方法

方法是什么

- 方法是程序中最小的执行单元。

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println(" HelloWorld ");
    }
}
```

main方法，主方法

方法是什么

用输出语句描述，玩一局游戏。

```
public class PlayGame {
    public static void main(String[] args) {
        System.out.println("步骤 1: 选择角色.");
        System.out.println("步骤 2: 设定角色属性.");
        System.out.println("步骤 3: 开始游戏，进入虚拟世界.");
        System.out.println("步骤 4: 探索环境，接受任务.");
        System.out.println("步骤 5: 完成任务，与怪物战斗.");
        System.out.println("步骤 6: 升级，获取经验和金币.");
        System.out.println("步骤 7: 购买装备，增强能力.");
        System.out.println("步骤 8: 持续探索和战斗.");
        System.out.println("步骤 9: 游戏结束，保存游戏进度.");
    }
}
```

方法是什么

玩两局游戏？

```
public class PlayGame {
    public static void main(String[] args) {
        System.out.println("步骤 1: 选择角色.");
        System.out.println("步骤 2: 设定角色属性.");
        System.out.println("步骤 3: 开始游戏，进入虚拟世界.");
        System.out.println("步骤 4: 探索环境，接受任务.");
        System.out.println("步骤 5: 完成任务，与怪物战斗.");
        System.out.println("步骤 6: 升级，获取经验和金币.");
        System.out.println("步骤 7: 购买装备，增强能力.");
        System.out.println("步骤 8: 持续探索和战斗.");
        System.out.println("步骤 9: 游戏结束，保存游戏进度.");

        System.out.println("步骤 1: 选择角色.");
        System.out.println("步骤 2: 设定角色属性.");
        System.out.println("步骤 3: 开始游戏，进入虚拟世界.");
        System.out.println("步骤 4: 探索环境，接受任务.");
        System.out.println("步骤 5: 完成任务，与怪物战斗.");
        System.out.println("步骤 6: 升级，获取经验和金币.");
        System.out.println("步骤 7: 购买装备，增强能力.");
        System.out.println("步骤 8: 持续探索和战斗.");
        System.out.println("步骤 9: 游戏结束，保存游戏进度.");

    }
}
```

方法是什么

玩两局游戏？

```
System.out.println("步骤 1: 选择角色.");
System.out.println("步骤 2: 设定角色属性.");
System.out.println("步骤 3: 开始游戏，进入虚拟世界.");
System.out.println("步骤 4: 探索环境，接受任务.");
System.out.println("步骤 5: 完成任务，与怪物战斗.");
System.out.println("步骤 6: 升级，获取经验和金币.");
System.out.println("步骤 7: 购买装备，增强能力.");
System.out.println("步骤 8: 持续探索和战斗.");
System.out.println("步骤 9: 游戏结束，保存游戏进度.");
```

```
public class PlayGame {
    public static void main(String[] args) {
        playGame();
        playGame();

    }
}
```

方法是什么

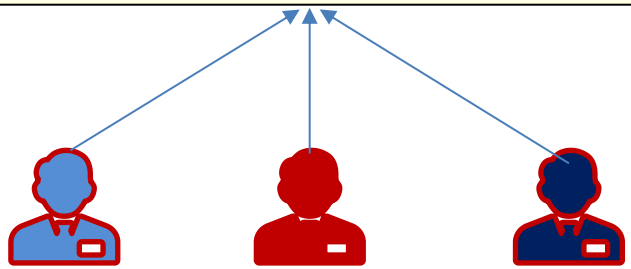
- 方法是一种语法结构，它可以把一段代码封装成一个功能，以方便重复调用。

```
public class Test {
    public static void main(String[] args) {


    }

    public static int sum(int a, int b){
        int c = a + b;
        return c;
    }
}
```

使用方法的好处是？

- 提高了代码的复用性。
- 让程序的逻辑更清晰。

```
public static int sum(int a, int b){
        int c = a + b;
        return c;
}
```



目录

Contents

1、最简单的方法定义和调用

```
public static void 方法名(  ){

        方法体代码(需要执行的功能代码)

}
```

示例：

```
public static void playGame(  ){
        System.out.println("步骤 1: 选择角色.");
        System.out.println("步骤 2: 设定角色属性.");
        System.out.println("步骤 3: 开始游戏，进入虚拟世界.");
        ……
}
```

1、最简单的方法定义和调用

格式：

方法名( );

示例：

playGame( );

```
public static void playGame( ){
        System.out.println("步骤 1: 选择角色.");
        System.out.println("步骤 2: 设定角色属性.");
        System.out.println("步骤 3: 开始游戏，进入虚拟世界.");
        ……
}
```

练习 • 看代码说结果

```
public class Test {
    public static void main(String[] args) {
     System.out.println("步骤 1");
     playGame( );
      System.out.println("步骤 2");
    }
    public static void playGame( ){
     System.out.println("步骤 3");
      System.out.println("步骤 4");
   }
}
```

单选题 1分

设置

```
public class Test {
    public static void main(String[] args) {
     System.out.println("步骤 1");
     playGame( );
      System.out.println("步骤 2");
    }
    public static void playGame( ){
     System.out.println("步骤 3");
      System.out.println("步骤 4");
   }
}
```

A
步骤 1
步骤 2
步骤 3
步骤 4

B
步骤 1
步骤 3
步骤 4
步骤 2

提交

练习 • 看代码说结果

```
public class Test {
    public static void main(String[] args) {
     System.out.println("步骤 1");
     playGame( );
      System.out.println("步骤 2");
    }
    public static void playGame( ){
     System.out.println("步骤 3");
      System.out.println("步骤 4");
   }
}
```

```
步骤 1
步骤 3
步骤 4
步骤 2
```

2、带参数的方法定义和调用

```
public static void 方法名( 参数1, 参数2, .....){

        方法体代码(需要执行的功能代码)

}
```

示例：

```
public static void getSum( int number1, int number2 ){
            ......
}
```

2、带参数的方法定义和调用

格式：

方法名(参数1, 参数2, ...);

示例：

getSum(10，20);

```
public static void getSum( int number1, int number2 ){
            ......
}
```

注意：带参数的方法调用时，参数的数量与类型必须与方法定义中小括号里面的变量一致。

2、带参数的方法定义和调用

```
public class Example {
    // 这是一个带参数的方法定义，方法名为'greet'，参数为'String name'
    public static void greet(String name) {
        System.out.println("Hello, " + name + "!");
    }

    public static void main(String[] args) {
        // 这是一个带参数的方法调用，我们传递参数 'World' 给 'greet' 方法
        greet("World");
    }
}
```

注意：带参数的方法调用时，参数的数量与类型必须与方法定义中小括号里面的变量一致。

2、带参数的方法定义和调用

```
public class Example {
    // 这是一个带有两个整数参数（int number1 和 int number2）的方法定义。
    public static void addAndPrint(int number1, int number2) {
        int sum = number1 + number2;
        System.out.println(sum);
    }

    public static void main(String[] args) {
        // 这是不带返回值的方法调用，我们传递参数 5 和 3 给 'addAndPrint' 方法。
        addAndPrint(5, 3);
    }
}
```

注意：带参数的方法调用时，参数的数量与类型必须与方法定义中小括号里面的变量一致。

3、带返回值的方法定义和调用

```
public static 返回值类型 方法名( 参数1, 参数2, .....){
        方法体;
        return  返回值;
}
```

示例：

```
public static int getSum( int a, int b ){
        int c = a + b;
         return c;
}
```

3、带返回值的方法定义和调用

01

直接调用：

方法名 (实参);

02

赋值调用：

整数类型 变量名 = 方法名 (实参);

03

输出调用：

```
System.out.println(方法名 (实参));
```

3、带返回值的方法定义和调用

```
public class Example {
    // 这是一个带两个参数的方法定义，方法名为'add'，参数为'int number1' 和 'int number2'
    public static int add(int number1, int number2) {
        return number1 + number2;
    }

    public static void main(String[] args) {
        // 这是一个带参数的方法调用，我们传递参数 8 和 2 给 'add' 方法
        int sum = add(8, 2);
        System.out.println("The sum of 8 and 2 is " + sum);
    }
}
```

方法定义的完整格式

```
修饰符  返回值类型  方法名( 形参列表 ){
        方法体代码(需要执行的功能代码)
        return 返回值;
}
```

示例：使用方法对2个整数求和并返回。

```
方法的修饰符   返回值类型  方法名称   形参列表
public static  int     add ( int a , int b ){
    int c = a + b;          → 方法的执行代码
    return c;               → 返回值
}
```

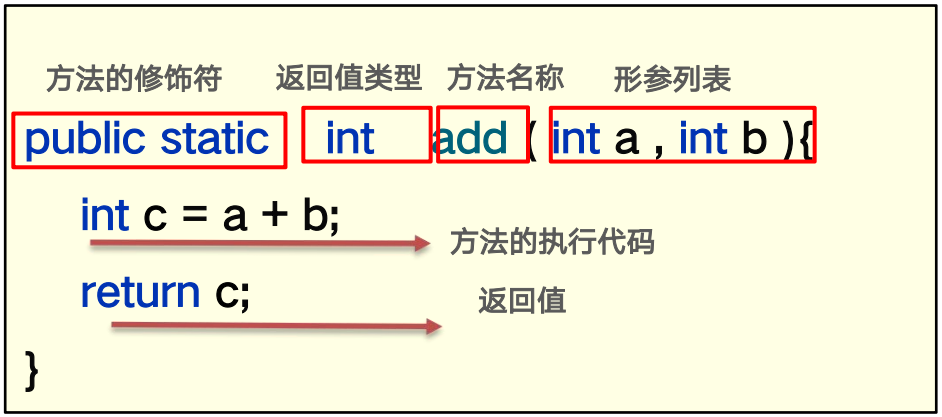
方法的调用格式

- 方法必须调用才可以跑起来，调用格式：

```
方法名(…);
```

```
int sum = add(10, 20);
System.out.println(sum);
```

方法格式的注意点



- 方法的修饰符：暂时都使用public static 修饰。
- 方法申明了具体的返回值类型，内部必须使用**return**返回对应类型的数据。
- 形参列表可以有多个，甚至可以没有； 如果有多个形参，多个形参必须用“，”隔开，且**不能给初始化值**。

return关键字单独使用

- return; ---> 可以立即跳出并结束当前方法的执行。

```
public class Test {
    public static void main(String[] args) {
        System.out.println("开始");
        chu(10 , 0);
        System.out.println("结束");
    }

    public static void chu(int a , int b){
        if(b == 0){
            System.out.println("您的数据有误！！ 不执行！！ ");
            return; // 直接结束当前方法chu
        }
        int c = a / b;
        System.out.println("除法结果是：  "+c);
    }
}
```

总结

1．如果要直接结束当前方法的执行，怎么解决？

- return; 跳出并立即结束所在方法的执行。
- break; 跳出并结束当前所在循环的执行。
- continue; 结束当前所在循环的当次继续，进入下一次执行。

目录

Contents

```
public class MethodDemo1 {
    public static void main(String[] args) {

        public static void playGame(){
            System.out.println("玩游戏");
        }

    }

}
```

方法常见问题

- 方法的编写顺序无所谓。
- 方法与方法之间是平级关系，不能嵌套定义。
- 方法的返回值类型为void（无返回值），方法内则不能使用return返回数据，如果方法的返回值类型写了具体类型，方法内部则必须使用return返回对应类型的数据。
- return语句下面，不能编写代码，因为永远执行不到，属于无效的代码。
- 方法不调用就不执行，调用时必须严格匹配方法的参数情况。
- 有返回值的方法调用时可以选择定义变量接收结果，或者直接输出调用，甚至直接调用；无返回值方法的调用只能直接调用。

目录

Contents

方法定义的技巧说明

```
修饰符  返回值类型  方法名( 形参列表 ){
        方法体代码(需要执行的功能代码)
        return 返回值;
}
```

返回值类型　　形参列表

```
public static int add(int a , int b){
    int c = a + b;
    return c;
}
```

- 修饰符：public static (暂时固定的)
- 返回值类型到底写什么是需要分析的。
- 方法名称：自己取名，有意义，英文小写，驼峰模式。（有意义的名字即可）
- 形参列表到底写什么也是需要分析的。
- 方法体代码：完成自己需要写的功能代码即可。

结论：定义方法时真正需要关注的就两点：1、分析方法是否需要申明返回值类型；2、分析方法是否需要接收参数。

案例 计算1-n的和返回

需求：定义一个方法，方法中计算出 1-n的和并返回。

分析：

1. 根据格式编写方法 ----> 因n不固定，故方法需要声明形参接收；要返回结果，还需申明返回值类型。

2. 方法内部使用 for 循环计算出 1-n 的和并返回。

总结

1.定义方法重点关注的是哪两点？

- **方法是否需要申明返回值类型。**
- **方法是否需要定义形参列表。**

2.如何使用方法完成1-n的求和?

```
public static int sum(int n){
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum += i;
    }
    return sum;
}
```

目录

Contents

案例 判断整数是奇数还是偶数

需求：拿一个整数，然后调用方法，把整数交给方法，在方法中输出该数为奇数还是偶数

分析：

1. 根据格式编写方法 ---->

   因要传入数据给方法，方法

   需要声明形参接收。

2. 方法内部使用if语句判断，

   并输出对应的结论。

案例 判断整数是奇数还是偶数

```
public class Main {
    // 定义一个方法检查数字是偶数还是奇数，并输出结果
    public static void printOddOrEven(int num) {
        if (num % 2 == 0) {
            System.out.println(num + " is an even number");
        } else {
            System.out.println(num + " is an odd number");
        }
    }

    // 调用该方法的实例
    public static void main(String[] args) {
        printOddOrEven(15);
    }
}
```

练习

- 定义方法，找出两个数(int类型)中的较大数

主观题 5分

设置

- 定义方法，找出两个数(int类型)中的较大数

作答

练习 • 定义方法，找出两个数(int类型)中的较大数

```
public class Main {
    // 定义一个找出两个数中较大数的方法，返回类型是 int
    public static int findMax(int number1, int number2) {
        return number1 > number2 ? number1 : number2;
    }

    // 调用的例子
    public static void main(String[] args) {
        int max = findMax(5, 9);
        System.out.println("较大的数是: " + max);
    }
}
```

目录

Contents

案例 把找出数组的最大值案例，改造成方法，可以支持返回任意整型数组的最大值数据。

```
public class Test {
    public static void main(String[] args) {
        // 1、定义一个静态初始化的数组存储这些数据
      int[] faceScores = {15, 9000, 10000, 20000, 9500, -5};
        // 2、定义一个变量用于保存最大值数据
      int max = faceScores[0];
        // 3、遍历数组中的每个元素
      for (int i = 1; i < faceScores.length; i++) {
        // 4、判断这个元素值是否大于最大值变量中存储的数据，若大，则替换。
        if(faceScores[i] > max){
                    max = faceScores[i];
                }
        }
        // 5、输出最大值变量即可
      System.out.println("数组的元素最大值是：" + max);
    }
}
```

```java
public class Test {
    // 定义一个方法，接受一个整型数组作为参数，返回数组中的最大值。
    public static int getMax(int[] array) {
        // 定义一个变量用于保存最大值数据
        int max = array[0];
        // 遍历数组中的每个元素
        for (int i = 1; i < array.length; i++) {
            // 判断这个元素值是否大于最大值变量中存储的数据，若大，则替换。
            if(array[i] > max){
                max = array[i];
            }
        }
        // 返回最大值
        return max;
    }

    public static void main(String[] args) {
        // 定义一个静态初始化的数组存储这些数据
        int[] faceScores = {15, 9000, 10000, 20000, 9500, -5};
        // 调用getMax方法并传递faceScores数组，打印返回的最大值
        System.out.println("数组的元素最大值是：" + getMax(faceScores));
    }
}
```

总结

1.如何使用方法返回一个整型数组的最大值的?

- **方法需要申明返回值类型：int**
- **方法需要定义形参列表: (int[] arr)**

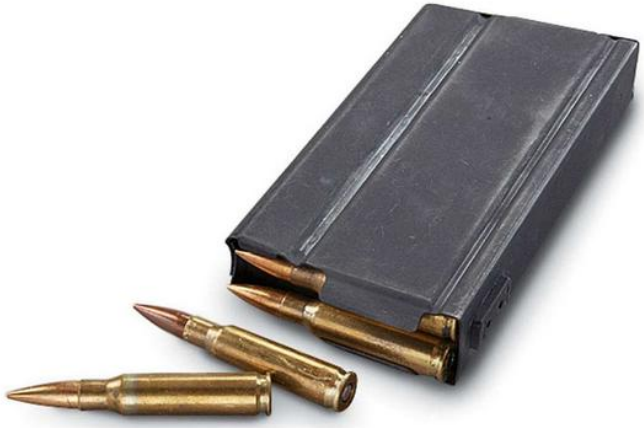
```
public static int max(int[] arr){
    int max = arr[0];
    for (int i = 0; i < arr.length; i++) {
                if(arr[i] > max){
                    max = arr[i];
                }
    }
    return max;
}
```

目录

Contents

方法的调用流程 – 内存图解

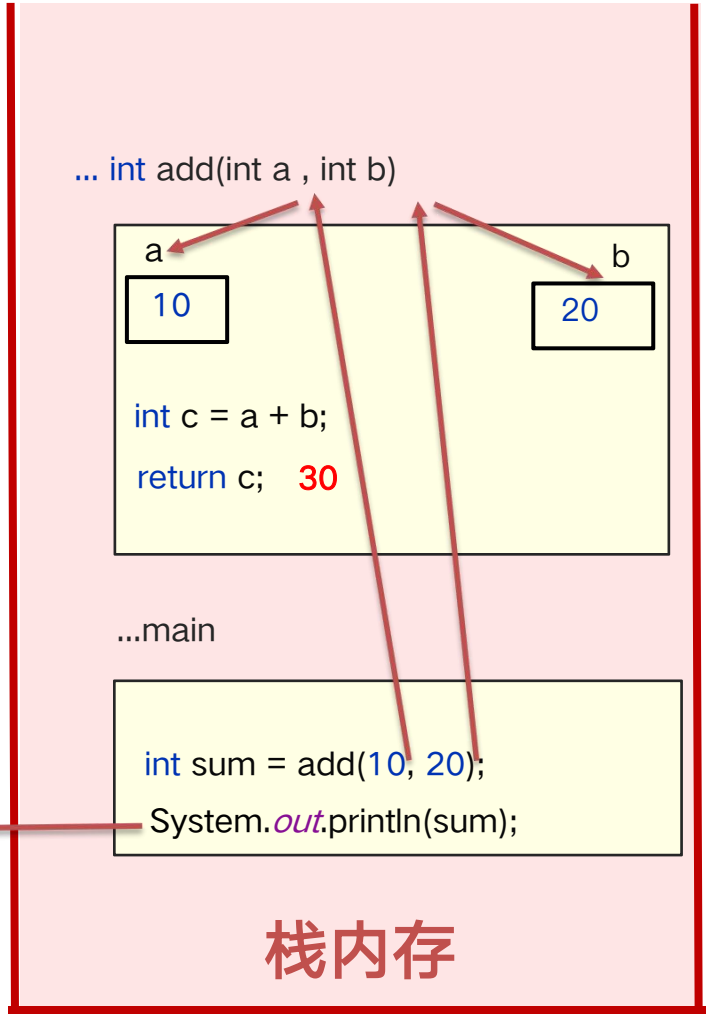
- 方法是放在方法区中的，被调用的时候，需要进入到栈内存中运行



方法的调用流程 – 内存图解

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

```
30
```



方法的调用流程 – 内存图解

```
public class Demo2Method {
    public static void main(String[] args) {
        study();
    }

    public static void sleep(){
        System.out.println("睡觉");
    }

    public static void eat(){
        System.out.println("吃饭");
    }

    public static void study(){

        eat();

        System.out.println("学习");

        sleep();

    }
}
```

```
吃饭
学习
睡觉
```

总结

1. 方法的运行区域在哪里？
   - **栈内存。**

目录

Contents

Java的参数传递机制：值传递

- 在传输实参给方法的形参的时候，并不是传输实参变量本身， 而是传输实参变量中存储的值，这就是值传递。

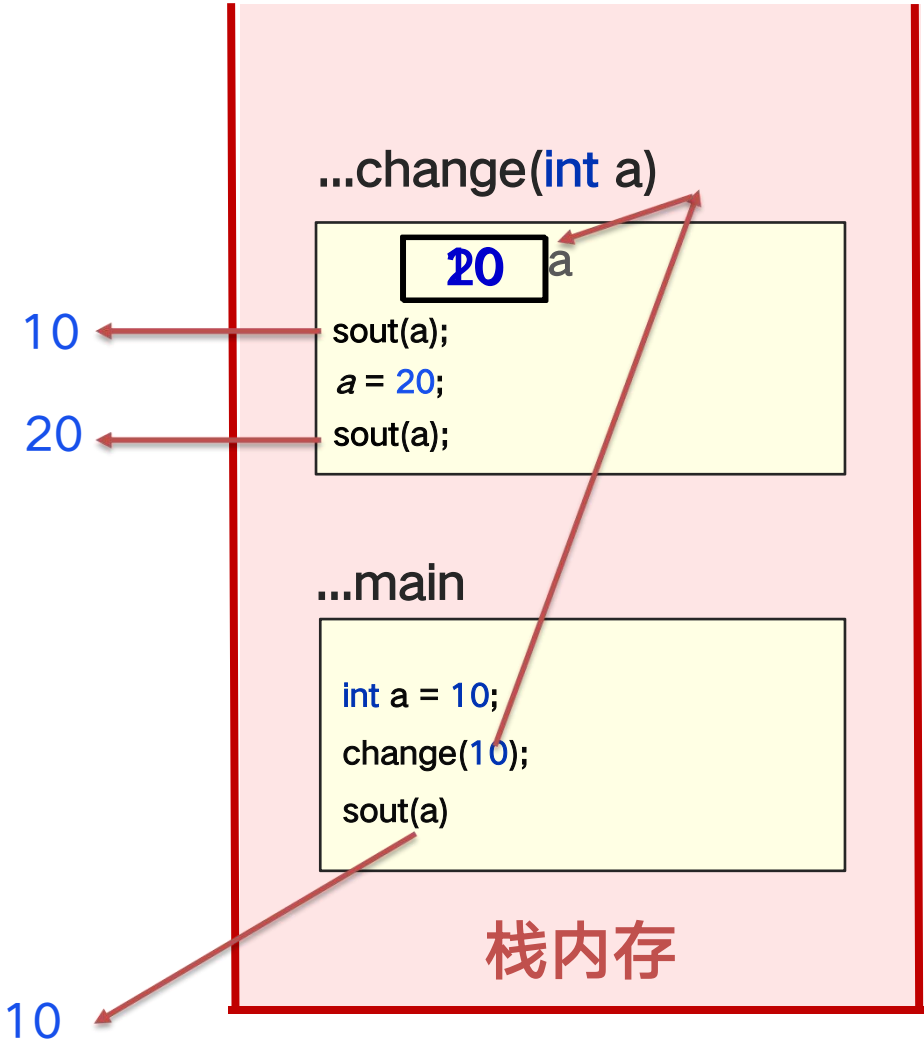
```
public class Test {
    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        change(a);
    }

    public static void change(int c){

    }
}
```

注意：

- 实参：如在方法内部定义的变量。
- 形参：如在定义方法时，“()”中所声明的参数。

基本类型的参数传递

```
public class Test {
    public static void main(String[] args) {
        int a = 10;
        change(a);
        System.out.println(a); // 10
    }

    public static void change(int a){
        System.out.println(a); // 10
        a = 20;
        System.out.println(a); // 20
    }
}
```



总结

1. 形参和实参各指什么？

- **形参：以方法为例，就是方法定义时的变量。**
- **实参：在方法内部定义的变量。**

2. Java的参数传递机制是什么样的？

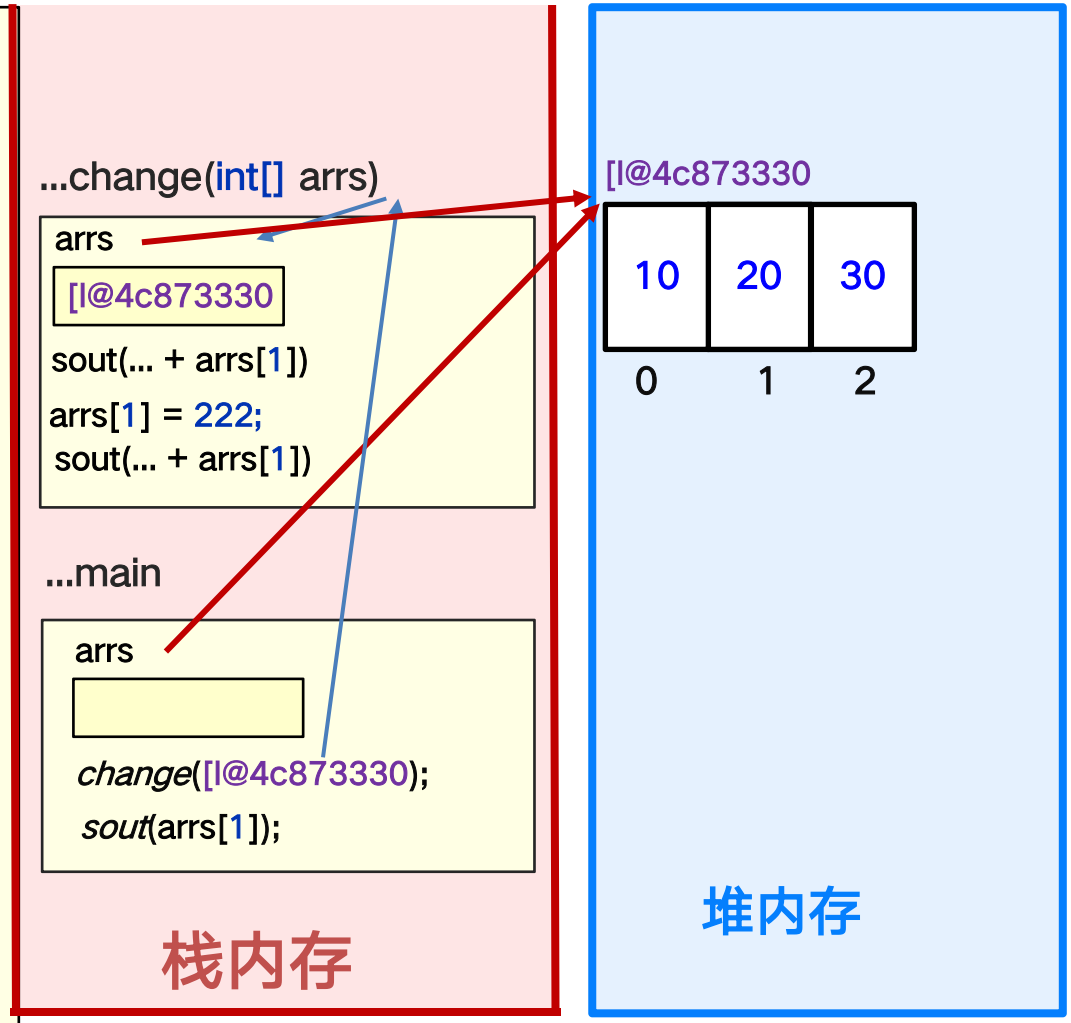
- **值传递，传输的是实参存储的值。**

目录

Contents

引用类型的参数传递

```
public class Test {
    public static void main(String[] args) {
        int[] arrs = new int[]{10, 20, 30};
        change(arrs);
        System.out.println(arrs[1]); // 222
    }

    public static void change(int[] arrs){
System.out.println("方法内部2：  "+arrs[1]);
// 20
        arrs[1] = 222;
System.out.println("方法内部2：  "+arrs[1]);
// 222
    }
}
```



总结

1、基本类型和引用类型的参数在传递的时候有什么不同？

- **都是值传递。**
- **基本类型的参数传输存储的数据值。**
- **引用类型的参数传输存储的地址值。**

填空题 2分

设置

```
public class Test {
    public static void change(int[] arr, int num) {
        arr[0] = 10;
        num = 20;
    }
    public static void main(String[] args) {
        int[] arr = new int[10];
        int num = 5;
        change(arr, num);
        System.out.println(arr[0]); // 输出是？ [填空1]
        System.out.println(num); // 输出是？ [填空2]
    }
}
```

作答

填空题 1分

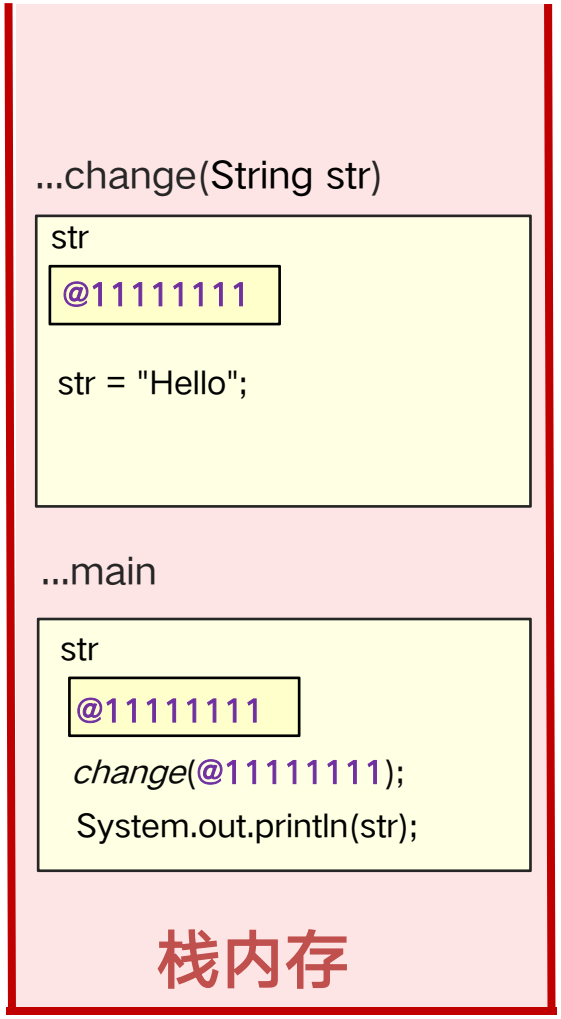
设置

```
public class Test {
    public static void change (String str) {
        str = "Hello";
    }

    public static void main(String[] args) {
        String str = "World";
        change (str);
        System.out.println(str); // 输出是？ [填空1]
    }
}
```

作答

引用类型的参数传递

```
public class Test {
    public static void change (String str) {
        str = "Hello";
    }

    public static void main(String[] args) {
        String str = "World";
        change (str);
        System.out.println(str); // 输出是？ [填空1]
    }
}
```



目录

Contents

方法重载

- 同一个类中，出现多个方法名称相同，但是形参列表不同，那么这些方法就是重载方法。

例子：

- 开发武器系统，功能需求如下：

  ① 可以默认发一枚武器。

  ② 可以指定地区发射一枚武器。

  ③ 可以指定地区发射多枚武器。

方法重载

- 同一个类中，出现多个方法名称相同，但是形参列表不同，那么这些方法就是重载方法。

```
public class Test {
    /**（1）默认发一枚武器。*/
    public static void fire(){
            System.out.println(“默认发射一枚武器给A地区！");
    }
    /**（2）可以指定地区发射一枚武器。*/
    public static void fire(String location){
        System.out.println("给"+location+"发射一枚武器！");
    }
    /**（3）可以指定地区发射多枚武器。*/
    public static void fire(String location , int nums){
        System.out.println("给"+location+"发射"+nums+"枚武器！");
    }
}
```

```
fire();
fire("A地区");
fire("B地区" , 1000);
```

调用方法的时候，
会通过参数的不同来区
分调用的是哪个方法

总结

1. 方法重载是什么样的？

- 同一个类中，多个方法的名称相同，形参列表不同。

2. 使用方法重载的好处 ？

- 对于相似功能的场景：可读性好，方法名称相同提示是同一类型的功能，通过形参不同实现功能差异化的选择。

目录

Contents

练习 • 以下方法是否构成重载

```
public class Test {
    public static void fire(int a){
        //方法体
    }
    public static int fire(int a){
        //方法体
    }
}
```

```
public class Test {
    public static void fire(int a){
        //方法体
    }
    public static int fire(int a, int b){
        //方法体
    }
}
```

```
public class Test1 {
    public static void fire(int a){
        //方法体
    }
}
public class Test2 {
    public static int fire(double a){
        //方法体
    }
}
```

```
public class Test {
    public static void fire(int a){
        //方法体
    }
    public static int fire(double a){
        //方法体
    }
}
```

多选题 1分

设置

- 以下方法哪些构成重载？

```
public class Test {
    public static void fire(int a){
        //方法体
    }
    public static int fire(int a){
        //方法体
    }
}
```

A

```
public class Test {
    public static void fire(int a){
        //方法体
    }
    public static int fire(int a, int b){
        //方法体
    }
}
```

B

```
public class Test1 {
    public static void fire(int a){
        //方法体
    }
}
public class Test2 {
    public static int fire(double a){
        //方法体
    }
}
```

C

```
public class Test {
    public static void fire(int a){
        //方法体
    }
    public static int fire(double a){
        //方法体
    }
}
```

D

提交

方法重载的识别技巧

- 只要是同一个类中，方法名称相同、形参列表不同，那么他们就是重载的方法，其他都不管！（如：修饰符，返回值类型都无所谓）
- 形参列表不同指的是：形参的个数、类型、顺序不同，不关心形参的名称。

练习 · 方法重载

使用方法重载的思想，设计比较两个整数是否相同的（isSame）方法。要求兼容全整数类型（byte，short，int，long）。然后在主方法中调用isSame方法以验证其功能。

实现要求

创建一个包含多个重载版本的isSame方法：

- 比较两个byte类型值
- 比较两个short类型值
- 比较两个int类型值
- 比较两个long类型值

每个方法应返回或者输出布尔值：相等返回true，不等返回false

在main方法中演示调用任意一种情况

主观题 10分

设置

使用方法重载的思想，设计比较两个整数是否相同的（isSame）方法。要求兼容全整数类型（byte，short，int，long）。然后在主方法中调用isSame方法以验证其功能。

实现要求

创建一个包含多个重载版本的isSame方法：

- 比较两个byte类型值
- 比较两个short类型值
- 比较两个int类型值
- 比较两个long类型值

每个方法应返回或者输出布尔值：相等返回true，不等返回false

在main方法中演示调用任意一种情况

作答

练习 · 方法重载

使用方法重载的思想，设计比较两个整数是否相同的（isSame）方法。要求兼容全整数类型（byte，short，int，long）。然后在主方法中调用isSame方法以验证其功能。

```
public class Main {
  public static void isSame(byte a, byte b) {
    System.out.println("byte类型参数执行！");
    System.out.println("结果是：" + (a == b));
  }
  public static void isSame(short a, short b) {
    System.out.println("short类型参数执行！");
    System.out.println("结果是：" + (a == b));
  }
  public static void isSame(int a, int b) {
    System.out.println("int类型参数执行！");
    System.out.println("结果是：" + (a == b));
  }
 public static void isSame(long a, long b) {
    System.out.println("long类型参数执行！");
    System.out.println("结果是：" + (a == b));
  }
}
```

```
isSame(20, 20);
```

```
isSame((byte)20, (byte)20);
```

总结

1. 方法重载的关键要求是什么样的？

- **同一个类中，多个方法的名称相同，形参列表不同，其他无所谓。**

2.形参列表不同指的是什么？

- **形参的个数、类型、顺序不同。不关心形参的名称。**