讲义04：类与对象

主要内容

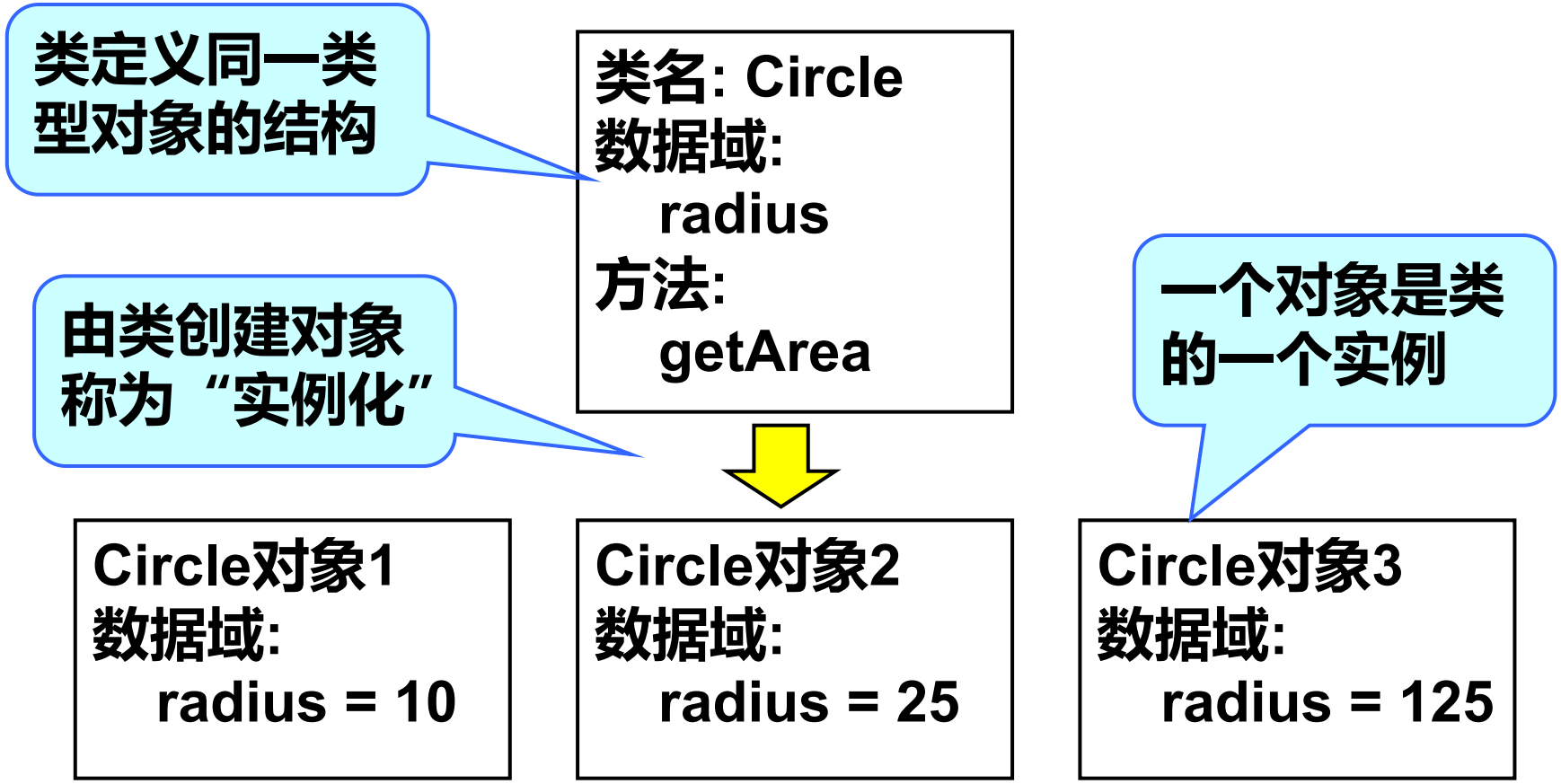
- 讲授内容
  - 面向对象的特性
  - 类
  - 构造方法与对象的创建
  - 参数传递
  - 对象的组合
  - 实例成员与类成员
  - 方法重载、覆盖与多态
  - this、包、import、访问权限
- 内容资料来源
  - 教材第6章、第7章(7.3节)
  - 其他资料补充

1 类和对象的基本概念(6.1)

- 对象(Object)代表现实世界中可以明确标识的一个整体.
  - 对象的状态(State)由具有当前值的数据域(data field)的集合组成.
  - 对象的行为(Behavior)是方法(method)的集合定义的.

- 类(Class)定义了同一类型的对象。它定义了该类对象的数据域和方法。

1 类和对象的基本概念


类定义同一类型对象的结构
类名: Circle
数据域:
radius
方法:
getArea
由类创建对象称为“实例化”
一个对象是类的一个实例
Circle对象1
数据域:
radius = 10
Circle对象2
数据域:
radius = 25
Circle对象3
数据域:
radius = 125

1 类和对象的基本概念

- 封装性：通过抽象找出具体实例中的共同特征(数据和操作)，将数据和对数据的操作封装在一起。
- 继承性：子类可以继承父类的数据及数据操作，同时又可以扩展子类独有的数据及数据操作。
- 多态性：
  - 基于重载实现的静态多态：多个操作具有相同名称，但是接收不同的消息类型。
  - 基于继承和覆盖实现的动态多态：同一操作被不同类型的对象所调用产生不同的行为。

2 类的定义 - Java类的结构(6.2)

```
class 类名 {
    数据域 – 成员变量
    操作 – 方法
    创建对象 – 构造方法
}
```

```
class Circle {
  /** 圆的半径 */
  double radius = 1.0;
  /** 构造一个圆对象 */
  Circle() {
  }
  /**构造一个圆对象 */
  Circle(double newRadius) {
    radius = newRadius;
  }
  /** 计算圆的面积 */
  double getArea() {
    return Math.PI * radius * radius;
  }
}
```

2 类的定义 - Java类的结构

(1) 类的命名

- 类名通常使用名词性英文单词或词组。
- 类名使用小写字母, 每个单词的首字母大写。

例如：Circle, Scanner, FileInputStream

(2) 由类创建对象

使用构造方法, 下一小节详细说明

2 类的定义 - Java类的结构

(3) 类的数据域

- 数据域(data field)也称为成员变量。
- 数据域用来描述类的属性，抽象过程中要抓住本质属性。
- 数据域变量可以是Java的任何一种基本数据类型或类。
- 数据域变量命名通常使用名词性单词或词组，当由多个单词组成时，从第2个单词开始首字母大写。

数据域的有效范围

- 数据域在类中可以按任何顺序声明，其作用域为整个类.
- 例外情况是被其他数据域引用的数据域必须先声明.

```
public class Foo {
  private int i;
  private int j = i + 1; //j必须在i之后定义
}
```

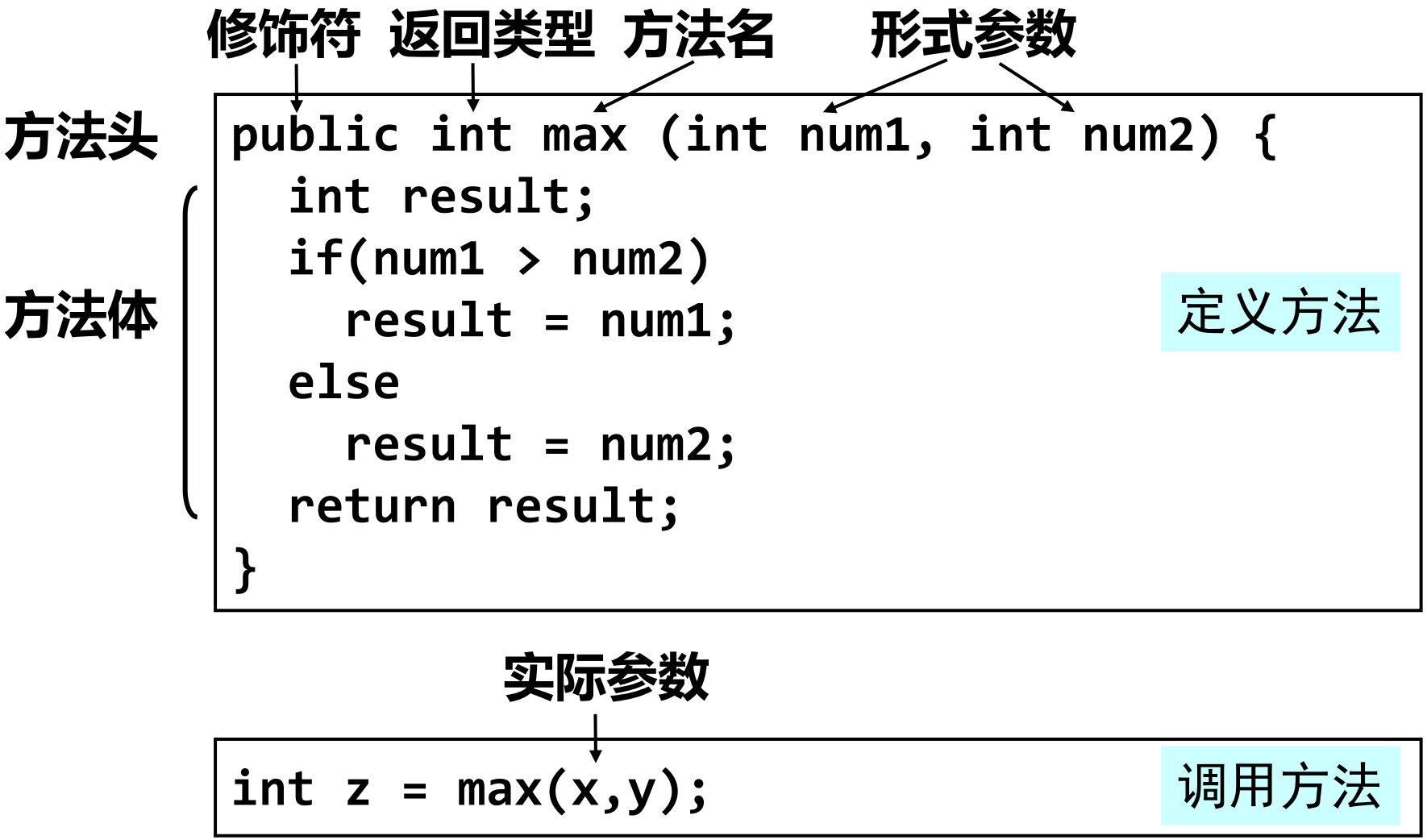
2 类的定义 - Java类的结构

(4) 类的方法

- Java中没有独立的方法，方法必须定义在类的内部。
- 方法用于说明对类的某些数据域的操作。
- 方法命名通常使用动词或动词性词组，当由多个单词组成时，从第2个单词开始首字母大写。

```
修饰符  返回值类型  方法名(参数列表)  {
   //方法体
}
```

2 类的定义 - Java类的结构



2 类的定义 - Java类的结构

(5) 局部变量与成员变量

局部变量：定义在方法中，从该变量的说明开始到包含该变量的块体结束为止。

不嵌套的不同块中，
允许同名局部变量.

```
public static void m () {
  int x = 1, y = 1;
  for(int i = 1; i < 10; i++) {
    x += i;
  }
  for(int i = 1; i < 10; i++) {
    y += i;
  }
}
```

嵌套的不同块中，
不允许同名局部变量.

```
public static void m () {
  int i = 1;
  int sum = 0;
  for(int i = 1; i < 10; i++) {
    sum += i;
  }
}
```

2 类的定义 - Java类的结构

(5) 局部变量与成员变量

成员变量：定义在类中，局部变量与成员变量同名时，在方法中局部变量优先。

```
class Foo {
  int x = 0; //成员变量
  int y = 0;
  Foo() { }
  void p() {
    int x = 1; //局部变量
    System.out.println("x = " + x);
    System.out.println("y = " + y);
  }
}
```

设: f = new Foo();
则: f.p()的输出是什么?

结果为: x = 1
y = 0

2 类的定义 - Java类的结构

不太好的类设计方案

area不是一个好的数据域，它可以由其他数据域计算得到。

```
package lader.bad;
public class Lader {
    float above;  //梯形上底
    float bottom; //梯形下底
    float height; //梯形的高
    float area;   //梯形的面积
    float getArea() {
        area = (above + bottom) * height / 2.0F;
        return area;
    }
    void setHeight(float newHeight) {
        height = newHeight;
    }
}
```

代码：examples/lecture03/ClassDefineDemo

2 类的定义 - Java类的结构

改进的类设计方案

```
package lader.good;
public class Lader {
    float above; // 梯形上底
    float bottom; // 梯形下底
    float height; // 梯形的高

    float getArea() {
        return (above + bottom) * height / 2.0F;
    }

    void setHeight(float newHeight) {
        height = newHeight;
    }
}
```

3 构造方法与对象的创建及使用(6.3 & 7.3)

- 构造方法是一种特殊方法, 创建对象时调用:
  - 构造方法必须与所在的类有相同的名字.
  - 构造方法必须没有返回类型, 连void也没有.
- 构造方法的主要作用是**对数据域变量进行初始化**。
- 当类中没有明确声明构造方法时, 隐含声明一个方法体为空的无参构造方法,称为“**默认构造方法**”.
- 一个类可以有多个构造方法, 通过**方法重载**实现, 即多个构造方法的**形式参数的个数或类型必须不同**.
- 构造方法的调用语法: new 构造方法名(参数);

3 构造方法与对象的创建

以类Lader为例说明构造方法与创建对象

```
package lader.good;
public class Lader {
    float above;  //梯形上底
    float bottom; //梯形下底
    float height; //梯形的高
    float getArea() {
        return (above + bottom) * height / 2.0F;
    }
    void setHeight(float newHeight) {
        height = newHeight;
    }
}
```

注意：Lader类中没有定义构造方法，此时可以使用默认构造方法。

代码：examples/lecture03/ObjectCreateAndReference

3 构造方法与对象的创建及使用

(1) 声明对象引用变量

语法: 类名 引用变量名;

示例: Lader lader;

(2) 使用默认构造方法创建对象并分配给引用变量

语法: 引用变量名 = new 构造方法(实际参数);

示例: lader = new Lader();

3 构造方法与对象的创建及使用

(3) 增加自定义构造方法使用创建对象更方便

在类Lader中增加2个构造方法

```
Lader() {  //如果不增加这个无参构造方法，会有什么问题?
}
Lader(float a, float b, float h) {
    above = a;
    bottom = b;
    height = h;
}
```

可以用2种方式创建对象:

```
Lader first = new Lader(10, 20, 5);
Lader second = new Lader();
```

3 构造方法与对象的创建及使用

(4) 对象的内存模型

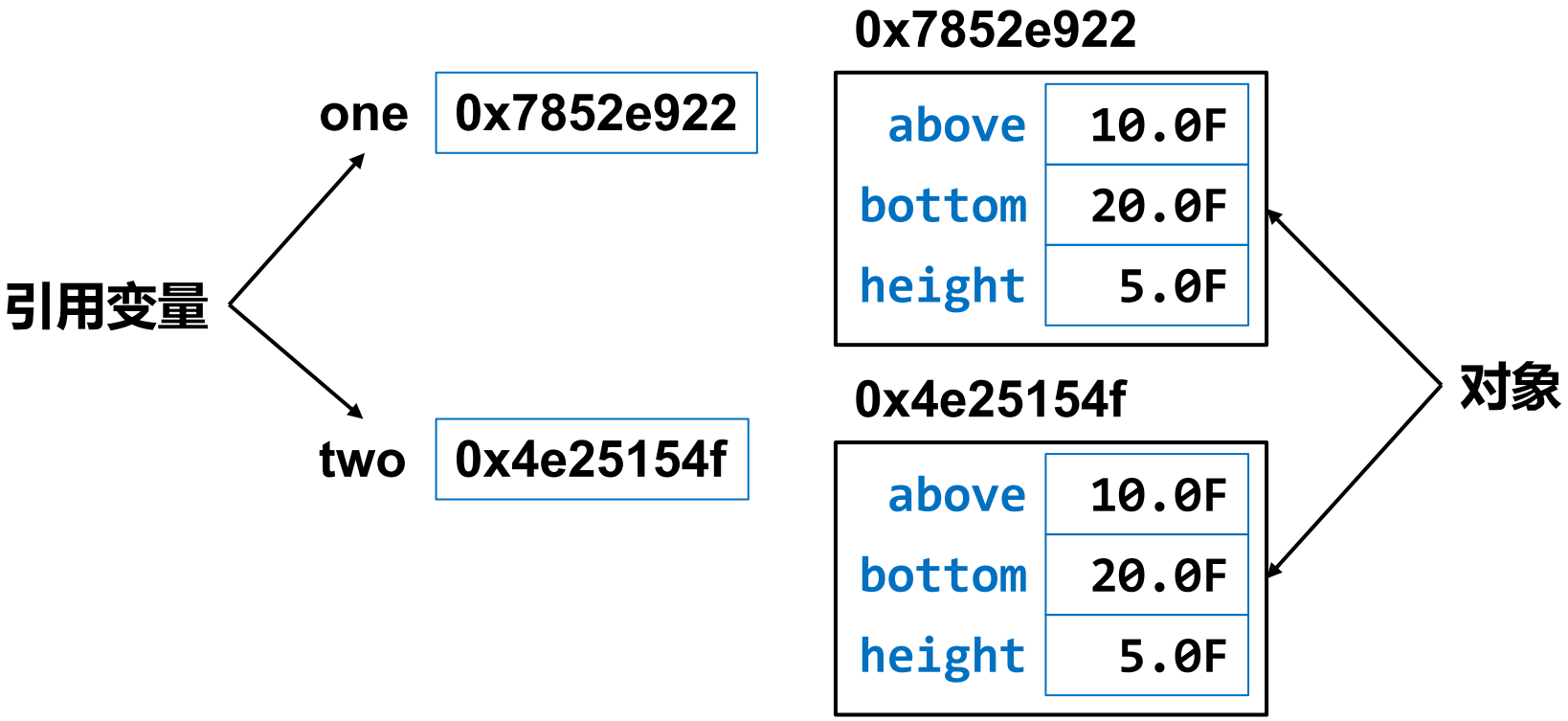
- 对象引用变量用于存放其引用对象的地址
- 对象创建过程
  - 第1步，为对象分配内存空间，包括其所有所有数据域变量。如果数据域没有进行初始化，默认规则：引用型为null；数值型为0；boolean型为false；char型为'\u0000'。
  - 第2步，调用构造方法，构造方法可以对数据域变量进行必要的二次初始化。

3 构造方法与对象的创建及使用

(4) 对象的内存模型：示例及内存分析

例如：

```
Lader one = new Lader(10, 20, 5);
Lader two = new Lader(20, 30, 15);
```



3 构造方法与对象的创建及使用

(5) 对象的使用

使用成员运算符“.”访问对象的数据和方法：

- 对象引用变量.data ------ 访问对象的数据
- 对象引用变量.method() ------ 访问对象的方法

示例：

```
float AreaOfOne = one.getArea();
two.height = 10.0F;
float AreaOfTwo = two.getArea();
one.setHeight(20.0F);
```

3 构造方法与对象的创建及使用

一个完整的示例，计算梯形面积。

```java
//Lader.java
package lader.good;

public class Lader {
    float above;  // 梯形上底
    float bottom; // 梯形下底
    float height; // 梯形的高

    // 无参构造方法
    Lader() {
    }

    // 有参构造方法
    Lader(float a, float b, float h) {
        above = a;
        bottom = b;
        height = h;
    }

    float getArea() {
        return (above + bottom) * height / 2.0F;
    }

    void setHeight(float newHeight) {
        height = newHeight;
    }
}
```

```java
//Main.java
package lader.good;

public class Main {

    public static void main(String[] args) {
        Lader one = new Lader(10, 20, 5);
        System.out.println("梯形one的面积：" + one.getArea());
        one.height = 10.0F; // 直接修改梯形one的高度
        System.out.println("梯形one的面积：" + one.getArea());
        one.setHeight(20.0F); // 调用one的方法修改梯形one的高度
        System.out.println("梯形one的面积：" + one.getArea());
    }

}
```

代码：examples/lecture03/ObjectCreateAndReference

3 构造方法与对象的创建及使用

(6) 对象的使用中的null

对象引用变量被赋值为null时，表示没有引用任何对象实体。通常称为“空对象”或“空引用”。使用“空对象”会导致程序出现 NullPointerException。

Java编译器不会检查“空对象”，需要自行编码避免使用。

示例：

```
if (one != null) {
    one.setHeight(20.0F);
}
```

3 构造方法与对象的创建及使用

(6) Java的垃圾回收机制

Java的 “垃圾回收机制” 周期性的检测是否有 “对象实体” 不再被任何引用变量 “引用”(称为 “垃圾对象”)。

Java会定期回收 “垃圾对象”, 注意不是随时回收。

可以在程序中使用如下方法主动回收, **但是不建议这样做**。

```
System.gc();
```

补充：如果希望深入了解Java的垃圾回收机制，可以阅读《深入理解Java虚拟机》的相关章节。

4 方法的参数传递(6.4)

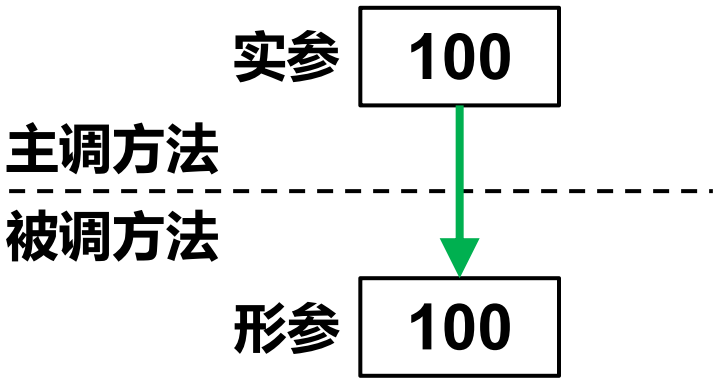
参数传递是使用方法时需要重点理解的问题之一。

Java语言中, 所有参数都是 “**值传递**”, 即发生方法调用时, 方法调用语句中的 “实际参数” 把其值 “单向传递” 给 “形式参数”。

方法中**修改形式参数的值**, 对 “实际参数” 的值没有影响。

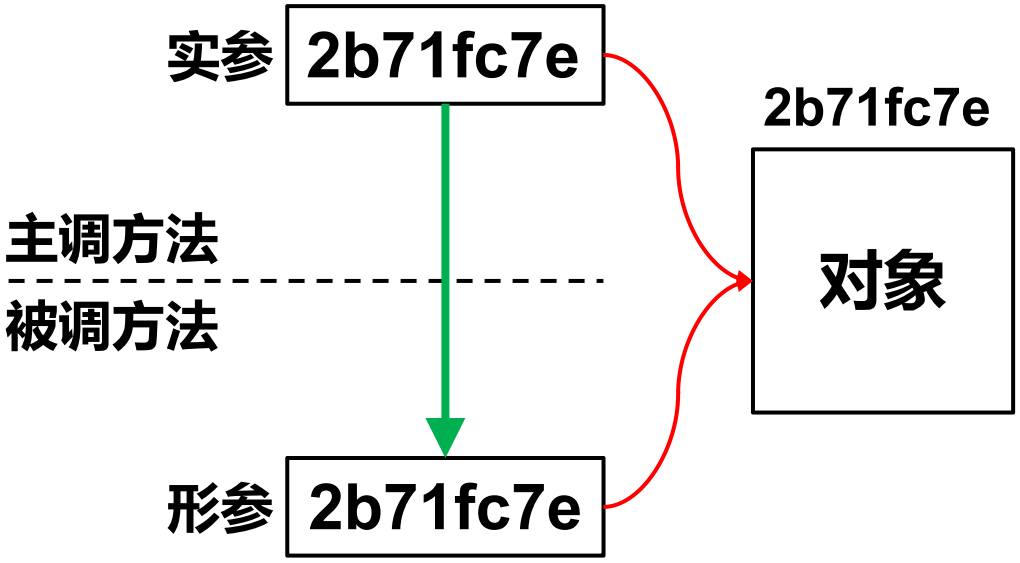
4 方法的参数传递

基本类型的参数传递



基本数据类型的实参向形参传递时，可以进行自动类型转换。
实参将自己的值赋给形参。

引用类型的参数传递



引用类型的实参向形参传递时，实参将自己的值(某个对象的地址)赋给形参。
实参和形参引用同一个对象实体。

表示参数传递方向

表示引用对象

4 方法的参数传递

(1) 基本类型的参数传递示例

完整代码: examples/lecture04/PassParameter

```
public void swapInteger(int a, int b) {
    // 交换形参a和b的值
    int temp = a;
    a = b;
    b = temp;
}
```

定义 int x = 10, y = 100;
调用 swapInteger(x, y) 后, 变量x和y的值有没有交换?

4 方法的参数传递

(2) 引用类型的参数传递示例

完整代码: examples/lecture04/PassParameter

```
void swapCircle(Circle c1, Circle c2) {
    // 交换形参c1和c2的值
    Circle temp = c1;
    c1 = c2;
    c2 = temp;
}
```

执行如下代码后first和second的值有无交换?

```
Circle first = new Circle(10.0);
Circle second = new Circle(20.0);
demo.swapCircle(first, second);
System.out.println(first + " , " + second);
System.out.println(first.radius + " , " + second.radius);
```

4 方法的参数传递

(3) 引用类型的参数传递示例

完整代码: examples/lecture04/PassParameter

```
public void swapRadiusOfCircle(Circle c1, Circle c2) {
    // 交换形参c1和c2的radius
    double temp = c1.radius;
    c1.radius = c2.radius;
    c2.radius = temp;
}
```

执行如下代码后third和fourth的值有无交换?

```
Circle third= new Circle(10.0);
Circle fourth = new Circle(20.0);
demo.swapRadiusOfCircle(third, fourth);
System.out.println(third+ " , " + fourth);
System.out.println(third.radius + " , " + fourth.radius);
```

4 方法的参数传递 - 可变参数

Java5引入方法的可变参数，即允许方法接受个数不定的参数。

```
返回类型 方法名(固定参数列表，数据类型... 可变参数名){
    方法体
}
```

说明：

- 可变参数必须位于形参的最后一项
- 可变参数在方法中以数组形式调用

4 方法的参数传递 - 可变参数

一个变长参数的例子: examples/lecture04/VariableLengthParameter

```
int max(int... numbers) {
    int result = 0;
    for (int x : numbers) {
        if (x > result) {
            result = x;
        }
    }
    return result;
}
```

下面是几个调用的示例:

```
Main demo = new Main();
System.out.println(demo.max(22, 3, 44, 5));
System.out.println(demo.max(22, 3));
System.out.println(demo.max());
```

课后工作

- 结合教材内容，复习课堂讲授内容
- 完成作业网站：随堂实验1