CalendarPanel
 * \* @author: li-jihong
 * \* Date: 2023-05-08 2:28
 */
public class CalendarPanel extends JPanel {
    GiveCalendar calendar;
    public String[] name = {"日", "一", "二", "三", "四", "五", "六"};
    public LocalDate currentDate;
    LocalDate[] dateArrays;

    public CalendarPanel() {
        calendar = new GiveCalendar();
        currentDate = LocalDate.now();
        dateArrays = GiveCalendar.getCalendar(currentDate);
        showCalendar(dateArrays);
    }

    public void showCalendar(LocalDate[] dataArrays) {
        removeAll();
        GridLayout gridLayout = new GridLayout(7, 7);
        setLayout(gridLayout);
        JLabel[] titleWeek = new JLabel[7];

```

```

JLabel[] showDay = new JLabel[42];
for (int i = 0; i < 7; i++) {
    titleWeek[i] = new JLabel(name[i], JLabel.CENTER);
    add(titleWeek[i]);
}
for (int i = 0; i < 42; i++) {
    showDay[i] = new JLabel("", JLabel.CENTER);
}
for (int k = 7, i = 0; k < 49; k++, i++) {
    add(showDay[i]);
}
int space = printSpace(dateArrays[0].getDayOfWeek());
for (int i = 0, j = space + i; i < dateArrays.length; i++, j++) {
    showDay[j].setText("" + dateArrays[i].getDayOfMonth());
}
repaint();
}

public void setNext() {
    currentDate = currentDate.plusMonths(1);
    dateArrays = GiveCalendar.getCalendar(currentDate);
    showCalendar(dateArrays);
}

public void setPrevious() {
    currentDate = currentDate.plusMonths(-1);
    dateArrays = GiveCalendar.getCalendar(currentDate);
    showCalendar(dateArrays);
}

public int printSpace(DayOfWeek x) {
    int n = 0;
    switch (x) {
        case SUNDAY:
            n = 0;
            break;
        case MONDAY:
            n = 1;
            break;
        case TUESDAY:
            n = 2;
            break;
        case WEDNESDAY:

```

```

        n = 3;
        break;
    case THURSDAY:
        n = 4;
        break;
    case FRIDAY:
        n = 5;
        break;
    case SATURDAY:
        n = 6;
        break;
    }
    return n;
}
}
ShowCalendar.java:
package _8_.shiyang4;

import javax.swing.*;
import java.awt.*;

/**
 * \* Created with IntelliJ IDEA.
 * \* @ProjectName: java_study_codes
 * \* @FileName: ShowCalendar
 * \* @author: li-jihong
 * \* Date: 2023-05-08 2:28
 */
public class ShowCalendar extends JFrame {
    CalendarPanel showCalendar;
    JButton nextMonth;
    JButton previousMonth;
    JLabel showYear, showMonth;

    public ShowCalendar() {
        showCalendar = new CalendarPanel();
        add(showCalendar);
        nextMonth = new JButton("下一个月");
        previousMonth = new JButton("上一个月");
        showYear = new JLabel();
        showMonth = new JLabel();
        JPanel pNorth = new JPanel();
        showYear.setText("" + showCalendar.currentDate.getYear() + "年");
    }
}

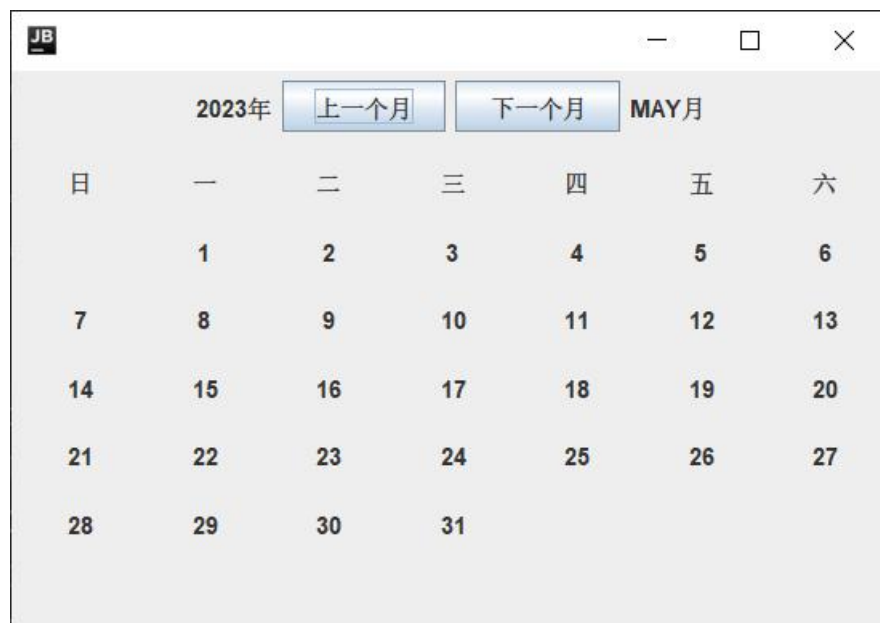
```

```

showMonth.setText("" + showCalendar.currentDate.getMonth() + "月");
pNorth.add(showYear);
pNorth.add(previousMonth);
pNorth.add(nextMonth);
pNorth.add(showMonth);
add(pNorth, java.awt.BorderLayout.NORTH);
nextMonth.addActionListener((e) -> {
    showCalendar.setNext();
    showYear.setText("" + showCalendar.currentDate.getYear() + "年");
    showMonth.setText("" + showCalendar.currentDate.getMonthValue() + "月");
});
previousMonth.addActionListener((e) -> {
    showCalendar.setPrevious();
    showYear.setText("" + showCalendar.currentDate.getYear() + "年");
    showMonth.setText("" + showCalendar.currentDate.getMonth() + "月");
});
setSize(290, 260);
setVisible(true);
setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
}

public static void main(String[] args) {
    new ShowCalendar();
}
}

```



实验 4 运行截图

实验 4 课后练习题:

请在ShowCalender窗口的SOUTH区域显示日历上的年份和月份。

源码:

```
package _8_.shiyan4;

import javax.swing.*;
import java.awt.*;

/**
 * \* Created with IntelliJ IDEA.
 * \* @ProjectName: java_study_codes
 * \* @FileName: ShowCalendar
 * \* @author: li-jihong
 * \* Date: 2023-05-08 2:28
 */
public class ShowCalendar extends JFrame {
    CalendarPanel showCalendar;
    JButton nextMonth;
    JButton previousMonth;
    JLabel showYear, showMonth;

    public ShowCalendar() {
        showCalendar = new CalendarPanel();
        add(showCalendar);
        nextMonth = new JButton("下一个月");
        previousMonth = new JButton("上一个月");
        showYear = new JLabel();
        showMonth = new JLabel();
        JPanel pNorth = new JPanel();
        showYear.setText(" " + showCalendar.currentDate.getYear() + "年");
        showMonth.setText(" " + showCalendar.currentDate.getMonth() + "月");
        pNorth.add(showYear);
        pNorth.add(previousMonth);
        pNorth.add(nextMonth);
        pNorth.add(showMonth);
        add(pNorth, java.awt.BorderLayout.NORTH);

        // 添加显示年份和月份的 JLabel 到窗口的南部
        JPanel pSouth = new JPanel();
        pSouth.add(showYear);
        pSouth.add(showMonth);
        add(pSouth, BorderLayout.SOUTH);

        nextMonth.addActionListener((e) -> {
            showCalendar.setNext();
        });
    }
}
```

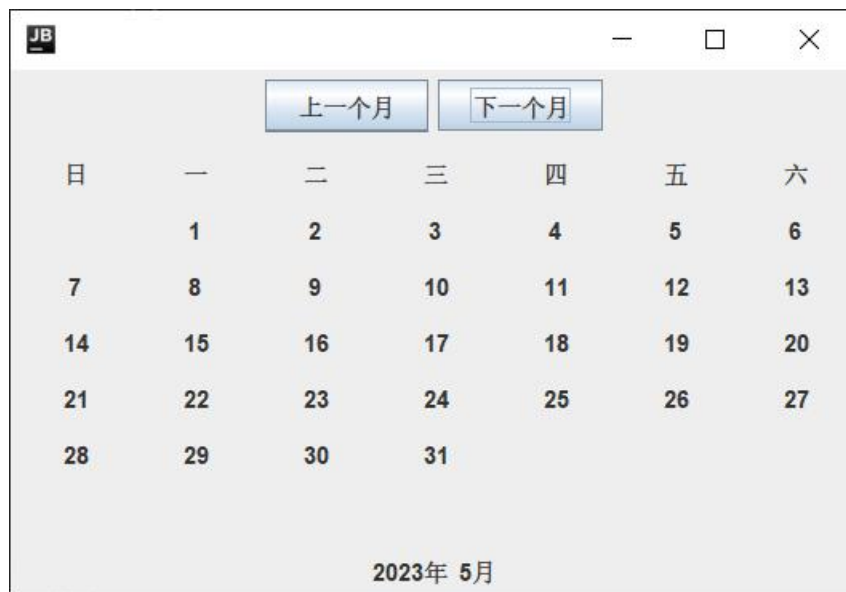
```

        showYear.setText("" + showCalendar.currentDate.getYear() + "年");
        showMonth.setText("" + showCalendar.currentDate.getMonthValue() + "月");
    });
    previousMonth.addActionListener((e) -> {
        showCalendar.setPrevious();
        showYear.setText("" + showCalendar.currentDate.getYear() + "年");
        showMonth.setText("" + showCalendar.currentDate.getMonth() + "月");
    });
    setSize(290, 300);
    setVisible(true);
    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
}

public static void main(String[] args) {
    new ShowCalendar();
}
}

```

结果显示：



实验 4 课后题运行截图

4. 实验总结

写出实验中的心得体会（对第 8 章理论课重点简述）。

常用类

常用类概述：

1. 内部类
2. Object 类

3. 包装类
4. 数学类
5. 时间类
6. 字符串
7. String Builder 和 StringBuffer
8. DecimalFormat

一、内部类

概念： 在一个类内部再定义一个完整的类。

一般情况下类与类之间是相互独立的，内部类的意思就是打破这种独立思想，让一个类成为另一个类的内部信息，和成员变量、成员方法同等级别。

内部类的好处：

把一个类写在外面和写在里面最终达到的结果都一样，那我们为什么还要使用内部类，岂不是多此一举吗？

采用内部类这种技术，可以隐藏细节和内部结构，封装性更好，让程序的结构更加合理！如果类很多且都暴露在外面，那么类与类之间的调用就会十分繁琐！

内部类的分类：

1.成员内部类（非静态内部类）

【参考代码】

```
package NeiBuLei;

public class OuterClass {
    //成员变量
    private String OuterName;
    //成员方法
    public void display(){
        System.out.println("这是外部类方法！");
        System.out.println(OuterName);
    }
    //内部类
    public class InnerClass{
        //成员变量
        private String InnerNme;
        //构造方法
```



```

    public InnerClass() {
        InnerNme = "Inner Class";
    }
    //成员方法
    public void display(){
        System.out.println("这是内部类方法！");
        System.out.println(InnerNme);
    }
}
// 主方法
public static void main(String[] args) {
    OuterClass outerClass = new OuterClass();
    outerClass.display();//这是外部类方法！ null

    // 这个类是内部类，已经不是独立的类了，因此不能像外部类一样直接创建！
    //InnerClass innerClass = new InnerClass(); 行不通
    OuterClass.InnerClass innerClass = outerClass.new InnerClass();// 同成员
方法/变量 只是加了个前缀
    innerClass.display();// 这是内部类方法！
}
}

```

输出结果：

这是外部类方法！

null

这是内部类方法！

Inner Class

总结：成员内部类（非静态内部类）的使用就是将内部类作为外部类的的一个成员变量/成员方法来使用，所以必须依赖于外部类的对象才能调用，用法和成员变量/成员方法一致！

2.局部内部类

局部内部类：基本的内部类还可以在一个方法体中定义。

```

package NeiBuLei;

public class OuterClass {
    //成员变量
    private String OuterName;
}

```

```

//成员方法
public void display(){
    class InnerClass {
        public void print(){
            System.out.println("这是一个局部内部类方法！");
        }
    }

    InnerClass innerClass = new InnerClass();
    innerClass.print();
}

// 主方法
public static void main(String[] args) {
    OuterClass outerClass = new OuterClass();
    outerClass.display();
}
}

```

3.静态内部类

静态内部类的构造不需要依赖于外部类对象,类中的静态组件都不需要依赖于任何对象,可以直接通过**类本身**进行构造。

```

package NeiBuLei;

public class OuterClass {
    //成员变量
    private String OuterName;

    //成员方法
    public void display(){
        System.out.println("这是外部类方法！");
        System.out.println(OuterName);
    }

    //静态内部类
    public static class InnerClass{
        private String InnerName;

        public InnerClass() {
            InnerName = "Inner Class";
        }
    }

    //成员方法
    public void display(){
        System.out.println("这是静态内部类方法！");
    }
}

```

```

        System.out.println(InnerName);
    }
}

// 主方法
public static void main(String[] args) {
    OuterClass outerClass = new OuterClass();
    outerClass.display();
    // 静态内部类的构造不依赖与外部类，可以直接通过类本身进行构造！
    InnerClass innerClass = new InnerClass();
    innerClass.display();
}
}

```

输出结果：

这是外部类方法！

null

这是静态内部类方法！

Inner Class

4.匿名内部类

匿名内部类：没有名字的内部类。

匿名内部类主要应用与接口的实现！

接口：

```

package NeiBuLei;

public interface MyInterface {
    public void test();
}

```

实现类：

```

package NeiBuLei;

public class MyImplement implements MyInterface{
    @Override
    public void test() {
        System.out.println("test");
    }
}

```

```
}
```

匿名内部类的使用：

```
package NeiBuLei;

public class Test {
    public static void main(String[] args) {

        //实现类
        MyInterface myInterface = new MyImplement();
        myInterface.test();

        //匿名内部类
        MyInterface myInterface1 = new MyInterface() { // 接口是不能 new 的，这里 new
w 的是接口的实现类（和实现类是一样的（同一个东西），没有实例而已）
            @Override
            public void test() {
                System.out.println("test");
            }
        };
        myInterface.test();
        /**
         * 最终两种实现方式的结果都是一样的！
         */
    }
}
```

匿名内部类的好处：

我们定义接口之后，它的实现类不需要去单独创建一个文件去写它的实现，我们可以把这个实现类的操作写到我们调用的地方就可以了！写起来更加简洁、方便。

匿名内部类的缺点：

耦合度太高了！

二、Object 类

jdk 中文在线文档：[Java 8 中文版](#) - [在线 API 中文手册](#) - [码工具 \(matools.com\)](#)

所有方法	接口方法	具体的方法
Modifier and Type		Method and Description
protected Object		<code>clone()</code> 创建并返回此对象的副本。
boolean		<code>equals(Object obj)</code> 指示一些其他对象是否等于此。
protected void		<code>finalize()</code> 当垃圾收集确定不再对该对象的引用时，垃圾收集器在对象上调用该对象。
类<?>		<code>getClass()</code> 返回此 <code>Object</code> 的运行时常量。
int		<code>hashCode()</code> 返回对象的哈希码值。
void		<code>notify()</code> 唤醒正在等待对象监视器的单个线程。
void		<code>notifyAll()</code> 唤醒正在等待对象监视器的所有线程。
String		<code>toString()</code> 返回对象的字符串表示形式。
void		<code>wait()</code> 导致当前线程等待，直到另一个线程调用该对象的 <code>notify()</code> 方法或 <code>notifyAll()</code> 方法。
void		<code>wait(long timeout)</code> 导致当前线程等待，直到另一个线程调用 <code>notify()</code> 方法或该对象的 <code>notifyAll()</code> 方法，或者指定的时间已过。
void		<code>wait(long timeout, int nanos)</code> 导致当前线程等待，直到另一个线程调用该对象的 <code>notify()</code> 方法或 <code>notifyAll()</code> 方法，或者某些其他线程中断当前线程，或一定量的实时时间。

Object 类常用方法:

1.equals 方法

== 与 equals 的对比【面试题】+ jdk 查看原码

== 是一个比较运算符

1. ==: 既可以判断基本类型，又可以判断引用类型
2. ==: 如果判断的是基本类型，判断的是值是否相等。

```

3.      //==: 如果判断的是基本类型，判断的是 值 是否相等
4.      int x1 = 10;
5.      int x2 = 10;
6.      double x3 = 10.0;
7.      System.out.println(x1 == x2); //true
8.      System.out.println(x1 == x3); //true

```

9. ==: 如果判断的是引用类型，判断的是地址是否相等，即判断是不是同一个对象

```

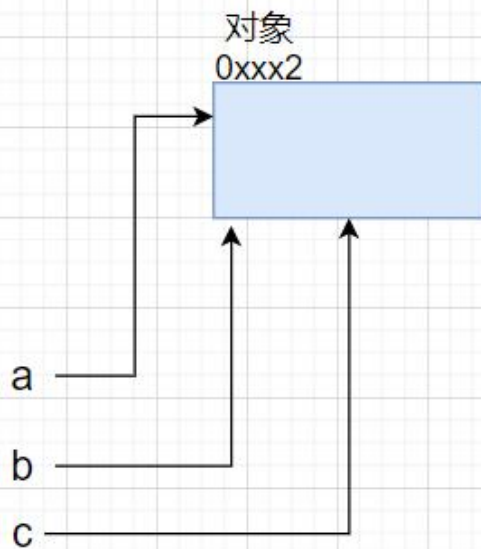
10. package Equals;
11.
12. public class Test01 {
13.     public static void main(String[] args) {
14.         //==: 如果判断的是引用类型，判断的是地址是否相等，即判断是不是同一个对象
15.         A a = new A();
16.         A b = a;

```

```

17.      A c = b;
18.      System.out.println(a==c); // ? true
19.      System.out.println(b==c); // true
20.
21.      B obj = a;
22.      System.out.println(obj==c); // true
23.
24.  }
25. }
26.
27. class B {}
28. class A extends B {}

```



都是指向同一个对象（地址）

4. equals 方法是 Object 类中的方法，只能判断引用类型。

idea 查看 Jdk 源码：鼠标光标放在要查看的方法上，直接输入 `ctrl + b`

查看某个类所有方法：`ctrl + F12`

5. 默认判断的是地址是否相等，子类(Object 类是所有类的父类)往往重写该方法，用于判断内容是否相等。

```

/*
Object 类 equals () 方法原码

//默认判断地址是否一样
public boolean equals(Object obj) {
    return (this == obj);
}

```

```
}
```

子类往往重写该方法，用于判断内容是否相等 String 类中的 equals() 方法原码（重写了父类 equals() 方法）

```
public boolean equals(Object anObject) {  
    if (this == anObject) { // 如果是同一个对象(地址相同)  
        return true; // 返回 true  
    }  
    if (anObject instanceof String) { // 判断类型  
        String anotherString = (String)anObject; // 向下转型  
        int n = value.length;  
        if (n == anotherString.value.length) { // 如果长度相同  
            char v1[] = value;  
            char v2[] = anotherString.value;  
            int i = 0;  
            while (n-- != 0) { // 比较每一个字符  
                if (v1[i] != v2[i])  
                    return false;  
                i++;  
            }  
            return true; // 如果两个字符串每一个字符都相同，则返回 true  
        }  
    }  
    return false;  
}  
*/
```

在看个例子

```
public boolean equals(Object obj) {  
    return (this == obj);  
}
```

Object中的equals()

```
public boolean equals(Object obj) {  
    if (obj instanceof Integer) {  
        return value == ((Integer)obj).intValue();  
    }  
    return false;  
}
```

Integer中的equals()

【小练习】

写出输出结果：

```
package Equals;

public class EqualsTest01 {
    public static void main(String[] args) {
        Person p1 = new Person();
        p1.name = "tom";
        Person p2 = new Person();
        p2.name = "tom";

        System.out.println(p1 == p2); // 引用类型——判断是否为同一个对象（地址）
        System.out.println(p1.name.equals(p2.name)); // p.name 是 String 类型，重写了 equals() 方法——判断内容是否一样
        System.out.println(p1.equals(p2)); // p1, p2 属于 Person 类，该类并没有重写 equals() 方法（继承父类 equals() 方法，即判断地址）

        String s1 = new String("abc");
        String s2 = new String("abc");

        System.out.println(s1.equals(s2));
        System.out.println(s1 == s2);

    }
}

class Person{
    public String name;
}
```

输出结果：

false

true

false

true

false

2.hashCode 方法

hashCode

```
public int hashCode()
```

返回对象的哈希码值。支持这种方法是为了散列表，如HashMap提供的那样。

hashCode的总合同是：

- 只要在执行Java应用程序时多次在同一个对象上调用该方法，hashCode方法必须始终返回相同的整数，前提是修改了对象中equals比较中的信息。该整数不需要从一个应用程序的执行到相同应用程序的另一个执行保持一致。
- 如果根据equals(Object)方法两个对象相等，则在两个对象中的每个对象上调用hashCode方法必须产生相同的整数结果。
- 不要求如果两个对象根据equals(java.lang.Object)方法不相等，那么在两个对象中的每个对象上调用hashCode方法必须产生不同的整数结果。但是，程序员应该意识到，为不等对象生成不同的整数结果可能会提高哈希表的性能。

尽可能多的合理实用，由类别Object定义的hashCode方法确实为不同对象返回不同的整数。（这通常通过将对象的内部地址转换为整数来实现，但Java的编程语言不需要此实现技术。）

结果

该对象的哈希码值。

小结：（可以当作地址来看但它本质上不是地址）

1. 提高具有哈希结构的容器的效率
2. 两个引用，如果指向的是同一个对象，则哈希值肯定一样
3. 两个引用，如果指向的是不同对象，则哈希值是不一样的
4. 哈希值主要根据地址号来！不能将哈希值完全等价于地址
5. 在后面的集合中 **hashCode** 如果需要的话，也会重写

```
package hashCode;

public class HashCode {
    public static void main(String[] args) {
        AA aa = new AA();
        AA aa2 = new AA();
        AA aa3 = aa;
        System.out.println("aa.hashCode()="+ aa.hashCode());
        System.out.println("aa2.hashCode()="+ aa2.hashCode());
        System.out.println("aa3.hashCode()="+ aa3.hashCode());
    }
}

class AA{ }
```

```
aa.hashCode()=460141958
aa2.hashCode()=1163157884
aa3.hashCode()=460141958
```

3.toString 方法

toString

```
public String toString()
```

返回对象的字符串表示形式。一般来说，toString方法返回一个“textually代表”这个对象的字符串。结果应该是一个简明扼要的表达式，容易让人阅读。建议所有子类覆盖此方法。

该toString类方法Object返回一个由其中的对象是一个实例，该符号字符`的类的名称的字符串@”和对象的哈希码的无符号的十六进制表示。换句话说，这个方法返回一个等于下列值的字符串：

```
getClass().getName() + '@' + Integer.toHexString(hashCode())
```

结果

对象的字符串表示形式。

基本介绍：

默认返回：全类名 + @ + 哈希值的十六进制

```
/*
Object toString() 原码
// (1) getClass().getName() 类的全类名（包名+类名）
// (2) Integer.toHexString(hashCode()) 将 hashCode 的值转成 16 进制字符串
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
*/
```

```
Monster monster = new Monster( name: "小旋风", job: "巡山的", salary: 1000.0);
//子类未重写toString()方法，即调用的是Object类的toString()方法
System.out.println(monster.toString()); //toStringDetail.Monster@1b6d3586
```

子类往往重写 toString 方法，用于返回对象的属性信息（**快捷键：alt + insert**），当然我们也可以自己定制。

```
//重写toString()方法，输出对象的属性
//快捷键：alt + insert
@Override
public String toString() {
    return "Monster{" +
        "name='" + name + '\'' +
        ", job='" + job + '\'' +
        ", salary=" + salary +
        "'}";
}
```

```
Monster monster = new Monster( name: "小旋风", job: "巡山的", salary: 1000.0);
//子类重写toString()方法后
System.out.println(monster.toString()); //Monster{name='小旋风', job='巡山的', salary=1000.0}
```

当我们输出一个对象时，toString()方法会被默认调用

```
//当我们输出一个对象时，toString()方法会被默认调用
System.out.println(monster); //Monster{name='小旋风', job='巡山的', salary=1000.0}
```

4. finalize 方法

finalize
<pre>protected void finalize() throws Throwable</pre>
<p>当垃圾收集确定不再对该对象的引用时，垃圾收集器在对象上调用该对象。一个子类覆盖了处理系统资源或执行其他清理的finalize方法。</p> <p>finalize的一般合同是，如果Java虚拟机已经确定不再有任何方法可以被任何尚未死亡的线程访问的方法被调用，除非是由于最后确定的其他对象或类的准备工作所采取的行动。finalize方法可以采取任何行动，包括使此对象再次可用于其他线程；然而，finalize的通常目的是在对象不可撤销地丢弃之前执行清除动作。例如，表示输入/输出连接的对象finalize方法可能会在对象被永久丢弃之前执行显式I/O事务来中断连接。</p> <p>所述finalize类的方法Object执行任何特殊操作；它只是返回正常。Object的Object可以覆盖此定义。</p> <p>Java编程语言不能保证哪个线程将为任何给定的对象调用finalize方法。但是，确保调用finalize的线程在调用finalize时不会持有任何用户可见的同步锁。如果finalize方法抛出未捕获的异常，则会忽略该异常，并终止该对象的定类。</p> <p>在为对象调用finalize方法之后，在Java虚拟机再次确定不再有任何方式可以通过任何尚未死亡的线程访问此对象的任何方法的情况下，将采取进一步的操作，包括可能的操作由准备完成的其他对象或类别，此时可以去掉对象。</p> <p>finalize方法从不被任何给定对象的Java虚拟机调用多次。</p> <p>finalize方法抛出的任何异常都会导致该对象的终止被停止，否则被忽略。</p>

finalize 方法：当垃圾收集确定不再对该对象的引用时，垃圾收集器在对象上调用该对象。

1. 当对象被回收时，系统自动调用该对象的 finalize 方法。子类可以重写该方法，做一些释放资源的操作
2. 什么时候被回收：当某个对象没有任何引用时，则 jvm 就认为这个对象是一个垃圾对象，就会时候垃圾回收机制来销毁该对象，在销毁该对象前，会先调用 finalize 方法。

```
class Car {
    private String name;
    //属性，资源。。
    public Car(String name) {
        this.name = name;
    }
    //重写finalize
    @Override
    protected void finalize() throws Throwable {
        System.out.println("我们销毁 汽车" + name );
        System.out.println("释放了某些资源...");
    }
}

Car bmw = new Car("宝马");
//这时 car对象就是一个垃圾,垃圾回收器就会回收(销毁)对象，在销毁对象前，会调用该对象的finalize方法
//,程序员就可以在 finalize中，写自己的业务逻辑代码(比如释放资源：数据库连接,或者打开文件..)
//,如果程序员不重写 finalize,那么就会调用 Object类的 finalize,即默认处理
//,如果程序员重写了finalize, 就可以实现自己的逻辑
bmw = null;
```

3. 垃圾回收机制的调用，是由系统来决定（即有自己的 GC 算法），也可以通过 System.gc()主动触发垃圾回收机制。

注：在实际开发中，几乎不会用 `finalize` 方法，更多的是为了应付面试

三、包装类

1.基本数据类型以及对应的包装类：

`byte` -> `Byte`

`short` -> `Short`

`int` -> `Integer`

`long` -> `Long`

`float` -> `Float`

`double` -> `Double`

`char` -> `Character`

`boolean` -> `Boolean`

这些类都在 `java.lang` 包

2.包装类的意义：

1. 让基本数据类型有面向对象的特征
2. 封装了字符串转化成基本数据类型的方法（重点）

3.包装类常用方法：

1. `Integer.parseInt()`
2. `Long.parseLong()`
3. `Double.parseDouble()`

【参考代码】

```
public class Test {  
    public static void main(String[] args) {
```

```

//      Integer i = new Integer(10); // 创建包装类对象
//      Integer ii = 10; // 自动打包
//      System.out.println(i+10); // 在使用上，int 和 Integer 其实没有区别，可以互相
使用
//      System.out.println(ii+10);
//      int j = ii; // 自动解包
//      System.out.println(j+100);

String a = "12";
String b = "34";
System.out.println(a+b); // 1234
// 转型：
// 字符串转成 int 的唯一方案
int c = Integer.parseInt(a);
int d = Integer.parseInt(b);
System.out.println(c+d); // 46

// 字符串转成 double 类型
String e = "1.25";
double f = Double.parseDouble(e);
System.out.println(f*6); // 7.5

// 转成 long 类型
long l = Long.parseLong("1234567");
System.out.println(l);
}
}

```

四、数学类

数学类的方法都是静态方法，可以直接引用——`Math.方法()`;

常用数学类方法：

1. `abs()`: 获取绝对值
2. `max()`: 求最大值
3. `min()`: 求最小值
4. `pow()`: 求次幂
5. `round()`: 四舍五入
6. `sqrt()`: 求平方根

五、时间类

Java 常用时间类:

1. Date 日期类
2. Calendar 日历类
3. SimpleDateFormat 格式化时间类

Date 和 Calendar 类 在 java.util 包中

SimpleDateFormat 类 在 java.text 包

1.Date 日期

【1】new Date() 可以获取到系统时间

【2】getTime() 能获取到时间的 long 形式，可以用来计算时间差

getTime()——获取计算机底层存储的数字，返回一个数字用来表示时间，这个数字的类型 long，单位为毫秒。

【参考代码】

```
import java.util.Date;

public class Test {
    public static void main(String[] args) {
        Date d = new Date();
        System.out.println(d); // 系统时间
        //get...()—获取年月日.....
        System.out.println(d.getYear()+1900); // 从1900年开始算的
        System.out.println(d.getMonth()+1); // 月份从0开始计算
        System.out.println(d.getDate()); // 天数
        System.out.println(d.getHours()); // 小时

        //getTime()—获取到时间的毫秒形式 返回的是 long
        System.out.println(d.getTime());
    }
}
```

2.Calendar 日历

【1】get() 获取到时间的某一部分

【2】set() 设置时间 --> 计算时间：系统已经帮我们设置好了，不用担心二月有多少天等问题，计算时间十分方便

注：Calendar 日历类是抽象类，因此不可以去 new 对象。虽然抽象类不能创建对象，但是 jdk 官方提供了一个实例对象的操作：

```
Calendar rightNow = Calendar.getInstance();
```

我们通过这条代码就是直接造了一个 Calendar 的对象

【参考代码】：get() 获取到时间的某一部分：

```
package date;

import java.util.Calendar;

public class TestCalendar {
    public static void main(String[] args) {
        Calendar cal = Calendar.getInstance();
        //      System.out.println(cal);

        /*
        假设当天：
        2021
        8
        10
        */

        cal.set(Calendar.DATE, cal.get(Calendar.DATE) + 31); // 计算时间
        (这里用天数计算的)

        // 获取 Calendar 创建的对象里的所有内容
        System.out.println(cal.get(Calendar.YEAR)); // 2021 年
        System.out.println(cal.get(Calendar.MONTH) + 1); // 月份：从 0 开始的 结果：
        为 10 月
        System.out.println(cal.get(Calendar.DATE)); // 日
        System.out.println(cal.get(Calendar.HOUR_OF_DAY)); // 小时
        System.out.println(cal.get(Calendar.MINUTE));
        System.out.println(cal.get(Calendar.SECOND));

    }
}
```

【参考代码】：set() 设置时间 --> 计算时间：

注: cal.setTime(d); 把 Date 转化成 Calendar

```
package date;

import java.util.Calendar;
import java.util.Date;

public class TestCalendar {
    public static void main(String[] args) {

        Date d = new Date();

        Calendar cal = Calendar.getInstance();

        cal.setTime(d); // 把 Date 转化成 Calendar

        System.out.println(cal);
        System.out.println(cal.get(Calendar.YEAR)); // 年
        System.out.println(cal.get(Calendar.MONTH)+1); // 月份: 从 0 开始的
        System.out.println(cal.get(Calendar.DATE)); // 日

    }
}
```

3.SimpleDateFormat 格式化时间

Date, Calendar 通过引用也可以进行时间的格式化, 但比较繁琐, 而

SimpleDateFormat 类是专门帮我们格式化时间的工具类, 它在 java.text 包中。

【时间格式】: yyyy-MM-dd HH:mm:ss

SimpleDateFormat 类有两大常用方法:

【1】format(Date):

format(Date) 帮我们把时间转成字符串, 字符串的格式为 SimpleDateFormat 类定义对象时设置的时间格式

【参考代码】

```
package Simple;
```



```

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.logging.SimpleFormatter;

public class Test {
    public static void main(String[] args) {
        Date d = new Date();
        System.out.println(d); //Thu Aug 12 08:40:08 CST 2021 不美观

        // 设置格式化时间的模式，我们常用 yyyy-MM-dd HH:mm:ss 这个模式
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");//
        时间格式
        String s = sdf.format(d); // 格式化时间
        System.out.println(s); // 2021-08-12 08:45:09
    }
}

```

【2】parse(String):

parse(String) 帮我们把字符串转化成时间

【参考代码】

```

package Simple;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Scanner;

public class Test2 {
    public static void main(String[] args) throws ParseException {
        Scanner sc = new Scanner(System.in);

        System.out.println("请输入一个时间(yyyy-MM-dd HH:mm:ss): ");

        String s = sc.nextLine();

        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

        Date d = sdf.parse(s); // 把字符串转成时间
    }
}

```

```

        System.out.println(d);

        /*
        请输入一个时间(yyyy-MM-dd HH:mm:ss):
        2021-08-12 12:25:21
        Thu Aug 12 12:25:21 CST 2021
        */
    }
}

```

注：由于用户输入的字符串不一定是我们要求的格式，可能是任何东西，想把它们转成时间是不可能的，你不可能把一个人转成时间 对吧，因此存在着很大的风险未处理(异常：`java.text.ParseException`)，为此我们需要处理异常。

4.计算时间差

计算思路：

1. 格式化时间
2. 先将字符串转化成 `long` 类型时间
3. 计算毫秒级别时间差，取绝对值
4. 毫秒级别时间差转成秒级别
5. 秒级别时间差转成分钟级别
6. 分钟级别时间差转化显示成 `xx` 小时 `xx` 分钟

【参考代码】

```

package Simple;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class TestDiff {
    public static void main(String[] args) throws ParseException {

        String s1 = "2021-08-12 12:00:00"; // 开始时间
        String s2 = "2021-08-12 14:35:00"; // 结束时间

        //格式化时间
        SimpleDateFormat sdf = new SimpleDateFormat("YYYY-MM-dd HH:mm:ss");

        //将字符串转成时间形式
    }
}

```

```

Date d1 = sdf.parse(s1);
Date d2 = sdf.parse(s2);

//计算时间差:先要获取时间毫秒形式(long 类型) 再做差
long long1 = d1.getTime();
long long2 = d2.getTime();
long diffTime = Math.abs(long1 - long2);

// 秒级别时间差
long diffSec = diffTime / 1000;

// 分级别时间差
long diffMin = diffSec / 60;

//显示 xx 小时 xx 分钟
long displayHours = diffMin / 60; // 小时
long displayMin = diffMin % 60; //分钟

System.out.println("您学习的时长为: "+displayHours+"小时"+displayMin+"分钟");
}
}

```

六、String 类

字符串类常用方法

方法汇总：

修饰符和返回值的类型	方法名	解释
char	charAt()	获取某个位置的字符
String	concat()	字符串的拼接。一般字符串拼接直接相加就好了
boolean	contains()	判断原字符串是否含有 xxx 字符串，常用于子串的判断
boolean	endsWith()	判断原字符串是否以 xxx 字符串结尾
boolean	startsWith()	判断原字符串是否以 xxx 字符串开头
boolean	equals()	判断两边字符串内容是否相同；==判断地址是否相同
boolean	equalsIgnoreCase()	忽略大小写判断两边字符串的内容是否一样
int	indexOf()	计算给出字符串第一个出现的位置

修饰符和返回值的类型	方法名	解释
int	LastindexOf()	计算给出字符串最后一个出现的位置
int	length()	计算字符串的长度
String	replace()	字符串内容的替换
String[]	split()	字符串切割，最终结果是一个字符串数组
String	substring()	字符串截取，左闭右开：[)
String	trim()	去掉字符串左右两边的空格，中间的不行
static String	valueOf()	官方：基本数据类型转为字符串操作；直接：变量 + ""

注：字符串是一个不可变的类型（**final** 类），几乎所有的字符串操作都会返回一个新字符串而不是在原有基础上进行修改。

【示例代码】

```
public class Test {
    public static void main(String[] args) {
        String s = "我的名字叫李华";

        s.concat("hhh"); // 在字符串 s 上拼接，拼接 hhh
        System.out.println(s); // 我的名字叫李华
        //字符串是不可变的数据类型
        //几乎所有的字符串操作都会返回一个新字符串
        String s1 = s.concat("hhh"); // 在字符串 s 上拼接，拼接 hhh
        System.out.println(s1); //我的名字叫李华 hhh

        String str1 = "李华喜欢看罗老师的视频";
        str1.replace("李华", "张三");
        System.out.println(str3); //李华喜欢看罗老师的视频 并没有替换 字符串是不变的 s
        tr1 还是 str1

        String str2 = str1.replace("李华", "张三"); //几乎所有的字符串操作都会返回一个
        新字符串 新串要用新变量接
        System.out.println(str2); //张三喜欢看罗老师的视频

    }
}
package String;

import java.util.Scanner;

public class Test {
```

```

public static void main(String[] args) {
    String s = "我的名字叫李华";
    System.out.println(s.charAt(0)); // 获取第 0 个位置的字符

    s.concat("hhh");
    System.out.println(s); // 我的名字叫李华
    //字符串是不可变的数据类型
    //几乎所有的字符串操作都会返回一个新字符串
    String s1 = s.concat("hhh"); // 在字符串 s 上拼接，拼接 hhh
    System.out.println(s1); //我的名字叫李华 hhh

    System.out.println(s.contains("李华")); //true
    System.out.println(s.contains("牛津")); //false

    System.out.println("邀请李华来参加英语沙龙活动".endsWith("活动")); //true 判断是否以 xxx 为结尾
    System.out.println("邀请李华来参加英语沙龙活动".startsWith("李华")); //false 判断是否以 xxx 开头

    // equals 字符串内容是否相同

    // 接受邀请参加活动的李华到现场后要输入验证码
    // String yanZhengMa = "AAkm";
    //
    // Scanner sc = new Scanner(System.in);
    //
    // System.out.println("请输入验证码 (" + yanZhengMa + ")");
    //
    // String userInput = sc.nextLine();
    //
    // if (yanZhengMa.equalsIgnoreCase("aakm")) { // 忽略大小写判断两边的内容是否一样
    //     System.out.println("欢迎参加英语沙龙活动!");
    // } else {
    //     System.out.println("您未受到邀请，请现场报名!");
    // }

    // String str = "李华玩得很开心! ";
    // System.out.println(str.indexOf("开心")); // 5 计算给出字符串第一个出现的位置

    String str2 = "李华成绩很好";
    System.out.println(str2.length()); // 6 计算字符串的长度

```

```

String str3 = "李华喜欢看罗老师的视频";
str3.replace("李华", "张三");
System.out.println(str3); //李华喜欢看罗老师的视频 并没有替换 字符串是不变的 s
tr3 还是 str3

String str4 = str3.replace("李华", "张三");//几乎所有的字符串操作都会返回一个
新字符串 新串要用新变量接
System.out.println(str4); //张三喜欢看罗老师的视频

String str5 = "哈哈_呵呵_嘻嘻_噢 no";
String[] ss = str5.split("_");//切割
System.out.println(ss[0]); //哈哈
System.out.println(ss[1]); //哈哈
System.out.println(ss[2]); //嘻嘻
System.out.println(ss[3]); //噢 no

String str6 = "今天天气不错";
System.out.println(str6.substring(2,4)); //天气 字符串截取 [ ) 左闭右开, 右边
取不到

String str7 ="    哈    哈    ";
System.out.println(str7.trim()); // 去掉左右两边的空格

int i = 10;
System.out.println(String.valueOf(i)); // 基本数据类型转为字符串
System.out.println(i+""); // 野路子

    }
}

```

七、String Builder 和 StringBuffer

String 类的缺点：

String 是一个不可变的数据类型，每每拼接都会产生一个新的字符串，那么内存迟早会被这些拼接的字符串塞满。

String 类和 StringBuilder 和 StringBuffer 类的区别：

StringBuilder 和 StringBuffer：可变的字符串，不产生新对象，比较省内存，当进行大量的字符串拼接时建议使用 StringBuffer 和 StringBuilder，但它们两个一些方法的实现几乎跟 String 一样。

StringBuffer 和 StringBuilder 类：

【相似点】

两者用法一模一样，可以认为是一个类

【区别】

1. StringBuffer 线程安全，StringBuilder 非线程安全。
2. StringBuilder 相比于 StringBuffer 有速度优势，多数情况下建议使用 **StringBuilder** 类，但当被要求线程安全时必须使用 **StringBuffer** 类

字符串拼接方法：append()方法

StringBuffer 和 StringBuilder 转成 String 类：

```
StringBuilder sb = new StringBuilder("猫喜欢吃鱼");
String s = sb.toString();
```

【参考代码】

```
package String;

public class TestStringBuilder {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder();// 一个空的字符串""

        StringBuilder sb2 = new StringBuilder("猫喜欢吃鱼");
        System.out.println(sb2); // 猫喜欢吃鱼

        sb2.append(", 狗也喜欢吃鱼");
        System.out.println(sb2); // 追加 猫喜欢吃鱼, 狗也喜欢吃鱼

        sb2.insert(1, "哈哈");
        System.out.println(sb2); //猫哈哈喜欢吃鱼, 狗也喜欢吃鱼

        // 上述的操作 huan'c

        // 把 StringBuilder 转化成 String
        String s = sb2.toString();
```

```

        System.out.println(s); //猫哈哈喜欢吃鱼，狗也喜欢吃鱼

        // 上述操作都可以将 StringBuilder 换成 StringBuffer，结果一样

    }
}

```

八、DecimalFormat

DecimalFormat: 对小数进行格式化，保留几位小数。与格式化时间联想记。

. 表示小数点

0 和# 表示数位，保留几位就几个 0 或者#

【参考代码】

```

import java.text.DecimalFormat;
import java.util.Scanner;

public class Test {
    public static void main(String[] args) {
        double d= 10/3.0;
        System.out.println(d); //3.3333333333333335

        // . 表示小数点
        // 0 和#表示数字

        // 保留两位小数                格式
        DecimalFormat df = new DecimalFormat(".00"); // 或者.##
        String s = df.format(d); // 把 d 转成上面设置的格式
        System.out.println(s); //3.33

    }
}

```