**信息科学与工程学院**

**2020－2021学年第一学期**

实 验 报 告

课程名称： Java编程技术

实验名称： 实验三

专 业 班 级 通信工程 二班

学 生 学 号 201800121050

学 生 姓 名 孟麟芝

实 验 时 间 2020年10月8日

实验报告

【实验目的】

继续熟练使用Java实现一些简单的算法

【实验要求】

1.用while循环语句，计算1--200之间的所有3的倍数之和。

2.利用switch语句实现判断某年的某个月份有几天的程序。

3.水仙花数是指其个位、十位和百位上三个数的立方之和等于这个数本身。通过循环，判断100-999之间所有的数字，符合水仙花数条件的数字。（例如：1^3 + 5^3 + 3^3 = 153）

4.已知 XYZ + YZZ = 532，其中X、Y和Z为数字，编程求出X、Y和Z的值。

5.编程实现“百钱买百鸡”问题。母鸡5分钱一只，公鸡三分钱一只，小鸡一分钱三只，现在有百钱欲买百鸡，有多少种买法？程序分别用三种方法来写，第一方法是程序中有三层的循环，这个效率最低，第二方法是程序采用两层循环实现，第三方法是程序采用一层循环实现。

6.一个整数的各位数字之和能被9整除，则该数也能被9整除。验证这个定理的正确性。

注：验证从0--2147483647的整数中是否存在不符合该定理的数，若不存在即验证通过，验证时间可能需要2-4分钟，请耐心等待，但如果时间太长，务必思考为什么

7.有一对兔子，从出生后第3个月起每个月都生一对兔子，小兔子长到第三个月后每个月又生一对兔子，假如兔子都不死，问2年后的兔子总数为多少？（尝试用递归编程实现）

## 【实验具体内容】

## 【第一个实验】

1.实验源代码

**package arithmetic.multiple;**

**public class MultipleOfThree {**

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

**int i = 0, sum = 0;**

**while (i <= 200) {**

**sum += i;**

**i += 3;**

**}**

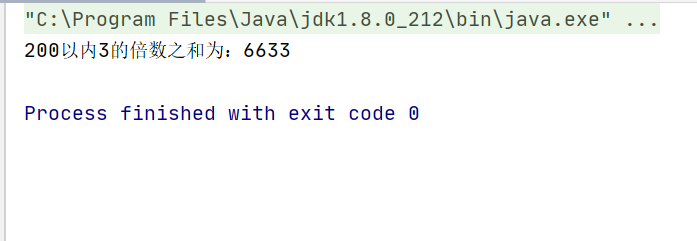
**System.out.println("200以内3的倍数之和为：" + sum);**

**}**

**}**

2.实验记录与说明分析

从0开始，计算0+3+6+9即可。



## 【第二个实验】

1.实验源代码

**package daysjudger;**

**import java.io.BufferedReader;**

**import java.io.InputStreamReader;**

**public class DaysJudger {**

**// 加throws Exception，意味着方法可能出现异常，但不必捕获它**

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

**BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));**

**System.out.println("请输入一个年份");**

**int days;**

**String strYear = reader.readLine();**

**System.out.println("请输入一个月份");**

**String strMonth = reader.readLine();**

**// parseInt方法是为了得到字符串代表的值**

**int year = Integer.parseInt(strYear);**

**int month = Integer.parseInt(strMonth);**

**boolean flag = judgeLeap(year);**

**switch (month) {**

**case 1:**

**case 3:**

**case 5:**

**case 7:**

**case 8:**

**case 10:**

**case 12:**

**days = 31;**

**break;**

**case 4:**

**case 6:**

**case 9:**

**case 11:**

**days = 30;**

**break;**

**case 2:**

**if (flag) {**

**days = 29;**

**break;**

**} else {**

**days = 28;**

**break;**

**}**

**default:**

**days = 0;**

**System.out.println("Error!");**

**break;**

**}**

**if (days != 0) {**

**System.out.println(year + "年" + month + "月份" + "的天数为" + days + "天");**

**}**

**}**

**public static boolean judgeLeap(int year) {**

**if (year % 400 == 0 || year % 4 == 0 && year % 100 != 0) {**

**return (true);**

**} else {**

**return (false);**

**}**

**}**

**}**

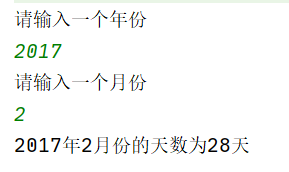
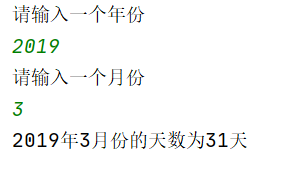
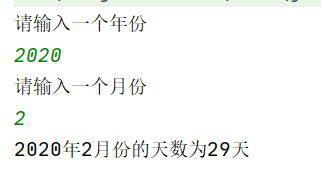
2.实验记录与说明分析

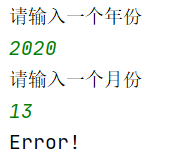
代码中使用了BufferedReader，可能会产生异常，但是在main方法后面加throws Exception即表示不必捕获方法中的异常。可以编译通过。

代码的核心部分在于switch case语句，1，3，5，7，8，12月均为31天，且不需要每个都break一下，4，6，9，11月均为30天，而2月需要考虑闰年，单独处理即可。default部分则是为了实现在用户输入非1~12月时进行报错。下面是单独截出switch部分的代码：

1. **switch** (month) {
2. **case** 1:
3. **case** 3:
4. **case** 5:
5. **case** 7:
6. **case** 8:
7. **case** 10:
8. **case** 12:
9. days = 31;
10. **break**;
11. **case** 4:
12. **case** 6:
13. **case** 9:
14. **case** 11:
15. days = 30;
16. **break**;
17. **case** 2:
18. **if** (flag) {
19. days = 29;
20. **break**;
21. } **else** {
22. days = 28;
23. **break**;
24. }
25. **default**:
26. days = 0;
27. System.out.println("Error!");
28. **break**;
29. }

下面是程序的运行结果：





可见程序输出了正确的结果，且有报错的机制。

## 【第三个实验】

1.实验源代码

**package arithmetic.narcissistic;**

**public class NarcissisticNumberFinder {**

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

**int units = 0, tens = 0, hundreds = 0;**

**for (int i = 100; i <= 999; i++) {**

**// 下面三行求解个位、十位、百位**

**units = i % 10;**

**tens = (i / 10) % 10;**

**hundreds = i / 100;**

**// Math.pow是计算乘方的方法**

**if ((Math.pow(units, 3) + Math.pow(tens, 3) + Math.pow(hundreds, 3)) == i) {**

**System.out.println(i + "是水仙花数");**

**}**

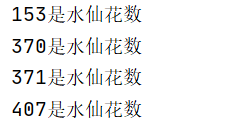
**}**

**}**

**}**

2.实验记录与说明分析

将程序运行，得到如下结果：



可见程序能够实现正确的功能

## 【第四个实验】

1.实验源代码

**package arithmetic.indefiniteequation;**

**public class IndefiniteEquation {**

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

**for (int x = 1; x <= 5; x++) {**

**for (int y = 1; y <= 5; y++) {**

**for (int z = 1; z <= 53; z++) {**

**if (100 \* x + 10 \* y + z + 100 \* y + 11 \* z == 532) {**

**System.out.println("X=" + x + " " + " Y=" + y + " " + " Z=" + z);**

**}**

**}**

**}**

**}**

**}**

**}**

2.实验记录与说明分析

这里的程序使用了最无脑的三重循环实现，其结果如下：



下面的百钱百鸡，则通过人工对算法的优化，可以实现循环数目的减少

## 【第五个实验】

1.实验源代码

**package arithmetic.indefiniteequation;**

**public class HundredChickens {**

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

**useTripleLoop();**

**useDoubleLoop();**

**useOneLoop();**

**}**

**public static void useTripleLoop() {**

**System.out.println("该方法使用了三重循环");**

**for (int roaster = 0; roaster <= 100; roaster++) {**

**for (int hen = 0; hen <= 100; hen++) {**

**for (int chick = 0; chick <= 100; chick++) {**

**if (3 \* roaster + 5 \* hen + chick / 3 == 100 && roaster + hen + chick == 100 && chick % 3 == 0) {**

**System.out.println("公鸡数=" + roaster + " " + " 母鸡数=" + hen + " " + " 小鸡数=" + chick);**

**}**

**}**

**}**

**}**

**}**

**public static void useDoubleLoop() {**

**System.out.println("该方法使用了二重循环");**

**// 将该三元不定方程组中消去chick，得4\*roaster+7\*hen=100，则roaster必小于等于25，hen必小于等于14**

**for (int roaster = 0; roaster <= 25; roaster++) {**

**for (int hen = 0; hen <= 14; hen++) {**

**if (4 \* roaster + 7 \* hen == 100) {**

**int chick = 100 - roaster - hen;**

**System.out.println("公鸡数=" + roaster + " " + " 母鸡数=" + hen + " " + " 小鸡数=" + chick);**

**}**

**}**

**}**

**}**

**public static void useOneLoop() {**

**System.out.println("该方法使用了一重循环");**

**// 将该三元不定方程组中消去chick，得4\*roaster+7\*hen=100，则hen=(100-4\*roaster)/7，其须为整数**

**for (int roaster = 0; roaster <= 25; roaster++) {**

**if ((100 - 4 \* roaster) % 7 == 0) {**

**int hen = (100 - 4 \* roaster) / 7;**

**int chick = 100 - roaster - hen;**

**System.out.println("公鸡数=" + roaster + " " + " 母鸡数=" + hen + " " + " 小鸡数=" + chick);**

**}**

**}**

**}**

**}**

2.实验记录与说明分析

三重循环对于编程而言是最简单的实现方式，对公鸡数、母鸡数、小鸡数分别在其取值范围内进行穷举即可。

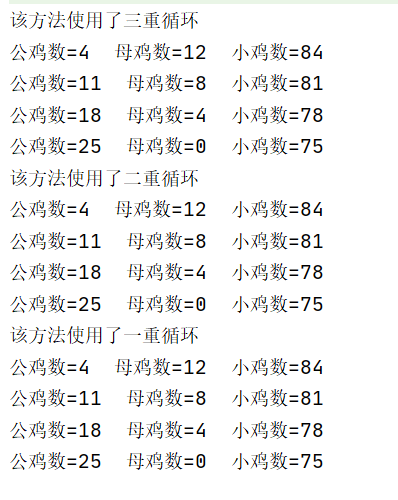
实际上，若设公鸡数为x，母鸡数为y，小鸡数为z，对于三元不定方程：

将两方程中第一个方程消去z，得到：

则这时只需要对x，y进行遍历，z恒等于100-x-y。

二重循环的方法是只对x，y进行遍历，那么一重循环则只需对x进行遍历，求出能让y为整数的x值即可。

下面是运行结果，可见三种方法的结果是一模一样的。



## 【第六个实验】

1.实验源代码

**package arithmetic.multiple;**

**public class MultipleOfNine {**

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

**double sum = 0;**

**int count = 10;**

**for (int i = 0; i < count; i++) {**

**sum += judgeMultiple();**

**}**

**double averageRunTime = sum / count;**

**System.out.println("程序运行 " + count + " 次的平均时间为：" + averageRunTime + "ms");**

**}**

**public static double judgeMultiple() {**

**int one, two, three, four, five, six, seven, eight, nine, ten, num = 0, count = 0;**

**double startTime = System.currentTimeMillis();**

**label2:**

**for (ten = 0; ten <= 2; ten++) {**

**for (nine = 0; nine <= 9; nine++) {**

**for (eight = 0; eight <= 9; eight++) {**

**for (seven = 0; seven <= 9; seven++) {**

**for (six = 0; six <= 9; six++) {**

**for (five = 0; five <= 9; five++) {**

**for (four = 0; four <= 9; four++) {**

**for (three = 0; three <= 9; three++) {**

**for (two = 0; two <= 9; two++) {**

**label1:**

**for (one = count; one <= 9; one++) {**

**// 筛查从0开始，0满足题目条件，则通过switch语句递推地寻找下一个各数位和为9的倍数的数**

**num = one + 10 \* two + 100 \* three + 1000 \* four + 10000 \* five**

**+ 100000 \* six + 1000000 \* seven + 10000000 \* eight**

**+ 100000000 \* nine + 1000000000 \* ten;**

**switch (one) {**

**// 若个位是0，则下一个数位和为9的倍数的数必为个位+9**

**// 若个位不是0，则下一个数位和为9的倍数的数必为个位-1，十位+1**

**case 0:**

**count = 9;**

**if (num % 9 == 0) {**

**// System.out.println(num);**

**// 将two减1（溢出也无妨）是为了抵消break label1后two的自增**

**two--;**

**break label1;**

**} else {**

**System.out.println("有数不满足");**

**break label2;**

**}**

**case 1:**

**case 2:**

**case 3:**

**case 4:**

**case 5:**

**case 6:**

**case 7:**

**case 8:**

**case 9:**

**// 个位-1**

**count = one - 1;**

**if (num % 9 == 0) {**

**// 2147483646是范围内最后一个需要检验的数，到此则应退出**

**if (num == 2147483646) {**

**break label2;**

**}**

**// System.out.println(num);**

**break label1;**

**} else {**

**System.out.println("有数不满足");**

**break label2;**

**}**

**default:**

**break;**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**double endTime = System.currentTimeMillis();**

**System.out.println("已到达" + num + "，完成了一轮检验，用时：" + (endTime - startTime));**

**return (endTime - startTime);**

**}**

**}**

2.实验记录与说明分析

本实验的目的在于找到十重循环做法下最简的算法，既然要让算法达到最简，那么就不可以验证“多余”的数，即我们只需要验证那些各个数位和为9的倍数的数，如何找到这些数？我们可以使用递推的方法。

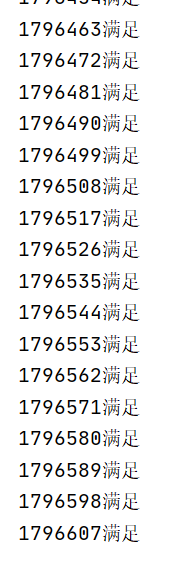
首先，0必然是满足这个条件的，0即是需要验证的第一个数，下一个离得最近的各个数位和为9的数为9，即将个位加9即可。那么再加9即可得到下下个各数位和为9的数。到这里可以发现，其实使用十重循环完全是多此一举，只需要使用一重循环，验证0，9，18，……是否满足题设条件即可，但这样显得没有意义。

回归到对十重循环做法的考虑，若一个数的个位为大于等于1的数，如81，离他最近的各数位和为9的数为90，这时，在十重循环的做法下显然不可以直接对个位加9，那么就可以采用另一种理解方式：十位减1，个位加1，即得到下一个离得最近的需要验证的数。

也就是说，能够做文章的部分就是个位和十位，这一思想通过下面的switch语句实现，其中label1在最内层循环外，是为了实现将十位+1，label2在最外层循环外，为了检验完成时跳出循环，count用于记录个位。

1. **switch** (one) {
2. // 若个位是0，则下一个数位和为9的倍数的数必为个位+9
3. // 若个位不是0，则下一个数位和为9的倍数的数必为个位-1，十位+1
4. **case** 0:
5. count = 9;
6. **if** (num % 9 == 0) {
7. // System.out.println(num);
8. // 将two减1（溢出也无妨）是为了抵消break label1后two的自增
9. two--;
10. **break** label1;
11. } **else** {
12. System.out.println("有数不满足");
13. **break** label2;
14. }
15. **case** 1:
16. **case** 2:
17. **case** 3:
18. **case** 4:
19. **case** 5:
20. **case** 6:
21. **case** 7:
22. **case** 8:
23. **case** 9:
24. // 个位-1
25. count = one - 1;
26. **if** (num % 9 == 0) {
27. // 2147483646是范围内最后一个需要检验的数，到此则应退出
28. **if** (num == 2147483646) {
29. **break** label2;
30. }
31. // System.out.println(num);
32. **break** label1;
33. } **else** {
34. System.out.println("有数不满足");
35. **break** label2;
36. }
37. **default**:
38. **break**;
39. }

先将程序验证过程中的数print出来，可见验证是没有问题的



由于print太浪费时间，直接注释掉，得到如下结果：



可见程序连续运行时是有越跑越快的倾向的，这可能和CPU的优化有关系。

## 【第七个实验】

1.实验源代码

**package arithmetic.fibonacci;**

**public class RecursiveFibonacciCalculator {**

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

**int month = 24;**

**System.out.println("这对兔子出生后第" + month + "个月时兔子总数为：" + getFibonacci(month) + "对");**

**}**

**public static int getFibonacci(int month) {**

**// 这里给出起始值,月份是从3开始的，且认为月份为2时兔子数为1**

**if (month <= 2) {**

**return (1);**

**} else {**

**// F(N)=F(N-1)+F(N-2)**

**return (getFibonacci(month - 1) + getFibonacci(month - 2));**

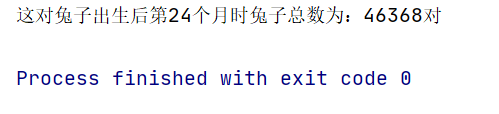
**}**

**}**

**}**

2.实验源代码

兔子个数问题可以简化为斐波那契数列问题，则可以使用递归求解，其实现也较为简单。



## 【修饰符的总结】

修饰符可以分为两大类：访问权限修饰符和非访问权限修饰符。下面先从大体分类上进行叙述。

**Ⅰ.1 访问权限修饰符**

在Java里面一共包含4种访问权限修饰符，分别是：private，friendly(default)，protected，public。

它们的权限如下所示：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **访问级别** | **访问控制修饰符** | **同类** | **同包** | **子类(不同包)** | **不同包(其他类)** |
| 公共 | public | 允许 | 允许 | 允许 | 允许 |
| 受保护 | protected | 允许 | 允许 | 允许 | 不允许 |
| 默认 | 没有修饰符 | 允许 | 允许 | 不允许 | 不允许 |
| 私有 | private | 允许 | 不允许 | 不允许 | 不允许 |

类只可以被public修饰，内部类、方法、变量则可以被所有访问权限修饰符修饰。

若是对父类的方法进行重写，则其权限不可以比父类的原方法更窄，只能相同或更宽（如原方法被public修饰，重写后的方法也必须用public，不加修饰符也不行）。

**Ⅰ.2 非访问权限修饰符**

1. static

用来创建类方法和类变量。

2. final

用来修饰类、方法和变量，final 修饰的类不能够被继承，修饰的方法不能被继承类重新定义，修饰的变量为常量，是不可修改的。

3. abstract

用来创建抽象类和抽象方法。

4. synchronized

用于多线程的同步。

5. volatile

修饰的成员变量在每次被线程访问时，都强制从共享内存中重新读取该成员变量的值。而且，当成员变量发生变化时，会强制线程将变化值回写到共享内存。这样在任何时刻，两个不同的线程总是看到某个成员变量的同一个值。

6. transient

序列化的对象包含被 transient 修饰的实例变量时，java 虚拟机(JVM)跳过该特定的变量。

下面将按被修饰的对象不同对修饰符进行分类讲解。

**Ⅱ.1 类修饰符**

1.访问修饰符

（1）公共类修饰符public

严格意义上讲，若不把无修饰符看作一种修饰符，那么Java 语言中类的访问控制符只有public，即公共的。

每个 Java 文件有且只有一个类是 public，它被称为主类，当然，若一个文件中只有一个类，完全可以不加public修饰（这时称之为friendly或default类），编译器也可找到它。其他外部类无访问控制修饰符，具有包访问性，即只允许在一个包内被访问，不允许在包外访问。

注意：一个类的内部类可以被其他访问控制修饰符protected、default、private、static修饰，这时它们相当于类的成员，被访问的规则与其他成员一致，不必想的过于复杂。

（2）类缺省访问控制符 default

如果一个类没有访问控制符，说明它具有缺省的访问控制符特性，也可称之为friendly类。此时，这个类只能被同一个包中的类访问或引用。这一访问特性又称为包访问性。

2.非访问控制符

（1）抽象类修饰符 abstract

用 abstract 修饰符修饰的类，被称为抽象类。

（2）最终类修饰符 final

当一个类不能被继承时可用修饰符 final修饰为最终类。被定义为 final 的类通常是一些有固定作用、用来完成某种标准功能的类。

**Ⅱ.2 方法修饰符**

1.访问控制修饰符

总的来讲分为公共访问控制符public、保护访问控制符protected、缺省默认default、私有访问控制符private

2.非访问控制修饰符

总的来讲分为抽象方法控制符abstract 、静态方法控制符static 、最终方法控制符final 、本地方法控制符native 、同步方法控制符synchronized

（1）抽象方法控制符 abstract

 abstract修饰的方法称为抽象方法。抽象方法仅有方法头，没有方法体和具体实现。

（2）静态方法控制符 static

用修饰符static修饰的方法称为静态方法。

静态方法是属于类的方法，存放在codesegment中，而不使用static修饰、限定的方法是属于某个具体对象（obj）的方法，存放在堆中。

由于static方法是属于整个类的，所以它不能操纵和处理属于某个对象的成员变量，而只能处理属于整个类的成员变量，即static方法只能处理static的域。

调用static方法时可以用类进行调用。即Class.method。讲类实例化以后还可以用实例化对象进行调用，即obj.method。

调用非静态的方法时则只能使用实例化对象进行调用，即obj.method。

（3）最终方法控制符 final

用修饰符final修饰的方法称为最终方法。最终方法是功能和内部语句不能更改的方法，即最终方法不能重写覆盖。final固定了方法所具有的功能和操作，防止当前类的子类对父类关键方法的错误定义，保证了程序的安全性和正确性。所有被private修饰符限定为私有的方法，以及所有包含在final类(最终类)中的方法，都被认为是最终方法。

（4）本地方法控制符 native

用修饰符 native 修饰的方法称为本地方法。为了提高程序的运行速度，需要用其它的高级语言书写程序的方法体，那么该方法可定义为本地方法用修饰符 native 来修饰。

（5）同步方法控制符 synchronized

该修饰符主要用于多线程程序中的协调和同步。

**Ⅱ.3 变量修饰符**

1.访问控制符

总体上可分为公共访问控制符public 、保护访问控制符protected 、缺省默认访问控制符default、私有访问控制符private。

（1）公共访问控制符 public

用public修饰的域称为公共域。由于public修饰符会降低运行的安全性和数据的封装性，所以一般应减少public域的使用。

（2）私有访问控制符 private

private修饰的成员变量只能被该类自身所访问，不能被其它任何类（包括子类）访问。

（3）保护访问控制符 protected

用protected修饰的成员变量可以被三种类所引用：

①该类自身；

②同一个包中的其它类；

③其它包中的子类。

使用修饰符protected的主要作用是允许其它包中的子类来访问父类的特定属性。

（4）缺省默认修饰符

没有访问控制符或者是用default修饰的成员变量可以被该类本身或同一包中的其他类访问。

2.非访问控制符

总体上分为静态域修饰符static、最终域修饰符final、易失（共享）域修饰符volatile、暂时性域修饰符transient。

（1）静态域修饰符 static

用static修饰的成员变量仅属于类的变量，而不属于任何一个具体的对象，静态成员变量的值是保存在类的内存区域的公共存储单元，而不是保存在某一个对象的内存区间。该类的任一对象访问它时取到的都是相同的数据；该类的任一对象修改它时,也都是对同一个内存单元进行操作。

（2）最终域修饰符 final

最终域修饰符final是用来定义常量的。一个类的域(成员变量)如果被修饰符final说明，则它的取值在程序的整个执行过程中都是不变的。

【实验心得】

经过一段时间的学习，发现面向对象的编程确实更易于“入门”，一些地方也更易于理解。处处皆对象，它们具有一些相应的的属性。例如在python中，我们该如何得到一个复数的实部呢？不需要多考虑，直接number.real即可，这就很“面向对象”。封装、继承、多态也为编程提供了许多便利，这在C语言中是不可想象的。从Java、python、C#等的编写回到C的编写，便不免感到局促。