今日学习目标

* 能够明确方法的作用: 提高代码的复用性
* 能够定义方法:
* 修饰符 返回值类型 方法名(参数类型 参数名...){ 方法体; return 返回值 }
* 能够定义求和案例的方法并使用
* 能够定义获取两个数据中的较大值的方法并使用
* 能够定义比较两个数据是否相等的方法并使用
* 能够定义获取三个数据中的较大值的方法并使用
* 能够定义void修饰的方法: 返回值类型为void的方法就是没有返回值的方法.
* 能够说出方法重载的概念
* 能够定义方法,完成数组遍历
* 能够定义方法,获取数组的最值
* 能够定义方法,实现数组元素求和

# 方法

## 方法概述

假设有一个游戏程序，程序在运行过程中，要不断地发射炮弹(植物大战僵尸)。发射炮弹的动作需要编写100行的代码，在每次实现发射炮弹的地方都需要重复地编写这100行代码，这样程序会变得很臃肿，可读性也非常差。

为了解决代码重复编写的问题，可以将发射炮弹的代码提取出来放在一个{}中，并为这段代码起个名字，这样在每次发射炮弹的地方通过这个名字来调用发射炮弹的代码就可以了。

上述过程中，所提取出来的代码可以被看作是程序中定义的一个方法，程序在需要发射炮弹时调用该方法即可。

## 定义格式

* 简单的说：方法就是完成特定功能的代码块
  + 在很多语言里面都有函数的定义
  + 函数在Java中被称为方法
* 格式

修饰符 返回值类型 方法名(参数类型 参数名1，参数类型 参数名2…) {

方法体;

return 返回值;

}

* 格式解释
  + 修饰符目前记住 public static
  + 返回值类型用于限定返回值的数据类型
  + 方法名一个名字，为了方便我们调用方法
  + 参数类型用于接收调用方法时传入的数据的类型
  + 参数名用于接收调用方法时传入的数据的变量
  + 方法体完成功能的代码
  + return 结束方法，把返回值带给调用者
* 写一个方法首先有两点需要明确
  + 返回值类型明确功能结果的数据类型
  + 参数列表明确有几个参数，以及参数的类型
* 注意事项:方法定义完成之后,没有调用没有意义,不调用不执行。

## 求和案例

* 求和方法编写

**public class** Demo01Method {

/\*

\* 写一个方法，用于求和。

\* 两个明确：返回值类型int参数列表int a,int b

\*/

**public static int** sum(**int** a,**int** b){

/\*int c = a + b;

return c;\*/

**return** a + b;

}

}

* 求和方法的调用(有明确返回值)
  + 使用格式：方法名(参数);
  + 单独调用，没有意义

sum(10,20);

* + 输出调用，有意义，但是不够好，因为我不一定非要把结果输出

System.out.println(sum(10,20));

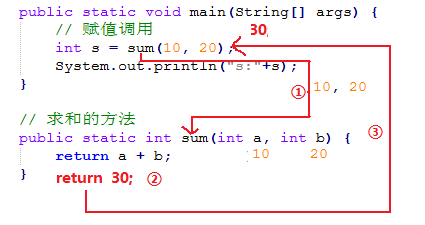
* + 赋值调用，推荐方式

int s = sum(10, 20);

// s+=100;

System.out.println("s:"+s);

* 求和方法的调用图解



## 练习

* 获取两个数据中的较大值

**public class** Demo03MethodTest {

**public static void** main(String[] args) {

//创建对象

Scanner sc = **new** Scanner(System.***in***);

//接收数据

System.***out***.println("请输入第一个数据：");

**int** x = sc.nextInt();

System.***out***.println("请输入第二个数据：");

**int** y = sc.nextInt();

//调用方法

**int**max = *getMax*(x,y);

System.***out***.println("max:"+max);

}

// 返回两个数中的较大值

**public static int** getMax(**int** a, **int** b) {

**if** (a > b) {

**return** a;

} **else** {

**return** b;

}

}

}

* 比较两个数据是否相等

**public class** Demo04MethodTest {

**public static void** main(String[] args) {

//创建对象

Scanner sc = **new** Scanner(System.***in***);

//接收数据

System.***out***.println("请输入第一个数据:");

**int** a = sc.nextInt();

System.***out***.println("请输入第二个数据:");

**int** b = sc.nextInt();

//调用方法

**boolean** flag = *compare*(a,b);

System.***out***.println("flag:"+flag);

}

//比较两个数是否相等

**public static boolean** compare(**int**a,**int**b){

**if**(a==b){

**return true**;

}**else** {

**return false**;

}

}

}

* 获取三个数据中的较大值

**public class** Demo05MethodTest {

**public static void** main(String[] args) {

//创建对象

Scanner sc = **new** Scanner(System.***in***);

//接收数据

System.***out***.println("请输入第一个数据：");

**int** a = sc.nextInt();

System.***out***.println("请输入第二个数据：");

**int** b = sc.nextInt();

System.***out***.println("请输入第三个数据：");

**int** c = sc.nextInt();

//调用方法

**int** max = *getMax*(a, b, c);

System.***out***.println("max:"+max);

}

// 返回三个数中的最大值

**public static int** getMax(**int** a, **int** b, **int** c) {

**if** (a>b) {

**if** (a>c) {

r**eturn** a;

} **else** {

**return** c;

}

} **else** {

**if** (b>c) {

**return** b;

} **else** {

**return** c;

}

}

}

}

## void修饰的方法

* 格式

修饰符 void 方法名(参数类型 参数名1，参数类型 参数名2…) {

方法体;

return ;

}

* 没有明确返回值的函数调用:
  + 其实就是void类型方法的调用
  + 只能单独调用
* 案例

写一个方法，在控制台输出10次HelloWorld案例。

**public class** Demo01Method {

**public static void** main(String[] args) {

//单独调用

*printHelloWorld*();

}

//在控制台输出10次HelloWorld案例。

**public static void** printHelloWorld() {

**for**(**int** x=1; x<=10; x++) {

System.***out***.println("HelloWorld");

}

}

}

## 练习

* 定义一个方法，传递一个整数(大于1)，在控制台打印1到该数据的值。

**public class** Demo02Method {

**public static void** main(String[] args) {

*printNumber*(10);

System.***out***.println("-------------------");

*printNumber*(100);

}

//在控制台打印1到该数据n的值

**public static void** printNumber(**int** n) {

**for**(**int** x=1; x<=n; x++) {

System.***out***.println(x);

}

}

}

* 定义一个方法，把所有的水仙花数打印在控制台

**public class** Demo03Method {

**public static void** main(String[] args) {

*printFlower*();

}

//把所有的水仙花数打印在控制台

**public static void** printFlower() {

**for**(**int** x=100; x<1000; x++) {

**int** ge = x%10;

**int** shi = x/10%10;

**int** bai = x/10/10%10;

**if**((ge\*ge\*ge+shi\*shi\*shi+bai\*bai\*bai) == x){

System.***out***.println(x);

}

}

}

}

## 方法重载（Overload）

* 方法重载概述
  + 在同一个类中，允许存在一个以上的同名方法，只要它们的参数个数或者参数类型不同即可。
* 方法重载特点
  + 与返回值类型无关，只看方法名和参数列表
  + 在调用时，虚拟机通过参数列表的不同来区分同名方法
* 基本使用

**public class** Demo01Method {

**public static void** main(String[] args) {

**int** a = 10;

**int** b = 20;

// 求和

**int** result = *sum*(a, b);

System.***out***.println("result:" + result);

**int** c = 30;

// 求和

result = *sum*(a,b,c);

System.***out***.println("result:"+result);

}

//两个float类型的数据求和

**public static float** sum(**float** a,**float** b) {

**return** a + b;

}

// 三个整数的求和

**public static int** sum(**int** a,**int** b,**int** c) {

**return** a + b + c;

}

// 两个整数的求和

**public static int** sum(**int** a, **int** b) {

**return** a + b;

}

}

* 练习
  + 比较两个数据是否相等。参数类型分别为两个byte类型，两个short类型，两个int类型，两个long类型，并在main方法中进行测试

**public class** Demo02MethodTest {

**public static void** main(String[] args) {

// 调用

System.***out***.println(*compare*(10, 20));//整数默认为int类型

System.***out***.println("-------------");

System.***out***.println(*compare*((**byte**)10, (**byte**)20));

System.***out***.println("-------------");

System.***out***.println(*compare*((**short**)10, (**short**)20));

System.***out***.println("-------------");

//System.out.println(compare((long)10, (long)20));

System.***out***.println(*compare*(10L, 20L));

}

// 两个byte类型的

**public static boolean** compare(**byte** a, **byte** b) {

System.***out***.println("byte");

// 第一种写法

// boolean flag = a==b?true:false;

// return flag;

// 第二种写法

// boolean flag = a == b;

// return flag;

// 第三种写法

**return** a == b;

}

// 两个short类型的

**public static boolean** compare(**short** a, **short** b) {

System.***out***.println("short");

**return** a == b;

}

// 两个int类型的

**public static boolean** compare(**int** a, **int** b) {

System.***out***.println("int");

**return** a == b;

}

// 两个long类型的

**public static boolean** compare(**long** a, **long** b) {

System.***out***.println("long");

**return** a == b;

}

}

## 方法参数传递问题

* 参数传递概述

可以理解当我们要调用一个方法时，我们会把指定的数值，传递给方法中的参数，这样方法中的参数就拥有了这个指定的值，可以使用该值，在方法中运算了。

这种传递方式，我们称为参数传递。

* + 在这里，定义方法时，参数列表中的变量，我们称为形式参数
  + 调用方法时，传入给方法的数值，我们称为实际参数
* 参数是基本类型及图解
  + 形式参数的改变不影响实际参数。

**public class** Demo01Args {

**public static void** main(String[] args) {

// 定义变量

**int** a = 10;

**int** b = 20;

System.***out***.println("a:" + a + ",b:" + b);// a:10,b:20

*change*(a, b);

System.***out***.println("a:" + a + ",b:" + b);// a:10,b:20

}

**public static void** change(**int** a, **int** b) { // a=10,b=20

System.***out***.println("a:" + a + ",b:" + b);// a:10,b:20

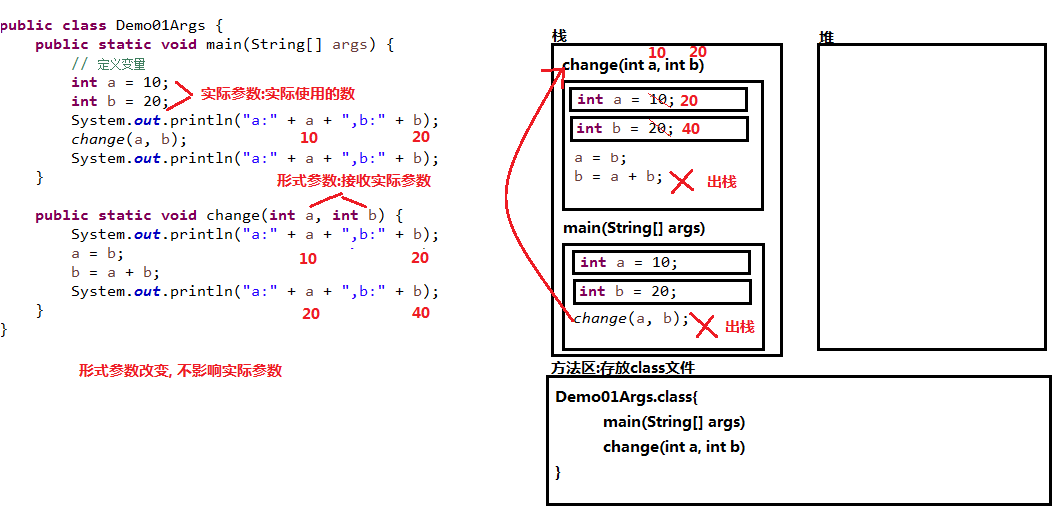
a = b; // a=20;

b = a + b;// b=40;

System.***out***.println("a:" + a + ",b:" + b);// a:20,b:40

}

}



* 参数是引用类型及图解

**public class** Demo02Args {

**public static void** main(String[] args) {

// 定义数组

**int**[] arr = { 1, 2, 3, 4, 5 };

// 遍历数组

**for** (**int** x = 0; x<arr.length; x++) {

System.***out***.println(arr[x]);

}

System.***out***.println("----------------");

*change*(arr);

**for** (**int** x = 0; x<arr.length; x++) {

System.***out***.println(arr[x]);

}

}

**public static void** change(**int**[] arr) {

**for** (**int** x = 0; x < arr.length; x++) {

// 如果元素是偶数，值就变为以前的2倍

**if** (arr[x] % 2 == 0) {

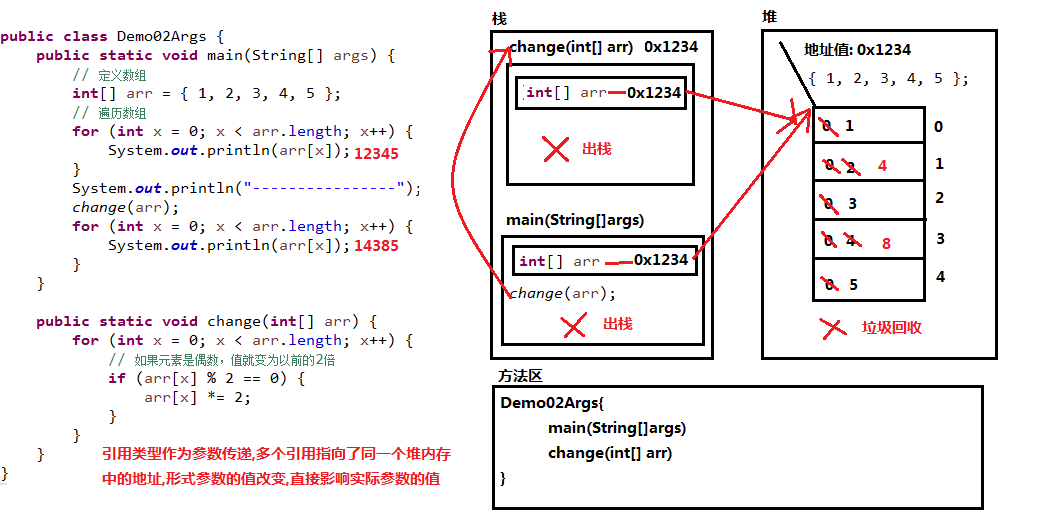
arr[x] \*= 2;

}

}

}

}



## 方法练习

* 定义方法,完成数组遍历

**public class** Demo01MethodTest {

**public static void** main(String[] args) {

// 定义数组

**int**[] arr = { 11, 22, 33, 44, 55 };

*printArray1*(arr);

*printArray2*(arr);

}

/\*

\* 按照以下格式遍历数组

\* [元素1, 元素2, 元素3, ...]

\*/

**public static void** printArray2(**int**[] arr) {

System.***out***.print("[");

**for**(**int**x=0; x<arr.length; x++){

**if**(x==arr.length-1){

System.***out***.println(arr[x]+"]");

}**else** {

System.***out***.print(arr[x]+", ");

}

}

}

/\*

\* 定义遍历数组的方法

\* 两个明确：

\* 返回值类型：void

\* 参数列表：int[] arr

\*/

**public static void** printArray1(**int**[] arr) {

**for**(**int** x=0; x<arr.length; x++){

System.***out***.println(arr[x]);

}

}

}

* 定义方法,获取数组的最值

**public class** Demo02MethodTest {

**public static void** main(String[] args) {

// 定义数组

**int**[] arr = { 34, 67, 10, 28, 59 };

//调用获取数组中的最大值的方法

**int** max = *getMax*(arr);

System.***out***.println("max:"+max);

//调用获取数组中的最小值的方法

**int** min = *getMin*(arr);

System.***out***.println("min:"+min);

}

//获取数组中的最小值的方法

**public static int** getMin(**int**[] arr) {

**int** min = arr[0];

**for**(**int** i=1; i<arr.length; i++) {

**if**(arr[i] <min) {

min = arr[i];

}

}

**return** min;

}

/\*

\* 获取数组中的最大值的方法

\* 两个明确：

\* 返回值类型：int

\* 参数列表：int[] arr

\*/

**public static int** getMax(**int**[] arr) {

**int** max = arr[0];

**for**(**int** i=1; i<arr.length; i++) {

**if**(arr[i] >max) {

max = arr[i];

}

}

**return** max;

}

}

* 定义方法,实现数组元素求和

**public class** Demo03MethodTest {

**public static void** main(String[] args) {

// 定义数组

**int**[] arr = { 1, 2, 3, 4, 5 };

//调用数组元素求和方法

**int** sum = *sum*(arr);

System.***out***.println("sum:"+sum);

}

/\*

\* 定义数组元素求和方法

\* 两个明确：

\* 返回值类型：int

\* 参数列表：int[] arr

\*/

**public static int** sum(**int**[] arr) {

**int** sum = 0;

**for**(**int** i=0; i<arr.length; i++) {

sum += arr[i];

}

**return** sum;

}

}