java04

Task1

```
package task4;

public class task4_1 {
    public static void main(String[] args) {
        Year year = new Year();
        System.out.println(year.isLeapYear(300));
    }


}
```

```
package task4;

public class Year {
    //这个函数用于判断传入的年份是否为闰年
    //是闰年返回1，不是闰年返回2
    boolean isLeapYear(int year){
        if (year % 4 == 0){
            if (year % 100 == 0){
                if (year % 400 == 0){
                    return true;
                }
                return false;
            }
            return true;
        }
        return false;
    }
}
```

以上就是Task1的代码，使用了if的嵌套，逻辑基于生活中的口诀“四年一闰，百年不闰，四百年又闰”

关于`switch-case`是否是`if-else`的语法糖，由于if-else可以进行不等关系的判断，Switch-case有穿透现象，所以不能直接说`switch-case`是`if-else`的语法糖。

Task2

```
package task4;

public class task4_2 {
    public static void main (String[] args){
        Print print = new Print();
        print.print(11);
    }
}
```

```
package task4;

public class Print {
    void print(int n){
//非法判断
        if (n % 2 == 0){
            System.out.println("请让n取奇数");
            return;
        }
//      确定空格长度
        for (int j = 0; j < n; j += 1) {
            int kong1 = n / 2 - j;
            if (kong1 < 0)kong1 = -kong1;
            int kong2 = Math.min(-1 + 2 * j  ,  2 * (n - j - 1) - 1);

//          先打印前面的空格
            for (int i = 0; i < kong1; i += 1) {
                System.out.print(" ");
            }
            System.out.print("*");

//          打印中间的空格
            if (kong2 < 0 ||kong2 == n){
                System.out.println();
            }else{
                for (int i = 0; i < kong2; i += 1) {
                    System.out.print(" ");
                }
                System.out.print("*");
                System.out.println();
            }
        }
    }
}
```

本题第一部分空白有规律：依次递减，减为零后又重新增加；

第二部分空白也有规律：0,1,3,5，...，5,3,1,0 所以据此完成打印。

Task3

```java
package task4;

public class task4_3 {
    public static void main(String[] args) {
        Fibonacci fibonacci = new Fibonacci();
        for (int i = 1; i < 20; i++) {
            System.out.println(fibonacci.Fibonacci(i));
        }
    }
}
```

```java
package task4;

public class Fibonacci {
    int Fibonacci(int n){
        if (n <= 0 || n % 1 == 0) {
            if(n == 1 || n == 2) return 1;
            int fib = Fibonacci(n-1) + Fibonacci(n-2);
            return fib;
        }else {
            System.out.println("请输入一个正整数");
            return -1;
        }
    }
}
```

以上就是递归版本

```java
package task4;

public class task4_3b {
    public static void main(String[] args) {
        int a1 = 1;
        int a2 = 1;
        for (int i = 1; i < 20; i++) {
            System.out.println(a1);
            int a = a1;
            int b = a2;
            int c = a + b;
            a1 = b;
            a2 = c;
        }
    }
}
```

以上就是迭代版本

关于为什么一般偏好迭代?

要我来说就是逻辑简单，好写!

但是查阅deepseek得知：核心原因是递归会多次调用栈，性能较差，不利于调试

Task4

```
package task4;

public class task4_4 {
    public static void main(String[] args) {
        move(3, 'A', 'B', 'C');
    }
    public static void move(int n , char from, char zhong, char to) {
        if (n == 1) {
            System.out.println(from + " -> " + to);
            return;
        }
        move(n-1, from, to, zhong);
        System.out.println(from + " -> " + to);
        move(n-1, zhong, from, to);
    }
}
```

骄傲的说一下：汉诺塔问题我在高中已经研究过了（送花）

以下代码基于这样的逻辑：

1.先把n - 1块铁饼任然按照从大到小的顺序移动到中间的柱子上

2.把最大的移到指定柱子上

3.把中间柱子上的移到指定柱子上