Java语言

案　例　讲　解

针对每一讲的内容提供实际应用案例，培养解决问题能力。

0.5-1课时，每讲若干个实用案例。

案 例 1

设计一个简单的购物方案小程序：现有商品如下，书单价6元，糖单价1元，铅笔单价0.5元，颜料单价2元，纸单价3元，有20元，必须买一本书，其余用完。提供所有可行购买方案并在屏幕输出。

□ **要求**：书单价*1+其余商品单价*件数=20。

□ **信息分析：**

- 20-书单价*1=要求的搭配
- → 经典的穷举搜索
- → for语句
- → 4种商品件数嵌套for循环

□ **信息分析：**

- →“总额必须=20”即“余额要用完=0”
- →if语句判断余额
- → 为0输出搭配，否则继续循环

案例实现

```
// 定义价格
int bookPrice = 6; // 书的价格
int sugarPrice = 1; // 糖的价格
double pencilPrice = 0.5; // 铅笔的价格
int colorPrice = 2; // 颜料的价格
int paperPrice = 3; // 纸的价格

// 已经购买了一本书，剩余的钱
int remainingMoney = 20 - bookPrice;
```

首先计算出购买书之后剩余的金额；
然后，使用嵌套的 for 循环来尝试所有可能的糖、铅笔、颜料和纸的购买数量；
if 语句用于检查当前的组合是否用完了剩余的金额。如果是，就将这种组合打印出来。

```
// 使用嵌套循环穷举所有可能的购买方案
for (int sugar = 0; sugar <= remainingMoney / sugarPrice; sugar++) {
    for (int pencil = 0; pencil <= (remainingMoney - sugar * sugarPrice) / pencilPrice; pencil++) {
        for (int color = 0; color <= (remainingMoney - sugar * sugarPrice - pencil * pencilPrice) / colorPrice; color++) {
            int paper = (remainingMoney - sugar * sugarPrice - pencil * pencilPrice - color * colorPrice) / paperPrice;

            // 检查是否用完了所有的钱
            if (sugar * sugarPrice + pencil * pencilPrice + color * colorPrice + paper * paperPrice == remainingMoney) {
                System.out.println("糖: " + sugar + ", 铅笔: " + pencil + ", 颜料: " + color + ", 纸: " + paper);
            }
        }
    }
}
```

案例1衍生

顾客现有3.2元的零钱兑换需求，并要求营业员只能兑换成1元、5角和1角的硬币，且零钱的数量越少越好。

口代码设计思路：

① 确定零钱面额：首先确定可用的零钱面额，如本题的1元、0.5元和0.1元。

② 从最大面额开始：按照从大到小的顺序使用零钱面额，可确保使用的零钱数量最少。

③ 计算零钱数量：对于每个面额，计算在不超过总金额的情况下，可以兑换的最大数量。

④ 更新总金额：每次兑换后，更新剩余需要兑换的金额。

⑤ 循环直到完成：重复上述过程，直到总金额减至零。

案例实现

```
兑换 3.2 元需要以下硬币:
3 枚 1.0 元硬币
1 枚 0.5 元硬币
2 枚 0.1 元硬币
```

```java
public class CoinChange {
    public static void main(String[] args) {
        double totalAmount = 3.2; // 需要兑换的总金额
        double[] coins = {1.0, 0.5, 0.1}; // 可用的硬币面额
        int[] counts = new int[coins.length]; // 记录每种硬币的数量

        // 从大到小遍历硬币面额
        for (int i = 0; i < coins.length; i++) {
            counts[i] = (int) (totalAmount / coins[i]); // 计算当前面额硬币的
数量
            totalAmount -= counts[i] * coins[i]; // 更新剩余金额
        }

        // 输出结果
        System.out.println("兑换 " + totalAmount + " 元需要以下硬币：");
        for (int i = 0; i < counts.length; i++) {
            if (counts[i] > 0) {
                System.out.println(counts[i] + " 枚 " + coins[i] + " 元硬
币");
            }
        }
    }
}
```

案例分析

口案例1与衍生的相同：

- 两者都是在解多元一次方程：$Y = \sum_{i=1}^{n} (a_i x_i)$
- 已知总额Y和分额值x、分额类n，求可能方案的分额数a_i。

口 案例1与衍生的不同：

- 案例1是穷举每一种可能的方案，分额的数值和类数对方案无影响，设计for循环的层数与顺序时可随意；
- 衍生是求$\sum_{i=1}^{n} (a_i)$ 最小的方案，因此for循环的顺序是有选择的，分额值最大的x优先，然后轮到分额第二，……最后分额最小。

案 例 2

“水仙花数”是指一个n位正整数，各位数字的n次幂之和等于其本身。
假如AB是水仙花数，则AB=A*A+B*B；
假如ABC是水仙花数，则ABC=A*A*A+B*B*B+C*C*C；
同理水仙花数ABCD=A*A*A*A+B*B*B*B+C*C*C*C+D*D*D*D。

□ 要求：找出所有3位数的水仙花数。

□ 信息分析：

- 水仙花数定义=已知
- 构建水仙花数等式
- → 位数值和位数
- → /取整，%取余

□ 信息分析：

- →三位数=100-~999
- 找出水仙花数→经典的穷举搜索
- → for语句

案例实现

```
// 自定义方法，计算一个数字的立方
public static int cube(int number) {
    return number * number * number;
}

public static void main(String[] args) {
    // 自定义方法，计算数字的立方
    int[] cubes = new int[10];
    for (int i = 0; i < 10; i++) {
        cubes[i] = cube(i); // 预先计算0-9的立方
    }

    // 寻找所有的三位水仙花数
    for (int num = 100; num < 1000; num++) {
        int sum = 0;
        int digits = 3; // 三位数的位数

        int originalNum = num;
        while (num > 0) {
            int digit = num % 10; // 获取当前最低位的数字
            sum += cubes[digit]; // 累加各个位数的立方
            num = num / 10; // 去掉当前最低位的数字
        }

        if (originalNum == sum) {
            System.out.println(originalNum + " 是一个三位水仙花数。");
        }
    }
}
```

```
153 是一个三位水仙花数。
370 是一个三位水仙花数。
371 是一个三位水仙花数。
407 是一个三位水仙花数。
```

首先定义了一个数组 cubes 来存储0到9每个数字的立方结果，以避免重复计算。
然后，遍历所有的三位数，使用自定义的 cube 方法来计算各位数字的立方和，并判断是否满足水仙花数的条件。

代码优化

```
public class DaffodilNumber {
    public static void main(String[] args) {
        // 寻找所有的三位水仙花数
        for (int i = 100; i < 1000; i++) {
            int originalNum = i;
            int sum = 0;
            int digits = (int)(Math.log10(i) + 1); // 计算数字位数

            while (i > 0) {
                int digit = i % 10; // 获取当前最低位的数字
                sum += Math.pow(digit, digits); // 累加各个位数的n次幂
                i = i / 10; // 去掉当前最低位的数字
            }

            if (originalNum == sum) {
                System.out.println(originalNum + " 是一个三位水仙花数。");
            }
        }
    }
}
```

遍历了所有的三位数（100-999），计算每个数的每一位数字的立方和，然后检查这个和是否与原数相等。如果相等，就打印出来表明它是一个水仙花数。这个案例不仅展示了循环和条件判断的使用，还演示了数学操作和字符串处理。

案 例 3

用户输入1-12的任意数字，输出对应数字的月份英文。

口 分析：1-12输入范围确定，根据输入得到确定的输出→条件控制语句

```
int monthNumber = 6; // 用户输入的月份数字

if (monthNumber == 1) {
    System.out.println("January");
} else if (monthNumber == 2) {
    System.out.println("February");
} else if (monthNumber == 3) {
    System.out.println("March");
} // ... 继续直到12
else {
    System.out.println("Invalid month");
}
```

if-else 缺点：当条件较多时，代码可能变得冗长；
性能可能略逊于 switch，因为每个 else if 都需要顺序检查。

```
int monthNumber = 6; // 用户输入的月份数字

switch (monthNumber) {
    case 1: System.out.println("January");    break;
    case 2: System.out.println("February");   break;
    case 3: System.out.println("March");      break;
    // ... 继续直到12
    default: System.out.println("Invalid month");
}
```

switch 优点：代码更简洁，易于扫描多个固定值的条件；
可能有更好的性能，因为编译器可以优化跳转。

案例分析

某航司在暑
根据

```
int age = 25; // 用户的年龄
boolean isVIP = true; // 用户是否是VIP
// 使用 if-else 语句进行复杂的条件判断
if (age >= 60 || (age >= 18 && isVIP)) {
    System.out.println("您有资格获得特别优惠。"); }
else { System.out.println("您没有资格获得特别优惠。"); }
```

□ 分析：条件1年龄≥18且为VIP或条件2年龄≥60→条件表达式较为复杂→if-else

```
int age = 25; // 用户的年龄
boolean isVIP = true; // 用户是否是VIP
// 使用 if-else 语句进行复杂的条件判断
if (age >= 60 || (age >= 18 && isVIP)) {
    System.out.println("您有资格获得特别优惠。"); }
else { System.out.println("您没有资格获得特别优惠。"); }
```

if-else 优点：
•可以处理复杂的条件判断，包括范围和逻辑运算。
•适合条件表达式不确定或可能变化的情况。

switch 语句不能直接处理复杂的布尔表达式，因此并不适用于这种场景；如果非要用 switch 实现，只能通过多个 case 来模拟逻辑，但这会非常笨拙和难以阅读。

```
// 这种方法实际上并不推荐，因为它违反了 `switch` 的使用原则
switch (true) {
    case age >= 60 || (age >= 18 && isMember):
        System.out.println("您有资格获得特别优惠。");
        break;
    default:
        System.out.println("您没有资格获得特别优惠。");
}
```

switch 缺点：
•只适用于固定数量的离散值，不支持范围或复杂的布尔表达式。
•如果添加或删除选项，可能需要修改多个地方。

PPT结束！

THANKS