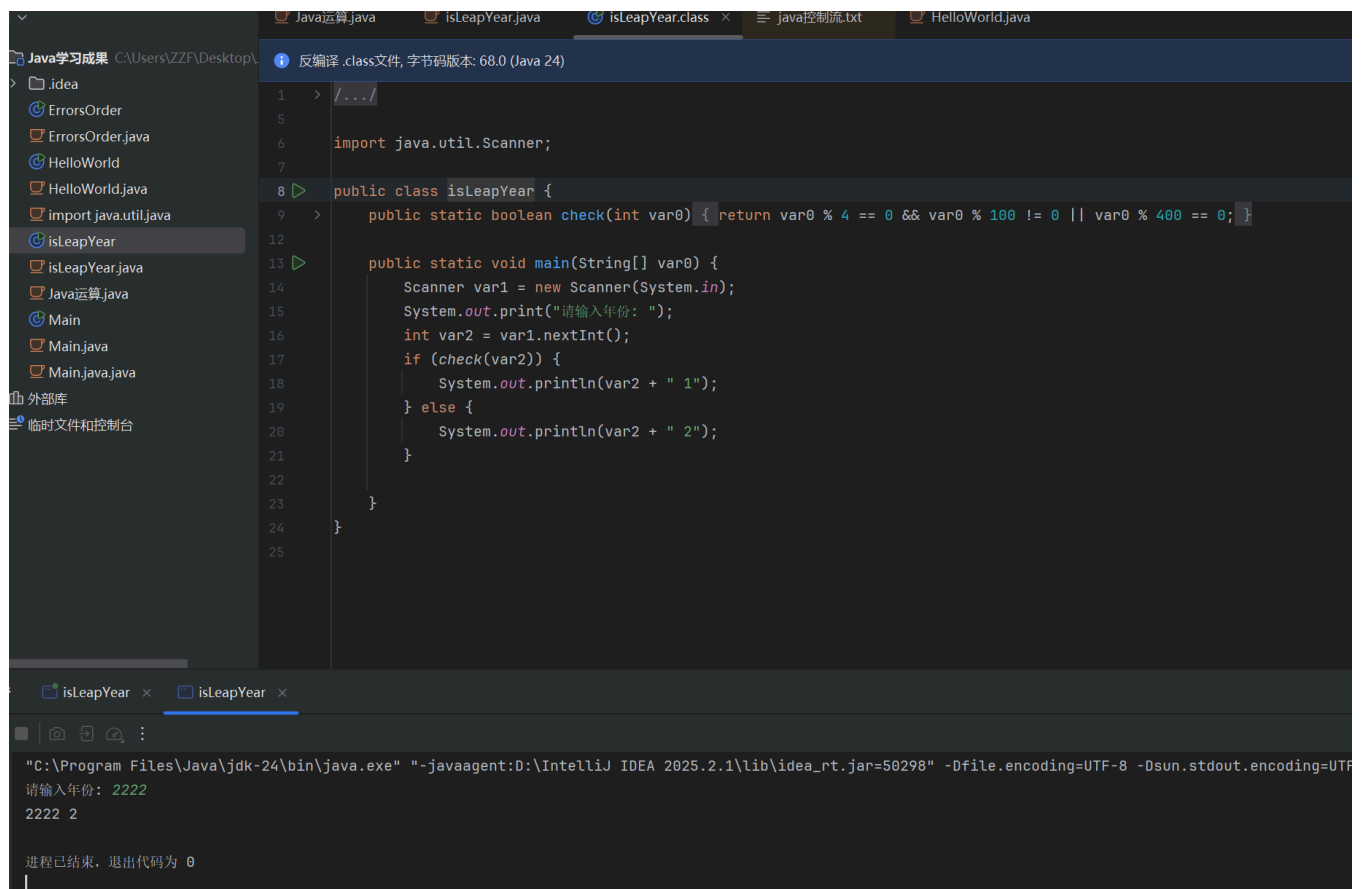


Task1

switch-case基本原理：首先计算 switch 后的表达式（控制表达式只能是整数型的常量），将表达式的结果与各个 case 后的值进行匹配，找到第一个匹配的 case 后，执行该 case 下的代码，如果没有break，则会顺序执行下一个 case直到遇到break;如果没有任何 case 匹配，且存在 default 分支，则执行 default 下的代码;如果没有匹配且无 default，则跳出整个 switch 结构

switch-case不是if-else的语法糖，switch-case 的底层是通过跳转表 / 哈希表实现的，执行效率远高于等价的 if-else，if-else 是通过条件判断的顺序执行，一旦找到第一个为真的条件，就执行对应的代码块并退出分支；如果所有条件都为假，则执行else



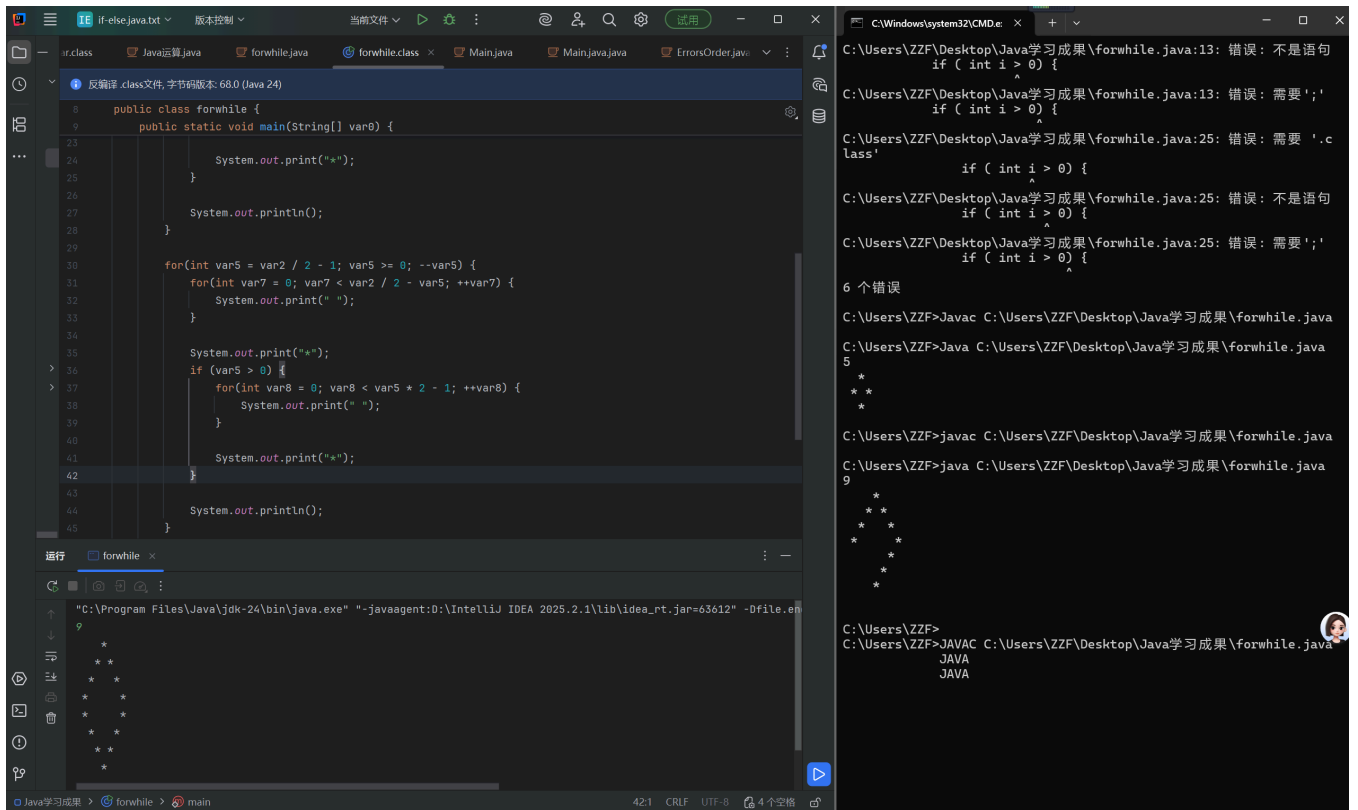
```
1  > /.../
5
6  import java.util.Scanner;
7
8  > public class isLeapYear {
9  >     public static boolean check(int var0) { return var0 % 4 == 0 && var0 % 100 != 0 || var0 % 400 == 0; }
12
13 >     public static void main(String[] var0) {
14         Scanner var1 = new Scanner(System.in);
15         System.out.print("请输入年份: ");
16         int var2 = var1.nextInt();
17         if (check(var2)) {
18             System.out.println(var2 + " 1");
19         } else {
20             System.out.println(var2 + " 2");
21         }
22     }
23 }
24
25
```

isLeapYear x isLeapYear x

```
"C:\Program Files\Java\jdk-24\bin\java.exe" "-javaagent:D:\IntelliJ IDEA 2025.2.1\lib\idea_rt.jar=50298" -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8
请输入年份: 2222
2222 2

进程已结束，退出代码为 0
```

Task2



Task3

递归

```
"C:\Program Files\Java\jdk-24\bin\java.exe" --enable-preview "-javaagent:D:\IntelliJ IDEA
请输入想算的斐波那契数列： 7
第 "n+" 个斐波那契数是： 13

进程已结束，退出代码为 0
```

```
Print.java  javaFibonacci01.java  javaFibonacci02.java  Hanno.java
package com.Example;

import java.util.Scanner;

public class javaFibonacci01 {
    public static int fibonacciRecursive(int n) { 3个用法
        if (n <= 0) {
            return 0;
        } else if (n == 1) {
            return 1;
        }
        return fibonacciRecursive(n - 1) + fibonacciRecursive(n - 2);
    }

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("请输入想算的斐波那契数列: ");
        int n = input.nextInt();
        int result = fibonacciRecursive(n);
        System.out.println("第 " + n + " 个斐波那契数是: " + result);
    }
}
```

迭代

```
java  Print.java  javaFibonacci01.java  javaFibonacci02.java  x  Hanno.java
1  package com.Example;
2
3  import java.util.Scanner;
4
5  public class javaFibonacci02 {
6      public static int fibonacciIterative(int n) { 1个用法
7          if (n <= 0) {
8              return 0;
9          } else if (n == 1) {
10             return 1;
11         }
12         int prevPrev = 0;
13         int prev = 1;
14         int current = 0;
15
16         for (int i = 2; i <= n; i++) {
17             current = prev + prevPrev;
18             prevPrev = prev;
19             prev = current;
20         }
21
22         return current;
23     }
24     public static void main(String[] args) {
25         Scanner input = new Scanner(System.in);
26         System.out.print("请输入想算的斐波那契数列: ");
27         int n = input.nextInt();
28         int result = fibonacciIterative(n);
29         System.out.println("第 "+n+" 个斐波那契数是: " + result);
30     }
31 }
```

运行 javaFibonacci02 x

```
"C:\Program Files\Java\jdk-24\bin\java.exe" --enable-preview "-javaagent:D:\IntelliJ IDEA
请输入想算的斐波那契数列: 9
第 "+n+" 个斐波那契数是: 34
进程已结束, 退出代码为 0
```

递归代码简洁，直接反映了斐波那契数列的数学定义

缺点：效率低，存在大量重复计算，时间复杂度为 $O(2^n)$ ，空间复杂度为 $O(n)$ （递归调用栈），不适合计算较大的 n 值（如 $n > 30$ ）

迭代效率高，时间复杂度为 $O(n)$ ，空间复杂度为 $O(1)$ ，适合计算较大的 n 值，

因为迭代的性能更好，尤其是当 n 值较大时，而递归主要用于理解递归思想

循环可以完全用递归来取代,因为循环和递归本质上都是控制程序重复执行某段代码的机制,只是实现方式不同;但需考虑递归会栈溢出,此时循环更高效

Task4

```
import java.util.*;

package com.Example;

import static java.nio.file.Files.move;

public class Hanno {
    public static void hanoi(int n) { 1个用法
        //
        if (n > 0) {
            //调用move递归辅助方法, 'A' 是起始柱, 'B' 是辅助柱, 'C' 是目标柱, 让铁饼从A柱子到C柱子, 用B作为辅助
            move(n, from: 'A', aux: 'B', to: 'C');
        }
    }

    private static void move(int n, char from, char aux, char to) { 3个用法
        //先决定递归的终止条件: 当 n=1 时, 直接将圆盘从A (from) 移到C(to)
        if (n == 1) {
            System.out.println(from + "->" + to);
            //写递归情况: 让n-1先暂放在辅助柱里, n放在目标柱, 再把n-1放进目标柱, 使得n-1在n之上
        } else {
            move(n - 1, from, to, aux); //先将n-1个圆盘从A(from)柱移到B(aux)柱
            System.out.println(from + "->" + to); //将最底下的第n个圆盘从A (from) 柱直接移到C (to) 柱
            move(n - 1, aux, from, to); //再将n-1个圆盘从B (aux) 柱移到C (to) 柱
        }
    }

    public static void main(String[] args) {
        System.out.println("当n=3时的移动");
        hanoi(n: 3); //通过 move 方法的递归逻辑, 逐步完成3个圆盘从A柱到C柱的移动
    }
}
```

运行 Hanno x

```
"C:\Program Files\Java\jdk-24\bin\java.exe" --enable-preview "-javaagent:D:\IntelliJ IDEA 2025.2.1\lib\ic
当n=3时的移动
A->C
A->B
C->B
A->C
B->A
B->C
A->C
```