方法

本节目标

1. 可以对什么是方法有自己的初步认知。
2. 理解方法分为定义和调用两个不同的阶段。
3. 会定义方法。
4. 会调用方法。
5. 可以对过程建模出方法。

1. 示例：计算 1! + 2! + 3! + 4! + 5!

1.1 不使用方法完成

```
public class Test {
    public static void main(String[] args) {
        int sum = 0;

        for (int i = 1; i <= 5; i++) {
            int tmp = 1;
            for (int j = 1; j <= i; j++) {
                tmp *= j;
            }
            sum += tmp;
        }

        System.out.println("sum = " + sum);
    }
}
```

1.2 使用方法完成

```
public class Test {
    public static int fac(int n) {
        int f = 1;

        for (int i = 1; i <= n; i++) {
            f *= i;
        }

        return f;
    }

    public static void main(String[] args) {
        int s = 0;

        for (int i = 1; i <= 5; i++) {
            s += fac(i);
        }

        System.out.println("sum = " + s);
```

```
    }
}
```

1.3 思考：使用方法的好处

1. 将重复代码抽取成方法，方便多次调用
2. 一个方法专注完成一件事情

2. 方法的定义和使用

如上面例子所示，方法分为**定义**和**使用**两个不同的阶段。使用又被称为执行、调用。

以做菜为例，定义一个方法，只是为制作某个菜书写了一份菜谱。而使用一个方法，则是按照之前写好的菜谱，真正的开始做菜。

2.1 方法的定义(define)

方法定义中，最重要的有以下几个个元素：

1. 方法的名称。如例子中的"绘制矩形"。
2. 方法的指令们。
3. 方法的形参（parameter / formal parameter)。
4. 方法可能出现的返回值类型。当没有返回值的时候，使用 void（缺乏的、无效的）来表示。
5. 其他修饰信息。**public**和**static**我们在以后的课程中讨论。

java 规定了方法定义的标准格式如下：

```
public static 方法返回值类型 方法名称(形参列表) {
    方法的指令;

    return 返回值;
}
```

例如(为了讲解方便，使用了中文名称，通常不是太建议)：

```
public static int 两个数相加(int a, int b) {
    int 和 = a + b;

    return 和;
}
```

方法名称：两个数相加
形参列表：(int a, int b)
返回值类型：int

```
public static double 求三个数的平均数(double a, double b, double c) {
    double 和 = a + b + c;
    double 平均数 = 和 / 3;

    return 平均数;
}
```

方法名称：求三个数的平均数
形参列表：(double a, double b, double c)
返回值类型：double

```java
public static void 打印HelloWorld() {
    System.out.println("Hello World");
}

方法名称：打印HelloWorld
形参列表：不需要，空的 -- ()
返回值类型：没有，空的 -- void
```

```java
public static void 没有返回值一样可以使用return(int a) {
    if (a == 0) {
        return;
    }

    System.out.println("a 不是 0");
}

方法名称：没有返回值一样可以使用return
形参列表：(int a)
返回值类型：没有，空的 -- void
```

包括我们的入口方法 —— main，也同样遵守规则

```java
public static void main(String[] args) {
    System.out.println("Hello World");
}

方法名称：main
形参列表：(String[] args) -- String[] 的类型是字符串数组，下节课我们就学习
返回值类型：没有，空的 -- void
```

2.2 方法的调用(invocation)

方法的使用中，有以下几个重要的元素：

1. 调用哪个方法。方法名称。
2. 使用哪些具体的值，进行本次方法调用。调用时的实参（argument / actual parameter）。
3. 调用方法后可能得到的返回值的后续处理。保存或者直接再次使用。

java 规定了方法定义的标准格式如下

```java
// 不关心返回值
方法名称(实参列表);

// 将返回值保存到变量中
变量 = 方法名称(实参列表);

// 直接使用返回值参与运算
方法名称(实参列表) + 方法名称(实参列表);
```

例如(为了讲解方便，使用了中文名称，通常不是太建议):

```java
// 调用方法，不需要实参，不关心返回值
打印HelloWorld();
```

```
// 调用方法，10、20 是实参，也就是计算 10 和 20 的和。不关心返回值
两个数相加(10, 20);

// 调用方法，10、20 是实参，也就是计算 10 和 20 的和。把计算的结果，保存到 r 这个变量中，r 的值最后是 30
int r = 两个数相加(10, 20);

// 调用方法，10、20 是实参，也就是计算 10 和 20 的和。把计算的结果，参与运算，再将最终的结果保存到变量 r 中，r 的值最后是 60
int r = 两个数相加(10, 20) + 30;

// 调用方法，10、20 是实参，也就是计算 10 和 20 的和。把计算的结果，作为实参，重新发起方法调用，再将最终的结果保存到变量 r 中，r 的值最后是 60
int r = 两个数相加(两个数相加(10, 20), 30);
```

```
// 直接在方法定义中，返回一个方法调用的返回值
public static int 无意义的方法(int a, int b) {
    return 两个数相加(a, b);
}
```

3. 课堂练习

3.1 示例：求两个数的最小值

```
public class Test {
    public static int min(int a, int b) {
        if (a > b) {
            return b;
        } else {
            return a;
        }
    }

    public static void main(String[] args) {
        System.out.println("3, 7 的最小值是 " + min(3, 7));
        System.out.println("2, 7 的最小值是 " + min(2, 7));
        System.out.println("7, 7 的最小值是 " + min(7, 7));
        System.out.println("7, 3 的最小值是 " + min(7, 3));
    }
}
```

其实 java 中已经事先定义了很多常见的方法，例如求两个数的最小值

```
public class Test {
    public static void main(String[] args) {
        System.out.println("3, 7 的最小值是 " + Integer.min(3, 7));
        System.out.println("2, 7 的最小值是 " + Integer.min(2, 7));
        System.out.println("7, 7 的最小值是 " + Integer.min(7, 7));
        System.out.println("7, 3 的最小值是 " + Integer.min(7, 3));
    }
}
```

3.2 示例：打印 1 - 100 之间所有的素数

```
public class Test {
    public static boolean isPrimeNumber(int n) {
        for (int i = 2; i < n; i++) {
            if (n % i == 0) {
                return false;
            }
        }

        return true;
    }

    public static void main(String[] args) {
        for (int i = 1; i <= 100; i++) {
            if (isPrimeNumber(i)) {
                System.out.println(i + " 是素数");
            }
        }
    }
}
```

3.3 示例：输出 1000 - 2000 之间所有的闰年

```
public class Test {
    public static boolean isLeapYear(int year) {
        if (year % 100 == 0) {
            if (year % 400 == 0) {
                return true;
            } else {
                return false;
            }
        } else {
            if (year % 4 == 0) {
                return true;
            } else {
                return false;
            }
        }
    }

    public static void main(String[] args) {
        for (int y = 1000; y <= 2000; y++) {
            if (isLeapYear(y)) {
                System.out.println(y + " 是闰年");
            }
        }
    }
}
```

3.4 示例：输出乘法口诀表

```
public class Test {
    public static void printLine(int lineNumber) {
        for (int i = 1; i <= lineNumber; i++) {
            System.out.printf("%d x %d = %-2d    ", i, lineNumber, lineNumber *
i);
        }
        System.out.println();
    }

    public static void printMultiplicationTable() {
        for (int i = 1; i <= 9; i++) {
            printLine(i);
        }
    }

    public static void main(String[] args) {
        printMultiplicationTable();
    }
}
```

3.5 示例：求两个正整数的最大公约数

```
public class Test {
    public static int calculateGreatestCommonDivisor(int m, int n) {
        int min = Integer.min(m, n);

        for (int i = min; i > 0; i--) {
            if (m % i == 0 && n % i == 0) {
                return i;
            }
        }
        // 理论上不会走到这里的，因为当 i == 1 的时候，必然返回
        // 但 java 编译器要求每个分支都必须有返回值，所以添加返回 1
        return 1;
    }

    public static void main(String[] args) {
        for (int i = 10; i <= 100; i++) {
            for (int j = 10; j <= 100; j++) {
                int gcd = calculateGreatestCommonDivisor(i, j);
                if (gcd != 1 && gcd != 2) {
                    // 考虑到最大公约数是 1 或者 2 的太多了，我们不打印
                    System.out.printf(
                        "%d 和 %d 的最大公约数是 %d%n", i, j,
calculateGreatestCommonDivisor(i, j)
                    );
                }
            }
        }
    }
}
```

3.6 示例：计算1/1-1/2+1/3-1/4+1/5 ..... + 1/99 - 1/n 的值（n 一定是偶数）

```
public class Test {
    // 计算 1/(n-1) - 1/n
    public static double calculateItem(int n) {
        return 1.0 / (n - 1) - 1.0 / n;
    }

    // n 一定是偶数
    public static double calculateSeries(int n) {
        // 两个两个处理
        double sum = 0;

        for (int i = 2; i <= n; i += 2) {
            sum += calculateItem(i);
        }

        return sum;
    }

    public static void main(String[] args) {
        System.out.println("n = 10, sum = " + calculateSeries(10));
        System.out.println("n = 100, sum = " + calculateSeries(100));
    }
}
```

3.7 示例：编写程序数一下 1到 100 的所有整数中出现多少个数字9

```
public class Test {
    // 计算 n 这个数中，一共有多少个 m
    // n > 0
    // m 是 0 到 9
    public static int calculateCountOfMInN(int n, int m) {
        int count = 0;

        do {
            int digit = n % 10;
            if (digit == m) {
                count++;
            }

            n = n / 10;
        } while (n != 0);

        return count;
    }

    // 计算 1 到 n 的数列中，一共有多少个 m
    // n > 0
    // m 是 0 到 9
    public static int calculateCountOfMInSeries(int n, int m) {
        int count = 0;

        for (int i = 1; i <= n; i++) {
```

```
            count += calculateCountOfMInN(i, m);
        }

        return count;
    }

    public static void main(String[] args) {
        System.out.println("1 到 100 的所有整数中，共出现了 " +
calculateCountOfMInSeries(100, 9) + " 个 9.");
    }
}
```

3.8 示例：求出0～999之间的所有“水仙花数”并输出

```
public class Test {
    // 计算 a 的 n 次方
    public static int power(int a, int n) {
        int p = 1;

        for (int i = 0; i < n; i++) {
            p *= a;
        }

        return p;
    }

    // 计算 n 的各位数字的立方和
    public static int cubeOfDigit(int n) {
        int sum = 0;

        do {
            int digit = n % 10;

            sum += power(digit, 3);

            n = n / 10;
        } while (n != 0);

        return sum;
    }

    // 计算 n 是不是水仙花数
    public static boolean isNarcissisticNumber(int n) {
        return cubeOfDigit(n) == n;
    }

    public static void main(String[] args) {
        for (int i = 100; i <= 999; i++) {
            if (isNarcissisticNumber(i)) {
                System.out.println(i + " 是一个水仙花数");
            }
        }
    }
}
```

3.9 示例：猜数字

```
import java.util.Random;
import java.util.Scanner;

public class Test {
    public static int generateToGuess() {
        Random random = new Random();
        return random.nextInt(100);
    }

    // true：猜对了    false：猜错了
    public static boolean guess(Scanner scanner, int toGuess) {
        System.out.println("请猜猜看（1-100）：");
        int num = scanner.nextInt();

        if (num < toGuess) {
            System.out.println("低了");
            return false;
        } else if (num > toGuess) {
            System.out.println("高了");
            return false;
        } else {
            System.out.println("猜对了");
            return true;
        }
    }

    public static void playGame(Scanner scanner) {
        int toGuess = generateToGuess();
        while (true) {
            if (guess(scanner, toGuess)) {
                break;
            }
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        playGame(scanner);
    }
}
```

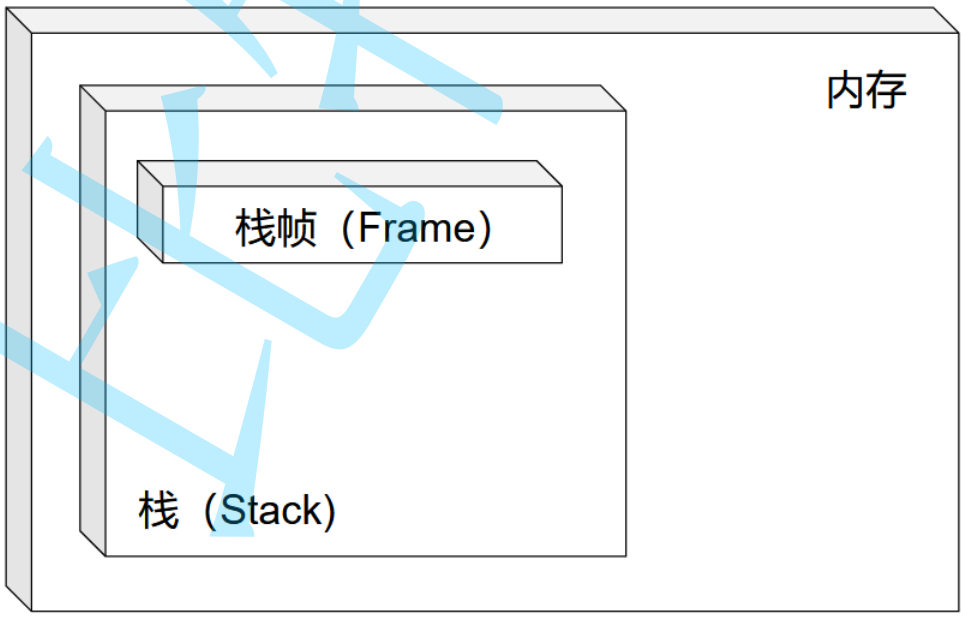
3.10 示例：编写代码模拟三次密码输入的场景

```
import java.util.Scanner;

public class Test {
    public static String typePassword(Scanner scanner) {
        System.out.println("请设置密码：");
        return scanner.nextLine();
    }

    public static boolean verifyPassword(Scanner scanner, String actualPassword,
int retryTimes) {
```

```
        for (int i = 1; i <= retryTimes; i++) {
            System.out.println("请输入密码：");
            String tryPassword = scanner.nextLine();

            // 这里不能使用 actualPassword == tryPassword
            // 必须使用 actualPassword.equals(tryPassword)
            // 原因下节课讲解
            if (actualPassword.equals(tryPassword)) {
                return true;
            }

            if (i != retryTimes) {
                System.out.printf("密码错误，还有 %d 次尝试机会%n", retryTimes - i);
            }
        }

        return false;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        String password = typePassword(scanner);
        if (verifyPassword(scanner, password, 3)) {
            System.out.println("登录成功");
        } else {
            System.out.println("已经达到尝试上限，退出程序");
        }
    }
}
```

4. 方法执行过程分析

4.1 内存、栈、栈帧的关系



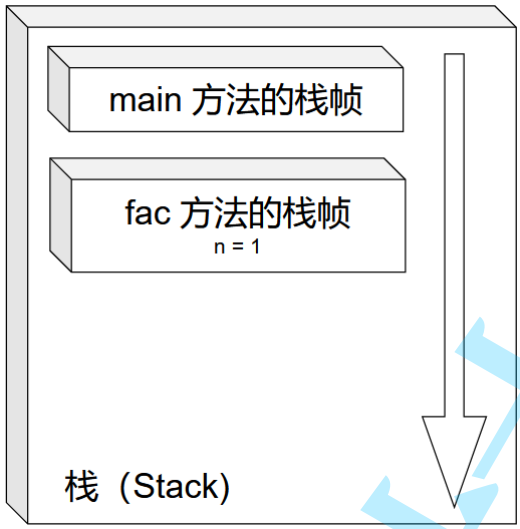
4.2 执行过程图示

我们以 示例 3.1 为例观察过程

```
public class Test {
    public static int fac(int n) {
        int f = 1;

        for (int i = 1; i <= n; i++) {
            f *= i;
        }

        return f;
    }

    public static void main(String[] args) {
        int s = 0;

        for (int i = 1; i <= 5; i++) {
            s += fac(i);
        }

        System.out.println("sum = " + s);
    }
}
```

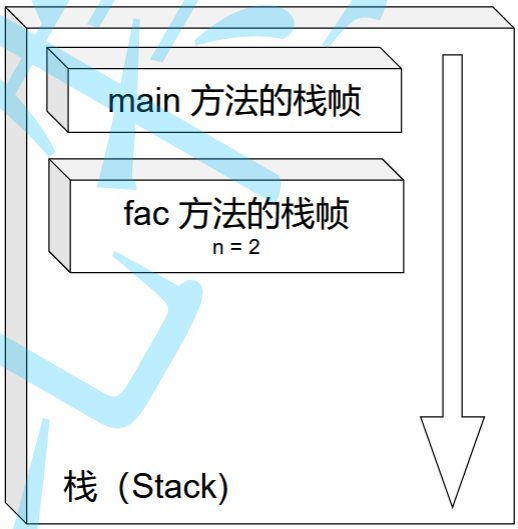
刚刚开始执行 main 方法：

第一次执行 fac 方法：

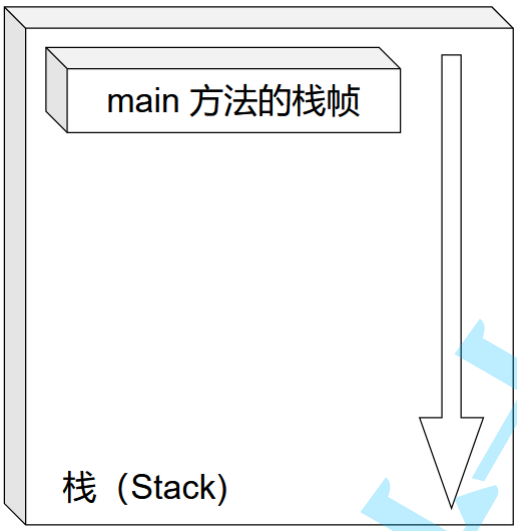


第一次执行 fac 方法结束，回到 main 方法：

第二次执行 fac 方法：

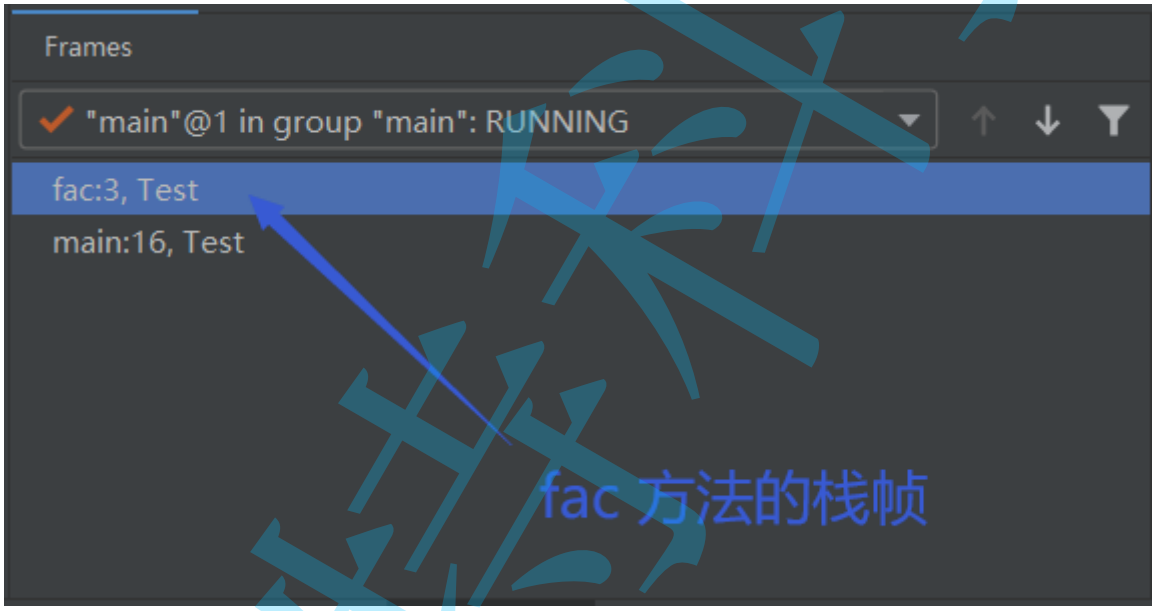


第二次执行 fac 方法结束，回到 main 方法：



剩余的过程省略。

4.3 通过调试工具观察栈帧的变化



5. 方法的重载

有些时候我们需要用一个函数同时兼容多种参数的情况, 我们就可以使用到方法重载.

5.1 重载要解决的问题

代码示例

```
class Test {
    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        int ret = add(a, b);
        System.out.println("ret = " + ret);

        double a2 = 10.5;
        double b2 = 20.5;
        double ret2 = add(a2, b2);
        System.out.println("ret2 = " + ret2);
    }
```

```
    public static int add(int x, int y) {
        return x + y;
    }
}

// 编译出错
Test.java:13: 错误: 不兼容的类型: 从double转换到int可能会有损失
                double ret2 = add(a2, b2);
                                  ^
```

由于参数类型不匹配, 所以不能直接使用现有的 add 方法.

那么是不是应该创建这样的代码呢?

代码示例

```
class Test {
    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        int ret = addInt(a, b);
        System.out.println("ret = " + ret);

        double a2 = 10.5;
        double b2 = 20.5;
        double ret2 = addDouble(a2, b2);
        System.out.println("ret2 = " + ret2);
    }

    public static int addInt(int x, int y) {
        return x + y;
    }

    public static double addDouble(double x, double y) {
        return x + y;
    }
}
```

这样的写法是对的(例如 Go 语言就是这么做的), 但是 Java 认为 `addInt` 这样的名字不友好, 不如直接就叫 `add`

5.2 使用重载

代码示例

```
class Test {
    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        int ret = add(a, b);
        System.out.println("ret = " + ret);
```

```
        double a2 = 10.5;
        double b2 = 20.5;
        double ret2 = add(a2, b2);
        System.out.println("ret2 = " + ret2);

        double a3 = 10.5;
        double b3 = 10.5;
        double c3 = 20.5;
        double ret3 = add(a3, b3, c3);
        System.out.println("ret3 = " + ret3);
    }

    public static int add(int x, int y) {
        return x + y;
    }

    public static double add(double x, double y) {
        return x + y;
    }

    public static double add(double x, double y, double z) {
        return x + y + z;
    }
}
```

方法的名字都叫 add. 但是有的 add 是计算 int 相加, 有的是 double 相加; 有的计算两个数字相加, 有的是计算三个数字相加.

同一个方法名字, 提供不同版本的实现, 称为 **方法重载**

5.3 重载的规则

针对同一个类:

- 方法名相同
- 方法的参数不同(参数个数或者参数类型)
- 方法的返回值类型不影响重载.

代码示例

```
class Test {
    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        int ret = add(a, b);
        System.out.println("ret = " + ret);
    }

    public static int add(int x, int y) {
        return x + y;
    }

    public static double add(int x, int y) {
        return x + y;
```

```
    }
}

// 编译出错
Test.java:13: 错误: 已在类 Test中定义了方法 add(int,int)
        public static double add(int x, int y) {
                             ^
1 个错误
```

当两个方法的名字相同, 参数也相同, 但是返回值不同的时候, 不构成重载.