

第3章 面向对象（上）

老师：李沁洳



/3.1

面向对象的思想



3.1 面向对象的思想

对象是人们要进行研究的任何事物，从最简单的整数到复杂的飞机等均可看作对象，它不仅能表示具体的事物，还能表示抽象的规则、计划或事件。

- 对象具有状态，一个对象用数据值来描述它的状态。Java通过为对象定义Field（以前常被称为属性，现在也称为字段）来描述对象的状态；对象还有操作，这些操作可以改变对象的状态，对象的操作也被称为对象的行为，Java通过为对象定义方法来描述对象的行为。
- 对象实现了数据和操作的结合，使数据和操作封装于对象的统一体中。

3.1 面向对象的思想

面向对象介绍

- 面向：拿，找
- 对象：能干活的东西
- 面向对象编程：拿东西过来做对应的事情

3.1 面向对象的思想

洗衣服



打电话



扫地



3.1 面向对象的思想

面向对象到底学什么？

学习获取已有对象并使用

学习如何自己设计对象并使用

/3.2

类与对象

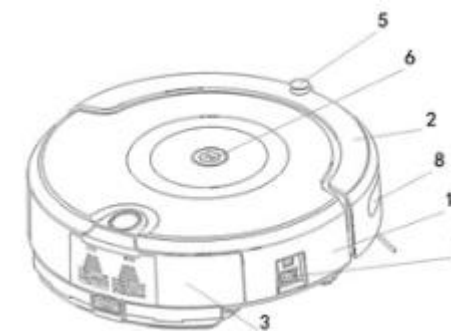
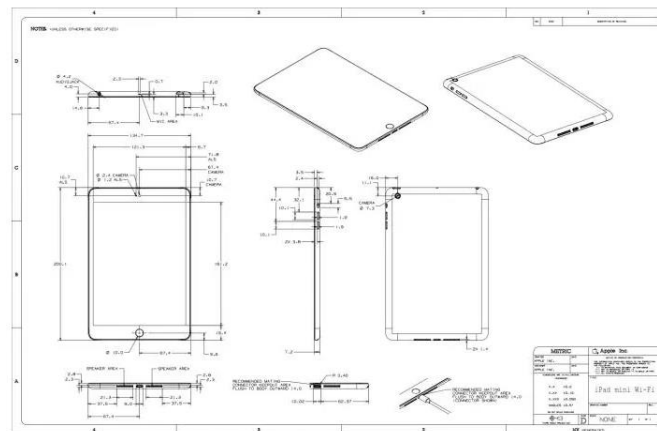
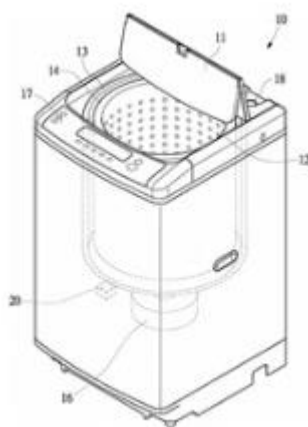


3.2 类和对象

对象:



设计图 (类)



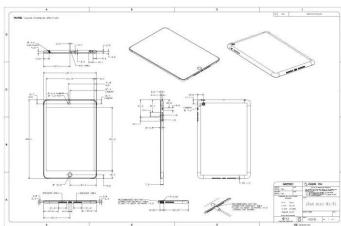
3.2 类和对象

类（设计图）：是对象共同特征的描述；

对象：是真实存在的具体东西，可以帮助我们解决问题。

在Java中，必须先设计类，才能获取对象

```
public class Phone {  
  
}
```



```
Phone phone1 = new Phone();
```

```
Phone phone2 = new Phone();
```



3.2.1 类的定义与创建和使用

如何定义类:

```
public class 类名{
```

1. 成员变量（代表属性，一般是名词）
2. 成员方法（代表行为，一般是动词）
3. 构造器（后面学习）
4. 代码块（后面学习）
5. 内部类（后面学习）

```
}
```

```
public class Phone {  
    //属性（成员变量）  
    String brand;  
    double price;  
  
    //行为（方法）  
    public void call(){  
  
    }  
  
    public void playGame(){  
  
    }  
}
```

如何得到类的对象

```
类名 对象名 = new 类名();
```

```
Phone phone1 = new Phone();
```

如何使用对象

访问属性：对象名.成员变量

访问行为：对象名.方法名(...)

3.2.1 类的定义与创建和使用

成员变量

- 定义在类中。
- 成员变量命名采用小驼峰命名法，最好由有意义的英文单词或单词缩写组成。
- 成员变量的作用域为整个类或实例。

局部变量

- 定义在方法中的变量是局部变量。
- 局部变量的作用域就是在方法中。

3.2.1 类的定义与创建和使用

区别	成员变量	局部变量
类中位置不同	类中，方法外	方法内，方法申明上
初始化值不同	有默认初始化值	没有，使用之前需要完成赋值
生命周期不同	随着对象的创建而存在，随着对象的消失而消失	随着方法的调用而存在，随着方法的运行结束而消失
作用域	整个类中有效	当前方法中有效

3.2.1 类的定义与创建和使用

注意事项:

- 类名首字母要大写，需要见名知意，采用大驼峰模式。
- 一个Java文件中可以定义多个class类，且只能一个类是public修饰，而且public修饰的类名必须成为代码文件名。
- 实际开发中建议还是一个文件定义一个class类。

3.2.1 类的定义与创建和使用

1. 类和对象是什么？

类：是共同特征的描述（设计图）；对象：是真实存在的具体案例

2. 如何得到对象？

```
public class Phone{
```

1. 成员变量（代表属性，一般是名词）
2. 成员方法（代表行为，一般是动词）

```
}
```

类名 对象名 = new 类名();

总结



3. 拿到对象后能做什么？

访问属性：对象名.成员变量

访问行为：对象名.方法名(...)

3.2.3 对象的引用传递

基本数据类型

- 整数类型
- 浮点数类型
- 布尔类型
- 字符类型

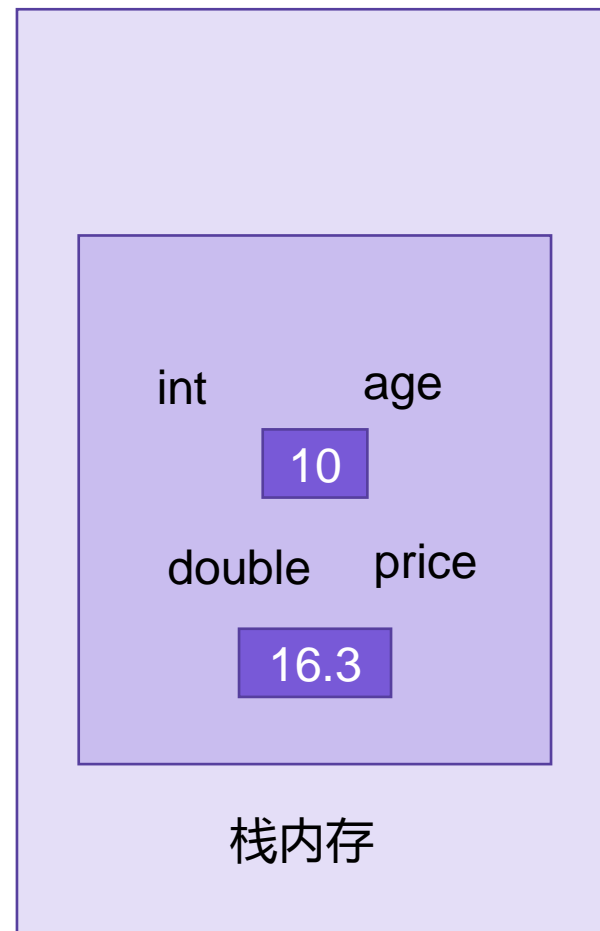
引用数据类型

除了左边的其他所有类型

3.2.3 对象的引用传递

基本数据类型

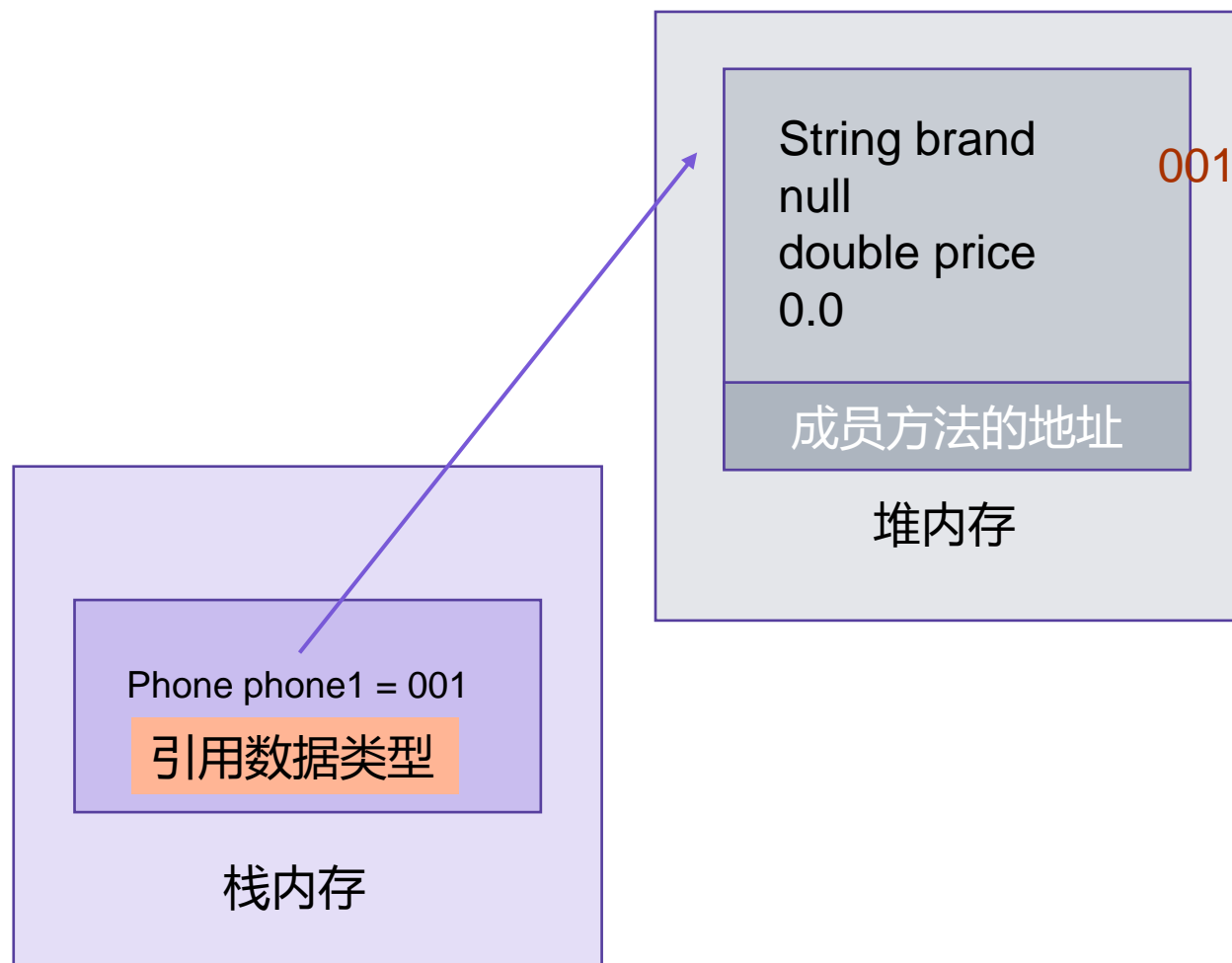
```
public class Main {  
    public static void main(String[] args) {  
        int age = 20;  
        double price = 16.3;  
    }  
}
```



3.2.3 对象的引用传递

引用数据类型

```
public class Main {  
    public static void main(String[] args) {  
        Phone phone1 = new Phone();  
    }  
}
```



3.2.3 对象的引用传递

从内存的角度去解释：

基本数据类型：数据值是存储在自己的空间中

特点：赋值给其他变量，也是赋的真实值。

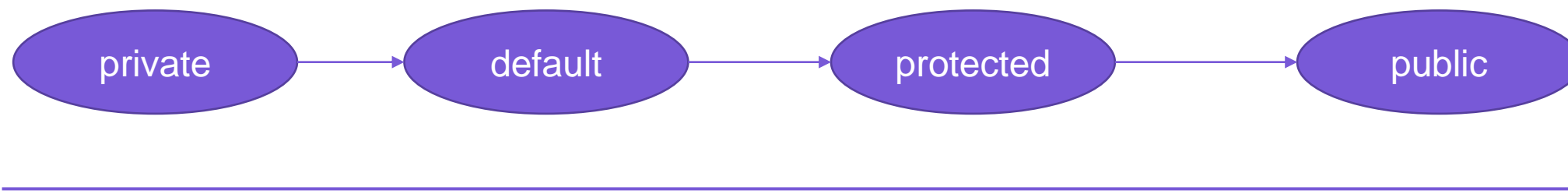
```
int a = 10;  
int b = a;
```

引用数据类型：数据值是存储在其他空间中，自己空间中存储的是地址值。

特点：赋值给其他变量，赋的地址值。

```
Phone phone1 = new Phone();  
Phone phone2 = phone1;
```

3.2.4 访问控制



private: private属于私有访问权限，用于修饰类的属性和方法。类的成员一旦使用了private关键字修饰，则该成员只能在本类中进行访问。

default: default属于默认访问权限。如果一个类中的属性没有任何的访问权限声明，则该属性或方法就是默认的访问权限，默认的访问权限可以被本包中的其他类访问，但是不能被其他的类访问。

protected: 属于受保护的访问权限。一个类中的成员使用了protected访问权限，则只能被本包及不同包的子类访问。

public: public属于公共访问权限。如果一个类中的成员使用了public访问权限，则该成员可以在所有类中被访问，不管是否在同一个包中。

3.2.4 访问控制

访问控制符	同一类中	同包子类	同包其他类	不同包子类	不同包其他类
public	√	√	√	√	√
protected	√	√	√	√	
default	√	√	√		
private	√				

/3.3

封装性



3.3.1 为什么要封装

面向对象三大特征

封装

继承

多态

封装：

- 告诉我们，如何正确设计对象的属性的方法。
- 封装可以被认为是一个保护屏障，防止该类的代码和数据被外部类定义的代码随机访问。

3.3.1 为什么要封装

需求：人画圆，请针对这个需求进行面向对象设计

```
public class Person{  
  
}
```

```
public class Circle{  
  
}
```

```
public void draw(){  
    System.out.println( "画圆" )  
}
```

```
Public class Circle(){  
    public void draw(){  
        System.out.println(画圆" )  
    }  
}
```

3.3.1 为什么要封装

封装：

对象代表什么，就得封装对应的数据，并提供数据对应的行为。

```
public class Circle{  
  
    double radius;  
    String backgroundColor;  
  
    public void draw(){  
        System.out.println("根据半径" + radius + "画一个"+backgroundColor+"圆")  
    }  
}
```


3.3.1 为什么要封装

案例： 人关门，关这个动作应该写在person还是door里面呢？

```
public class Door{  
    boolean flag = true;  
  
    public void open(){  
        开门，修改门的状态  
    }  
  
    public void close(){  
        关门，修改门的状态  
    }  
}
```

3.3.1 为什么要封装

封装：告诉我们，如何正确设计对象的属性的方法

```
public class Student{  
    String name;  
    int age;  
    String gender;  
}
```

```
Student st = new Student();  
st.age = 18;
```

3.3.1 为什么要封装

private: private属于私有访问权限，用于修饰类的属性和方法（成员变量和成员方法）。类的成员一旦使用了private关键字修饰，则该成员只能在本类中进行访问。

```
public class Student{  
  
    private String name;  
  
    private int age;  
  
    private String gender;  
  
}
```

```
Student st = new Student();  
    st.age = -18;
```



3.3.2 如何实现封装

```
public class Student{  
  
    String name;  
  
    int age;  
  
    String gender;  
  
}
```

```
int lily = 18  
    Student st = new Student();  
    if(lily > 0){  
        st.age = lily  
    }
```



每次判断都很不方便

对象代表什么，就得封装对应的数据，并提供数据对应的行为

3.3.2 如何实现封装

```
public class Student {  
    private String name;  
    private int age;  
    private String gender;  
  
    public void setAge(int b){  
        if(b >= 6 && b <= 60){  
            age = b;  
        }  
        else {  
            System.out.println("非法数据");  
        }  
    }  
  
    public int getAge(){  
        return age;  
    }  
}
```

```
Student st = new Student();  
st.setAge(30);
```

```
Student st = new Student();  
st.setAge(-30);
```

3.3.2 如何实现封装

1. private关键字是一个权限修饰符
2. 可以修饰成员（成员变量和成员方法）
3. 被private修饰的成员只能在本类中才能访问
4. 针对private修饰的成员变量，如果需要被其他类使用，提供相应的操作。
5. 提供 “setXxx(参数)” 方法，用于给成员变量赋值，方法用public修饰。
6. 提供 “getXxx()” 方法，用于获取成员变量的值，方法用public修饰



/3.4

构造方法



3.4 构造方法

实例化一个对象后，如果要为这个对象中的属性赋值，必须通过直接访问对象属性或调用 setter 方法才可以实现。

那如果说我想在实例化对象时为这个对象的属性赋值，那么我们就需要到构造方法。

```
public class Main {  
  
    public static void main(String[] args) {  
        Student st = new Student();  
    }  
}
```


3.4.1 定义构造方法

构造方法的格式

```
public class Student{  
    修饰符 类名 (参数) {  
        方法体  
    }  
}
```

执行时机：

1. 创建对象的时候由虚拟机调用，不能手动调用构造方法。
2. 每创建一次对象，就会调用一次构造方法。

特点：

1. 构造方法名称必须与类名一样。
2. 构造方法名称前不能有任何返回值类型的声明，连void都不能有。
3. 没有具体的返回值（不能由return带回结果数据）。

3.4.1 定义构造方法

定义构造方法

```
public class Constructor {  
    private int age;  
    private String name;  
    public Constructor(){  
        //不带参数  
    };  
    public Constructor(int age, String name){  
        //带参数  
    }  
}
```

调用构造方法

```
public class Main {  
    public static void main(String[] args) {  
        Constructor cs = new Constructor();  
        Constructor sc = new Constructor(12, "Li Lei");  
    }  
}
```

3.4.2 构造方法的重载

与普通方法一样，构造方法也可以重载，在一个类中可以定义多个构造方法，只要每个构造方法的参数类型或参数个数不同即可。

在创建对象时，可以通过调用不同的构造方法为不同的属性赋值。

```
public class Constructor {  
    private int age;  
    private String name;  
    private double salary;  
  
    public Constructor() {  
        //不带参数  
    }  
    public Constructor(int age, double salary) {  
        //带参数  
    }  
    public Constructor(int age, String name, double salary) {  
        //带参数  
    }  
}
```

3.4.2 构造方法注意事项

1. 构造方法的定义

- 如果没有定义构造方法，系统将给出一个默认的非参数构造方法
- 如果定义非构造方法，系统将不再提供默认的非构造方法

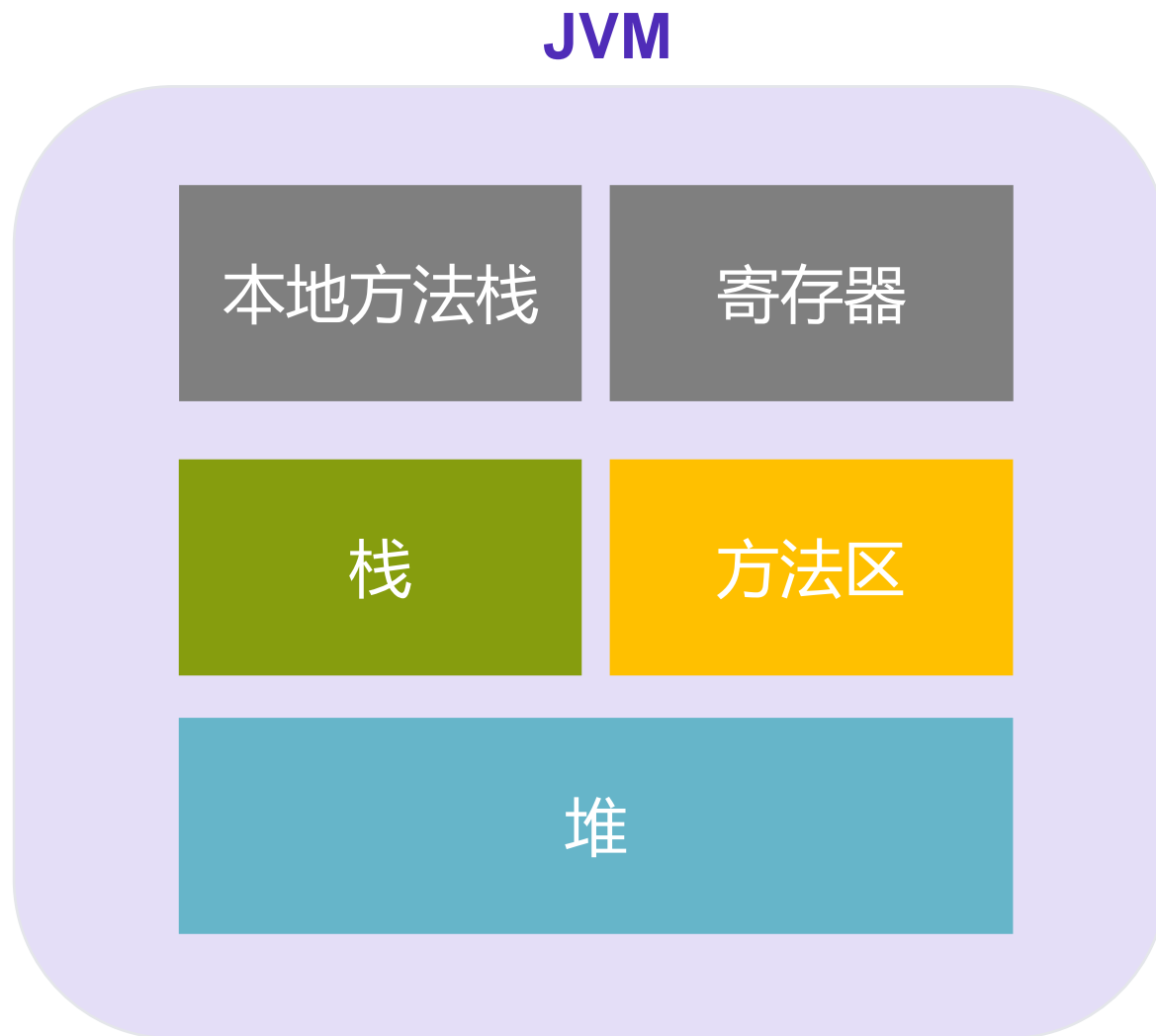
推荐的使用方法：

- 无论是否使用，都手动书写非参数构造方法，和带全部参数的构造方法。

三种情况的对象内存图

Java 内存分配介绍

- 栈
- 堆
- 方法区
- 本地方法栈
- 寄存器



三种情况的对象内存图

Java 内存分配介绍

- 栈
- 堆
- 方法区
- 本地方法栈
- 寄存器

方法区

HelloWorld.class
TestMain.class

字节码文件加载时进入的内存

栈内存

- 方法运行时所进入的内存
- 方法中的变量也是在这
- 执行完毕后就会出栈

堆内存

New 出来的东西会在这
块内存中开辟空间并产生
地址

三种情况的对象内存图

一个对象内存图

```
Student s = new Student();
```

1. 加载class文件（也就是将Student字节码文件加载到内存）
2. 申明局部变量（s）
3. 在堆内存中开辟一个空间（new）
4. 默认初始化
5. 显示初始化
6. 构造方法初始化
7. 将堆内存中的地址值赋值给左边的局部变量

三种情况的对象内存图

一个对象内存图

```
public class Student {  
    String name;  
    int age;  
    public void study(){  
        System.out.println("I am learning...");  
    }  
}
```

```
public class TestMain {  
    public static void main(String[] args) {  
        Student s = new Student();  
        System.out.println(s);  
        System.out.println(s.name+"..." +s.age);  
        s.name = "XiaoMing";  
        s.age = 18;  
        System.out.println(s.name+"..." +s.age);  
        s.study();  
    }  
}
```

栈内存

方法: study
sout("I am learning...")

方法: main
Student s **0x001**
sout(s);
sout(s.name+'...' +s.age)
s.name = "XiaoMing";
s.age=18;
sout(s.name+'...' +s.age)
s.study();

堆内存

String name **0x001**
XiaoMing
int age
18
成员方法的地址

方法区
TestMain.class Student.class
main();
String name
int age
Study();

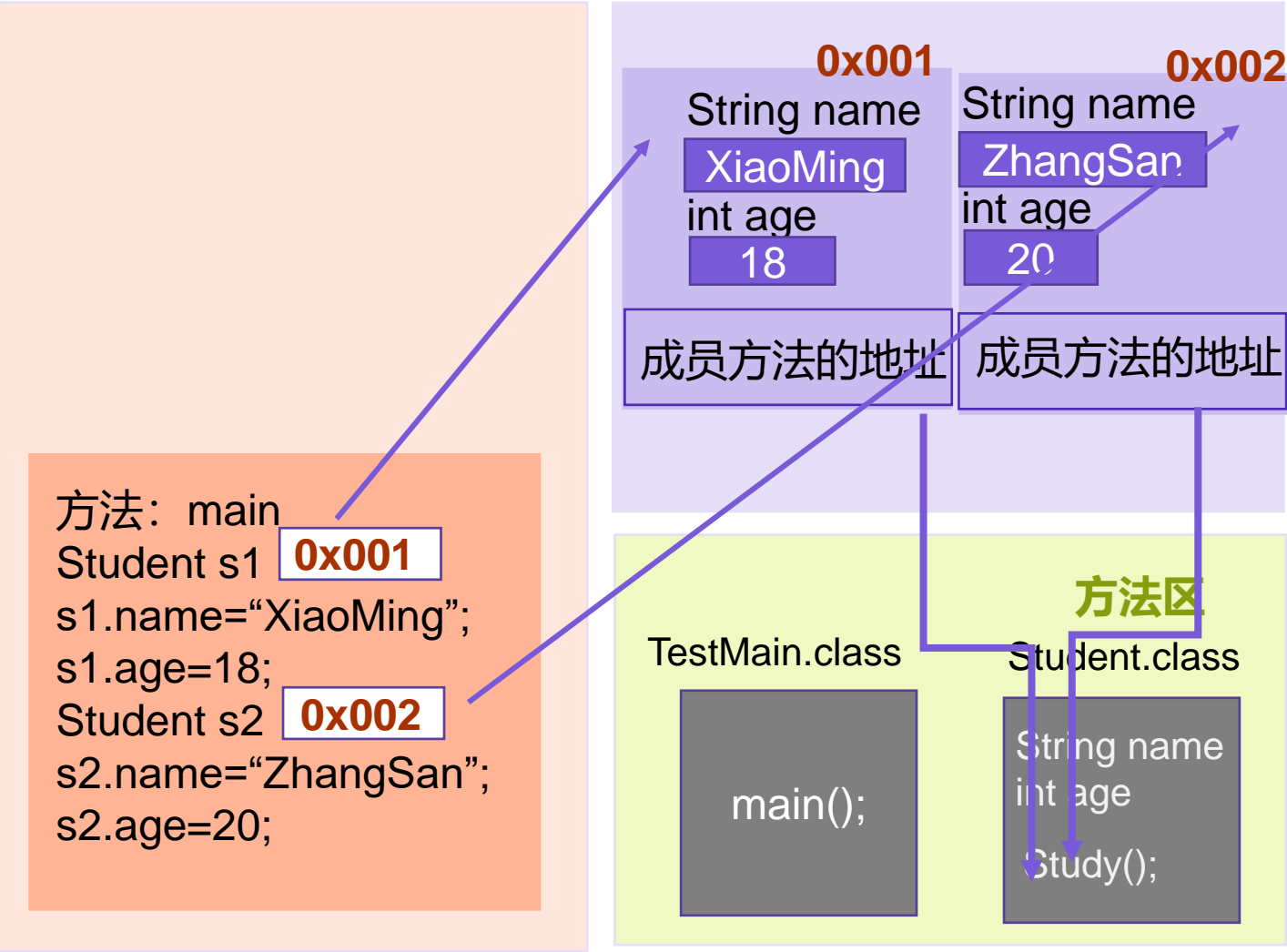
三种情况的对象内存图

两个对象内存图

```
public class TestMain {  
    public static void main(String[] args) {  
        Student s1 = new Student();  
        System.out.println(s1);  
        s1.name = "XiaoMing";  
        s1.age = 18;  
        System.out.println(s1.name+"..." +s1.age);  
        s1.study();  
        Student s2 = new Student();  
        System.out.println(s2);  
        s2.name = "ZhangSan";  
        s2.age = 20;  
        System.out.println(s2.name+"..." +s2.age);  
        s2.study();  
    }  
}
```

栈内存

堆内存



三种情况的对象内存图

两个引用指向同一个对象

```
public class Student {  
    String name;  
    int age;  
    public void study(){  
        System.out.println("I am learning...");  
    }  
}
```

```
public class TestMain {  
    public static void main(String[] args) {  
        Student s1 = new Student();  
        s1.name = "ZhangSan";  
        Student s2 = s1;  
        s2.name = "HanMeimei";  
        System.out.println(s1.name + "..." + s2.name);  
    }  
}
```

栈内存

方法: main
Student s1 **0x001**
s1.name="ZhangSan";
Student s2 = s1; **0x001**
s2.name = "HanMeimei";
sout(s1.name+"..." + s2.name)

堆内存

0x001

String name

HanMeimei

int age

0

成员方法的地址

方法区

TestMain.class

main();

Student.class

String name

int age

Study();

/3.5

this 关键字



3.5 this关键字

成员变量和局部变量重名时，需要使用到this关键字分辨成员变量和局部变量。

```
public class Friend {  
    private int age;  
    private String name;
```

成员变量

```
    public void playgame() {  
        int age = 20;  
        System.out.println(age);  
    }  
}
```

局部变量

请问输出是什么？

3.5 this关键字

就近原则

谁离我近，我就用谁

如果想打印出成员变量age?

```
public class Friend {  
    private int age;  
    private String name;  
  
    public void playgame() {  
        int age = 20;  
        System.out.println(age);  
    }  
}
```

20

```
public class Friend {  
    private int age;  
    private String name;  
  
    public void playgame() {  
        int age = 20;  
        System.out.println(this.age);  
    }  
}
```

0

3.5 this关键字

Java中的this关键字语法比较灵活，主要作用有以下3种：

1. 使用this关键字调用本类中的属性
2. 使用this关键字调用成员方法
3. 使用this关键字调用本类的构造方法

3.5.1 使用this关键字调用本类中的属性

1.set方法, 一般来说传入参数是需要有意义的, 所以会和成员变量发生冲突, 这个时候我们需要加this.

```
private String name
public void setName(String name){
    if(name.length()>100){
        System.out.println("非法名字, 超出数据库存储范围");
    }
    else {
        this.name = name;
    }
}
```

2. 在带参构造方法中, 我们在规范了参数命名的同时, 加this以避免成员变量和局部变量的混淆。

```
private int age;
private String name;
private double salary;

public Constructor(int age,double salary){
    this.age = age;
    this.salary = salary;
}
```

3.5.2 使用this关键字调用成员方法

以下代码中，用this关键字调用了readBook()方法。此处的this关键字也是可以省略不写的。

```
public class Friend {  
    public void readBook() {  
        ...  
    }  
    public void read() {  
        this.readBook();  
    }  
}
```


3.5.3 使用this关键字调用本类的构造方法

在一个构造方法中使用 “this (参数1, 参数2) ” 的形式调用其他的构造方法。

```
public class Friend {  
    private int age;  
    private String name;  
    private String gender;  
  
    public Friend() {  
        System.out.println("调用无参构造方法成功");  
    }  
    public Friend(int age, String name) {  
        this();  
        System.out.println("The age is " + age + ", and the name is " + name);  
        System.out.println("调用两个参数的构造方法成功");  
    }  
    public Friend(int age, String name, String gender) {  
        this(20, "XiaoMing");  
        System.out.println("The age is " + age + ", and the name is " + name + ", and the gender is " +  
            gender);  
    }  
}
```

3.5.3 使用this关键字调用本类的构造方法

注意事项:

- 1.只能在构造方法中使用this调用其他的构造方法，不能在成员方法中通过this调用其他构造方法。
2. 在构造方法中，使用this调用构造方法的语句必须位于第一行，且只能出现一次。
- 3.不能在一个类的两个构造方法中使用this互相调用。

标准的javabeen

一种类，而且是特殊的、可重用的类。

1. 类名需要见名知意
2. 成员变量使用private修饰
3. 提供至少两个构造方法
 - 无参数构造方法
 - 带全部参数得构造方法
4. 成员方法
 - 提供每一个成员变量对应得setXxx()/getXxx()
 - 如果还有其他行为，也需要写上

标准的javabea

Zhijian

zhijian-ask.com/#/signup

GmailYouTube翻译Quasar FrameworkSwagger UIMy Drive - Googl...OverviewExperts Detail - Z...GeeksforGeeks [...]

其他书签

知见 · ZHIJIAN

知见立知，即无明本，知见无见，斯即涅槃。

The Knowledge of Ignorance

User Name *

☒ Female ☐ male

Mobile No *

password*

repeat password*

Personal Interest

Input validation number

0 / 6

* By signing up, you agree to our Terms, Data Policy and [Cookie Policy](#).

SIGN UP

Have an account? [LOG IN](#)

知见 · ZHIJIAN

知见立知，即无明本，知见无见，斯即涅槃。

The Knowledge of Ignorance

User Name *

☒ Female ☐ male

Mobile No *

password*

repeat password*

Personal Interest

Input validation number

0 / 6

* By signing up, you agree to our Terms, Data Policy and [Cookie Policy](#).

SIGN UP

/3.6

代码块



3.6 代码块

代码块：

- 普通代码块

- 构造代码块

- 静态代码块

- 同步代码块


后面学到相应的地方会讲



3.6.1 普通代码块

直接在代码方法或语句中定义的代码块

```
public class Main {  
    public static void main(String[] args) {  
        {  
            int age = 20;  
            System.out.println("This is normal code block! "+ i);  
        }  
        {  
            int age = 200;  
            System.out.println("This is normal code block! " + i);  
        }  
    }  
}
```



1. 每一对 “{}” 括起来的代码都称为一个代码块。
2. main中的两个代码块起了限定作用，所以两个age互不影响。

3.6.2 构造块

在类中定义的代码块

```
public class ConstructorBlock {  
    private String name;  
    private int age;  
    {  
        System.out.println("This is constructor block!");  
    }  
    public ConstructorBlock() {  
        System.out.println("This is constructor method!");  
    }  
}
```

1. 构造块定义在成员位置，与构造方法，成员属性同级。
2. 重点在于构造块的执行顺序优于构造方法，与构造块是否写在构造方法后面是没有关系的。

/3.7

static关键字



3.7 static关键字

- static表示静态，是Java中的一个修饰符，可以修饰成员方法，成员变量
- 在堆内存中每个对象都有自己的属性，如果希望某些属性被所有对象共享，就必须将其声明为static属性。
- 如果属性使用了static关键字进行修饰，则该属性可以直接使用类名称进行调用。

3.7 static关键字

static表示静态，是Java中的一个修饰符，可以修饰成员方法，成员变量

被static修饰的成员变量，叫做静态变量

特点：

- 被该类所有对象共享

调用方法：

- 类名调用（推荐）
- 对象名调用

```
AirCon.brand = "美的";
```

```
ac1.brand = "格力";
```

被static修饰的成员方法，叫做静态方法

特点：

- 可以不创建对象，通过类名直接调用某个方法

调用方法：

- 类名调用（推荐）
- 对象名调用

3.7.1 静态属性

```
public class WashMachine {  
    public static String brand;  
    public int price;  
    public String version;  
  
    public void showDetail(){  
        System.out.println("Brand is "+ brand+", " +  
            "price is "+ price+", version is "+ version);  
    }  
}
```

```
public class TestMain {  
    public static void main(String[] args) {  
        WashMachine.brand = "Siemens";  
        WashMachine wm1 = new WashMachine();  
        wm1.price = 1999;  
        wm1.version = "SIE-20019";  
        wm1.showDetail();  
  
        WashMachine wm2 = new WashMachine();  
        wm2.showDetail();  
    }  
}
```

栈内存

方法: wm2.showDetail();
System.out.println("Brand
is "+ brand+", " + "price is
"+ price+", version is "+
version);
"+ price+", version is "+
version);

方法: main
WashMachine.brand =
"Siemens";
WashMachine wm1; 0x0011
wm1.price = 1999;
wm1.version = "SIE-20019"
wm1.showDetail();
WashMachine wm2; 0x0022

堆内存

0x0011
int price
1999
String version
SIE-20019

0x0022
int price
null
String version
0

static String brand
Siemens

静态存储位置 (静态区)

而加载

的

3.7.2 静态方法

StaticSample类中写一个静态方法

```
public class StaticSample {  
    public static double getAverage(int[] array){  
        double sum = 0;  
        for (int j : array) {  
            sum = sum + j;  
        }  
        return sum/array.length;  
    }  
}
```

在main方法中进行调用

```
public class TestMain {  
    public static void main(String[] args) {  
        int[] array1 = {1,2,3,4,5,6,7,8};  
        double averageArray = StaticSample.getAverage(array1);  
        System.out.println("The average is "+ averageArray);  
    }  
}
```

3.7.2 静态方法

注意事项：

- 静态方法只能访问静态变量和静态方法。
- 非静态方法可以访问静态变量或者静态方法，也可以访问非静态的成员变量和非静态的成员方法。
- 静态方法中是没有this关键字

3.7.2 静态方法

静态方法不能访问非静态

```
public class Student {  
    String name;  
    static String teacherName;  
  
    public static void method(){  
        System.out.println(name + "... " + teacherName);  
    }  
  
    public void show(){  
        System.out.println(name + "... " + teacherName);  
    }  
}
```

```
public class TestMain {  
    public static void main(String[] args) {  
        Student.teacherName = "Qinru";  
        Student.method();  
    }  
}
```

栈内存

静态方法不能调用实例变量

方法: Student.method();
System.out.println(name +
"..." + teacherName);

方法: main
Student.teacherName =
"Qinru";
Student.method();

堆内存

static String
teacherName
Qinru

静态存储位置 (静态区)

方法区

Student.class

name;
method();
show();

3.7.2 静态方法

非静态方法可以访问所有

```
public class Student {  
    String name;  
    static String teacherName;  
    public static void method(){  
        System.out.println("静态方法");  
    }  
    public void show(){  
        System.out.println(name + "... " + teacherName);  
    }  
}
```

```
public class TestMain {  
    public static void main(String[] args) {  
        Student s = new Student();  
        s.name = "XiaoMing";  
        s.show();  
    }  
}
```

栈内存

方法: s.show();
System.out.println(name +
"... " + teacherName);
Method();

方法: main
Student s; **0x0011**
s.name = "XiaoMing";

堆内存

0x0011
String name
XiaoMing

static String
teacherName
null

静态存储位置 (静态区)

方法区

Student.class
name;
method();
show();

3.7.3 静态代码块

代码块:

- 普通代码块
- 构造代码块
- 静态代码块
- 同步代码块

3.7.3 静态代码块

用static关键字修饰的代码块为静态代码块

```
public class ConstructorBlock {  
    static {  
        System.out.println("This is static block");  
    }  
}
```

1. 当类被加载时，静态代码块会执行，由于类只会被加载一次，因此静态代码块只会执行一次。
2. 代码块的执行顺序为静态代码，构造代码块，构造方法。

3.7.3 静态代码块

main方法中创建了3次实例化对象

```
public class Main {  
    public static void main(String[] args) {  
        ConstructorBlock cb = new ConstructorBlock();  
        System.out.println("-----");  
        ConstructorBlock cb1 = new ConstructorBlock();  
        System.out.println("-----");  
        ConstructorBlock cb2 = new ConstructorBlock();  
    }  
}
```

执行顺序为静态代码块，构造代码块，构造方法。static修饰的成员会随着class文件一同加载，并只会加载一次。

```
This is static block  
This is constructor block! female  
This is constructor block! male  
This is constructor method!  
-----  
This is constructor block! female  
This is constructor block! male  
This is constructor method!  
-----  
This is constructor block! female  
This is constructor block! male  
This is constructor method!
```

main 方法讲解

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

- public: 被JVM调用，访问权限足够大
- static: 被JVM调用，不用创建对象，直接类名访问。因为main方法是静态的，所以测试类中其他方法也需要是静态的。
- void: 被JVM调用，不需要给JVM返回值
- main: 一个通用的名称，虽然不是关键字，但是被JVM识别
- String[] args: 以前用于接收键盘录入数据的，现在没用。

第三章总结

1. 面向对象的思想
2. 类与对象
3. 封装性
4. 构造方法
5. this关键字
6. 代码块
7. static关键字



- 三种情况对象内存图
- 标准的javabean的写法
- main方法讲解