

目录

预备知识：通过键盘输入和文件输入	1
0.1 题目	1
0.2 结果展示	1
实验 1. UPC 码	2
1.1 题目	2
1.2 题目分析	2
1.3 数据设计	4
1.4 结果展示	4
1.6 总结与收获	4
实验 2. 数字转英语	6
2.1 题目	6
2.2 题目分析	6
2.3 代码展示	6
2.4 结果展示	9
2.5 总结与收获	9
实验 3. 长度 n 的子序列最大乘积	10

预备知识：通过键盘输入和文件输入

0.1 题目

请自行敲代码练习 Scanner 类接收键盘输入数据的模式。并通过 Java API 的查找发现 Scanner 与输入有关的其他函数，并加以验证练习。

0.2 结果展示

预备题较简单，完全为了熟悉 Scanner 的语法服务，代码附于附录。

```
Hi! yuan, the sum of 12 and 24.56 is 36.56
Enter an integer: 203
Enter a floating point number: 8.27
Enter your name: YangHao
Hi! YangHao, the sum of 203 and 8.27 is 211.27
```

图 1: Problem0

实验 1. UPC 码

1.1 题目

UPC 码由 12 个数字构成，其中最右侧的数字是校验位，我们用 d_{12} 表示，那么 UPC 码经过下面的公式计算之后必须是 10 的倍数

$$(d_1 + d_3 + d_5 + d_7 + d_9 + d_{11}) + 3(d_2 + d_4 + d_6 + d_8 + d_{10} + d_{12}).$$

比如，0-48500-00102 的校验位就是 8，因为经过上面的公式计算之后，只有数字 8 才可以得到 10 的倍数值 50

$$(8 + 0 + 0 + 0 + 5 + 4) + 3(2 + 1 + 0 + 0 + 8 + 0) = 50.$$

编写一个程序，通过命令行参数方式输入 1 个 11 位的数字，输出一个完整的 12 位的 UPC 码。注意：

1. 11 位数字可以包含前导零；
2. 11 位数字的大小已经使得数字的表示范围超过了 `int` 类型能够表示的大小了；
3. 如果不要用户严格的输入 11 位的数字，那么就需要在真正执行计算之前对用户输入的参数合法性进行考虑，请问你能列举出有多少种不合法的输入出现呢？如果程序的健壮性足够好，那就必须能处理这些意外的输入。

1.2 题目分析

本题目共分为三部分：

1. **数据的输入**。本题中数据是通过命令行输入的 11 位数字，也就是以 `String` 的形式存储在 `args[0]` 中，Java 中 `String` 类的丰富 `method` 可以帮助我们吧 `String` 当作数组来处理。
2. **异常数据的检测**。本题中可能出现三种异常数据：输入参数个数错误，长度不为 11，字符串有非数字符，校验位结果错误。
 - 输入参数个数错误可以通过 `args.length` 判断
 - 长度可以通过 `args[0].length()` 判断
 - 字符串的非字符数组在扫描数组过程中通过 `Character.isDigit()` 判断
 - 校验位结果错误可以在计算完成后判断

3. 算法设计. 观察 UPC 码的计算方法不难发现, UPC 码校验位的计算即用 50 减去三倍的偶位数和一倍的奇位数。将本题输入数据看作数组 s , 则 $s[0] = d_2$, 所以数组索引的奇偶和 UPC 码的奇偶相同。
4. 代码说明. 依据前三条的思路构建代码如下

Listing 1: Problem1.java

```

1 public class Problem1 {
2     public static void main(String[] args) {
3         System.out.print("Input:");
4         for(int i = 0; i < args.length ;i++)
5             System.out.print(" " + args[i]);
6         System.out.println();
7         if(args.length != 1){
8             System.out.println("ERROR! Please input 1 argument instead
9                 of " + args.length + ".");
10            return;
11        }
12        if (args[0].length() != 11) {
13            System.out.println("ERROR! Please input a 11-bit number
14                instead of " + args[0].length() + " bits.");
15            return;
16        }
17        int result = 50;
18        char tmp;
19        for (int i = 0; i < 11; i++) {
20            if (Character.isDigit(tmp = args[0].charAt(i))) {
21                if (i % 2 == 0) result -= 3 * Integer.parseInt(String.
22                    valueOf(tmp));
23                else result -= Integer.parseInt(String.valueOf(tmp));
24            } else {
25                System.out.println("ERROR! The " + (i + 1) + "th input
26                    bit is '" + tmp + "', it's not a digit.");
27                return;
28            }
29        }
30        if (result > 9 || result < 0) {
31            System.out.println("ERROR! The UPC code is " + result + ",
32                it's invalid.");
33        }
34    }
35 }

```

实验 1. UPC 码

```
28         return;  
29     }  
30     System.out.print(result + args[0]);  
31 }  
32 }
```

1.3 数据设计

本体主要目的为计算和检验异常数据，所以提供两个成功数据和几个对应于异常数据原因的数据得到验证即可。为了便于对比输入输出，将每一次的输出打印出来具体数据如下

Input	Output	Input	Output
04850000102	804850000102	04850000103	504850000103
04850000100	d_0 error(14)	04850000109	d_0 error(-13)
1	length error	012345678900	length error
0123456a789	not num	01?23456789	not num
04850000102 1	argu num error		

1.4 结果展示

The screenshot displays the program's output for several test cases. It shows the input string, the resulting UPC code (if valid), and an error message (if invalid). The errors include invalid lengths, non-digit characters, and invalid checksums.

Input	Output
Input: 04850000102 804850000102	
Input: 04850000109 ERROR! The UPC code is -13, it's invalid.	
Input: 1 ERROR! Please input a 11-bit number instead of 1 bits.	
Input: 012345678900 ERROR! Please input a 11-bit number instead of 12 bits.	
Input: 0123456a789 ERROR! The 8th input bit is 'a', it's not a digit.	
Input: 01?23456789 ERROR! The 3th input bit is '?', it's not a digit.	
Input: 04850000102 1 ERROR! Please input 1 argument instead of 2.	

图 2: Output of Problem1

1.6 总结与收获

本题比较简单，总结收获如下：

- 对字符串的处理中可以使用丰富的 Java 库函数，让人大开眼界；
- 对程序健壮性的测试需要考虑很多可能，有多种违规输入，同时为了提高实用性也应该为不同的违规输入定制错误提醒；

实验 1. *UPC* 码

- 由于违规输入较多，测试时如果采用 Junit 测试效果更好，可惜我被 Junit5 的配置折腾得快崩溃了，希望以后能再找机会学习。

实验 2. 数字转英语

2.1 题目

编写一个程序，从命令行参数中读取一个范围为 $[-999999999, 999999999]$ 的整数，输出为这个整数转换成英语表示的等价形式。下面列出程序中可能要用到的所有数字表示的英文单词：negative, zero, one, two, three, four, five, six, seven, eight, nine, ten, eleven, twelve, thirteen, fourteen, fifteen, sixteen, seventeen, eighteen, nineteen, twenty, thirty, forty, fifty, sixty, seventy, eighty, ninety, hundred, thousand, million。

注意：在转换过程中，尽可能使用更大的数字单位。比如 1500 这个数字应该的表示为 one thousand five hundred，而不应该是 fifteen hundred。

举例：数字 123419 转换成的英语表示为：one hundred twenty three thousand four hundred nineteen。

2.2 题目分析

1. 数据输入. 注意到本题的输入范围在 Java 的 `int` 范围内，所以可以通过 `Integer.parseInt` 直接得到数字。
2. 数据处理.
 - 可以参照英语的用语习惯，每三位划分，由数据范围可知最多划分成高中低三部分，分别用 `million` 和 `thousand` 分开，每一部分的打印逻辑都相同，可以总结为一个 `void print_3digit(int a)` 函数。
 - 为了避免负数可能导致的取余等计算问题，判断输入为负数并打印 `negative` 后即可。如果输入为 0 直接打印 `zero` 结束运行即可。
 - 特殊：如果输入为 0 直接打印 `zero` 结束运行即可
- 3.

2.3 代码展示

代码和注释如下所示，核心函数为 `void print_3digit(int a)` (line26)，该函数将输入的三位数打印出来，这一过程又分为打印 `hundred` 和两位数打印 `void print_2digit(int a)` (line35)。

Listing 2: Problem2.java

实验 2. 数字转英语

```
1 import static java.lang.Math.abs;
2
3 public class Problem2 {
4     public static void main(String[] args) {
5         int in = Integer.parseInt(args[0]);
6         if (in == 0) {
7             return;
8         } else if (in < 0) {
9             System.out.print("negative ");
10            in = abs(in);
11        }
12        int low = in % 1000, middle = in / 1000 % 1000, high = in /
            1000000;
13        if (high != 0) {
14            print_3digit(high);
15            System.out.print(" million ");
16        }
17        if (middle != 0) {
18            print_3digit(middle);
19            System.out.print(" thousand ");
20        }
21        print_3digit(low);
22    }
23
24    // print a(a 3-bit decimal number)'s literal form.
25    public static void print_3digit(int a) {
26        boolean mark = false;
27        if (a / 100 != 0) {
28            System.out.print(literal_digit(a / 100) + " hundred");
29        }
30        if ((a %= 100) != 0) print_2digit(a);
31    }
32
33    // print a(a 2-bit decimal number)'s literal form.
34    public static void print_2digit(int a) {
35        if (a >= 20) {
36            System.out.print(switch (a / 10) {
37                case 2 -> " twenty";
```



```
38         case 3 -> " thirty";
39         case 4 -> " forty";
40         case 5 -> " fifty";
41         case 6 -> " sixty";
42         case 7 -> " seventy";
43         case 8 -> " eighty";
44         default -> " ninety";
45     });
46     if (a % 10 != 0) System.out.print(" " + literal_digit(a % 10));
47 } else if (a >= 10) {
48     System.out.print(switch (a % 10) {
49         case 0 -> " ten";
50         case 1 -> " eleven";
51         case 2 -> " twelve";
52         case 3 -> " thirteen";
53         case 4 -> " fourteen";
54         case 5 -> " fifteen";
55         case 6 -> " sixteen";
56         case 7 -> " seventeen";
57         case 8 -> " eighteen";
58         default -> " nineteen";
59     });
60 } else {
61     System.out.print(literal_digit(a));
62 }
63 }
64
65 // return a string which is a(digit)'s literal form.
66 public static String literal_digit(int a) {
67     return switch (a) {
68         case 1 -> "one";
69         case 2 -> "two";
70         case 3 -> "three";
71         case 4 -> "four";
72         case 5 -> "five";
73         case 6 -> "six";
74         case 7 -> "seven";
75         case 8 -> "eight";
```

```

76         default -> "nine";
77     };
78 }
79 }
```

2.4 结果展示

zero	<u>negative one</u>	one hundred	one thousand	five million two hundred ten
(a) input: 0	(b) input: -1	(c) input: 100	(d) input: 1000	(e) input: 5000210
<u>nine hundred ninety nine million nine hundred ninety nine thousand nine hundred ninety nine</u> 九百九十九万九千九百九十九				
(f) input: 999999999				

图 3: Output of Problem2

2.5 总结与收获

- 程序一开始的设计过程并没有题目分析中写的那么有条理，函数的需要是在程序编写过程中感觉到的；
- idea 自动把switch的默认语法转换成一种更短更新颖易读的写法，之前从没接触过；
- **空格处理**. 我们已经将输出模块化了，虽然题目中没有要求，但如何在单词之间设计优雅的空格是一个值得思考的问题，具体设计过程会涉及很多条件处理。

实验 3. 长度 n 的子序列最大乘积

从文件中输入一个数字序列字符串，计算给定的长度 n 的子序列中的最大乘积值。

例如：如果输入“1027839564”，指定长度为 3 的最大子序列乘积值为 270 ($9*5*6$)；指定长度为 5 的最大子序列乘积值为 7560 ($7*8*3*9*5$)。

备注：

1. 数字序列字符串的最大长度 maxLength 的范围为：[1..1000]；
2. n 的取值范围为 [1..maxLength-1]；
3. 程序要注意处理边界情况；
4. 程序的输入数据必须从文件中读取。

下图为一个长度为 1000 的字符串数字序列，在这个序列中，长度为 4 的最大子序列乘积为 5832 ($9*9*8*9$)。

```
73167176531330624919225119674426574742355349194934
96983520312774506326239578318016984801869478851843
85861560789112949495459501737958331952853208805511
12540698747158523863050715693290963295227443043557
66896648950445244523161731856403098711121722383113
62229893423380308135336276614282806444486645238749
30358907296290491560440772390713810515859307960866
70172427121883998797908792274921901699720888093776
65727333001053367881220235421809751254540594752243
52584907711670556013604839586446706324415722155397
53697817977846174064955149290862569321978468622482
83972241375657056057490261407972968652414535100474
82166370484403199890008895243450658541227588666881
16427171479924442928230863465674813919123162824586
17866458359124566529476545682848912883142607690042
24219022671055626321111109370544217506941658960408
07198403850962455444362981230987879927244284909188
84580156166097919133875499200524063689912560717606
05886116467109405077541002256983155200055935729725
71636269561882670428252483600823257530420752963450
```