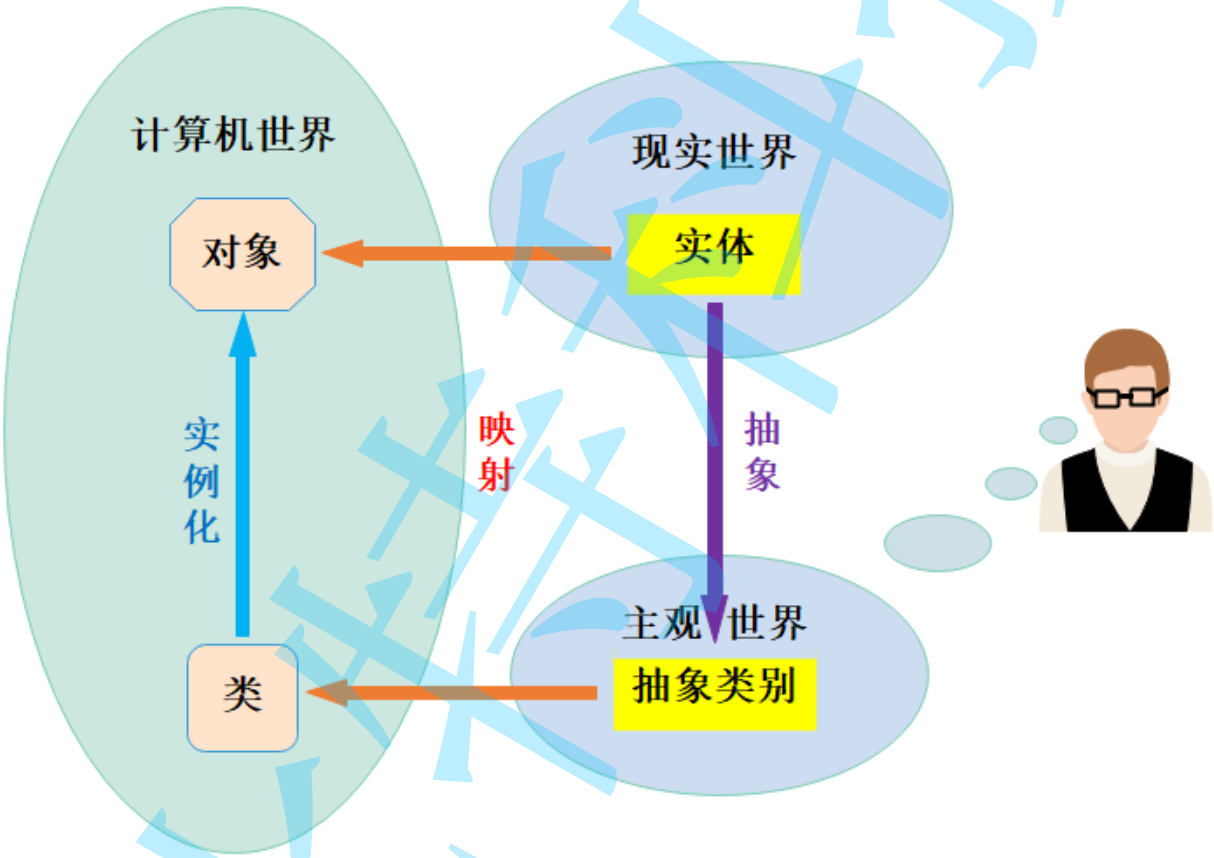
类和对象

本节目标

> - 掌握类的定义方式以及对象的实例化
> - 掌握类中的成员变量和成员方法的使用
> - 掌握对象的整个初始化过程

1. 类与对象的初步认知



C语言是**面向过程**的，**关注**的是**过程**，分析出求解问题的步骤，通过函数调用逐步解决问题。

JAVA是**基于面向对象**的，**关注**的是**对象**，将一件事情拆分成不同的对象，靠对象之间的交互完成。

面向过程注重的是过程，在整个过程中所涉及的行为，就是功能。

面向对象注重的是对象，也就是参与过程所涉及到的主体。是通过逻辑将一个个功能实现连接起来

面向过程： 1.把冰箱打开 2. 把大象放入 3. 冰箱关起来 **面向对象：** 打开冰箱，储存，关闭都是对冰箱的操作，是冰箱的行为。**冰箱就是一个对象，所以只要操作冰箱所具备的功能，都要定义在冰箱中。**

【面向对象概念】

1.面向对象是思考问题的一种思考方式，是一种思想。比如：概念与实例。理论与实践。名和实等等。。

2.类就是一类对象的统称。对象就是这一类具体化的一个实例。

3.面向对象的好处：将复杂的事情变简单了，只要面对一个对象就行。

【面向对象设计】

面向对象设计把握一个重要的经验：谁拥有数据，谁对外提供操作这些数据（私有）的方法！

（被动的一方是数据的拥有者，主动的一方是执行者）

开发时：找对象，建对象，用对象，并维护对象之间的关系。

后期学习过程当中，我们会就这三点进行深入学习。

简而言之

面向对象就是用代码(类)来描述客观世界的事物的一种方式. 一个类主要包含一个事物的属性和行为

2. 类和类的实例化

类就是一类对象的统称。对象就是这一类具体化的一个实例。

简单的例子：我们做月饼的模子就是一个类，而通过这个模子可以做出月饼，那么在这个例子当中，类就是那个模子，而月饼就是那个对象,所以月饼就是一个实体。**一个模子可以实例化无数个对象。**

总的来说：类相当于一个模板，对象是由模板产生的样本。**一个类，可以产生无数的对象。**

声明一个类就是创建一个新的数据类型，而类在`Java`中属于**引用类型**，`Java`使用关键字`class`来声明类。我们来看以下简单的声明一个类。

基本语法

```
// 创建类
class <class_name>{
    field;//成员属性
    method;//成员方法
}

// 实例化对象
<class_name> <对象名> = new <class_name>();
```

class为定义类的关键字，ClassName为类的名字，{}中为类的主体。

类中的元素称为：成员属性。类中的函数称为：成员方法。

示例：

```java
class Person {
    public int age;//成员属性  实例变量
    public String name;
    public String sex;
    public void eat() {//成员方法
        System.out.println("吃饭!");
    }
    public void sleep() {
        System.out.println("睡觉!");
    }
}
```

注意事项

- 和之前写的方法不同, 此处写的方法不带 static 关键字. 后面我们会详细解释 static 是干啥的.

类的实例化

用类类型创建对象的过程，称为类的实例化

1. **类只是**一个**模型**一样的东西，限定了类有哪些成员.
2. 一个类可以实例化出多个对象，**实例化出的对象 占用实际的物理空间，存储类成员变量**
3. 做个比方。**类实例化出对象就像现实中使用建筑设计图建造出房子，类就像是设计图**，只设计出需要什么东西，但是并没有实体的建筑存在，同样类也只是一个设计，实例化出的对象才能实际存储数据，占用物理空间


```java
class Person {
    public int age;//成员属性  实例变量
    public String name;
    public String sex;
```

```
    public void eat() {//成员方法
        System.out.println("吃饭!");
    }
    public void sleep() {
        System.out.println("睡觉!");
    }
}
public class Main{
    public static void main(String[] args) {
        Person person = new Person();//通过new实例化对象
        person.eat();//成员方法调用需要通过对象的引用调用
        person.sleep();
        //产生对象      实例化对象
        Person person2 = new Person();
        Person person3 = new Person();
    }
}
```

输出结果为：

```
吃饭!
睡觉!
```

注意事项

- new 关键字用于创建一个对象的实例.
- 使用 `.` 来访问对象中的属性和方法.
- 同一个类可以创建对个实例.

3. 类的成员

类的成员可以包含以下：字段、方法、代码块、内部类和接口等。

此处我们重点介绍前三个.

3.1 字段/属性/成员变量

在类中, 但是方法外部定义的变量. 这样的变量我们称为 "字段" 或 "属性" 或 "成员变量"(三种称呼都可以, 一般不会严格区分).

用于描述一个类中包含哪些数据.

```
class Person {
    public String name;    // 字段
    public int age;
}

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

```
        Person person = new Person();
        System.out.println(person.name);
        System.out.println(person.age);
    }
}

// 执行结果
null
0
```

注意事项

- 使用 `.` 访问对象的字段.
- "访问" 既包含读, 也包含写.
- 对于一个对象的字段如果没有显式设置初始值, 那么会被设置一个默认的初值.

> 默认值规则
>
> - 对于各种数字类型, 默认值为 0.
> - 对于 boolean 类型, 默认值为 false.
> - 对于引用类型(String, Array, 以及自定制类), 默认值为 null

认识 null

null 在 Java 中为 "空引用", 表示不引用任何对象. 类似于 C 语言中的空指针. 如果对 null 进行 `.` 操作就会引发异常.

```
class Person {
    public String name;
    public int age;
}

class Test {
    public static void main(String[] args) {
        Person person = new Person();
        System.out.println(person.name.length());   // 获取字符串长度
    }
}

// 执行结果
Exception in thread "main" java.lang.NullPointerException
        at Test.main(Test.java:9)
```

字段就地初始化

很多时候我们不希望字段使用默认值, 而是需要我们显式设定初值. 可以这样写:

```
class Person {
    public String name = "张三";
```

```
    public int age = 18;
}

class Test {
    public static void main(String[] args) {
        Person person = new Person();
        System.out.println(person.name);
        System.out.println(person.age);
    }
}

// 执行结果
张三
18
```

3.2 方法 (method)

就是我们曾经讲过的方法.

用于描述一个对象的行为.

```
class Person {
    public int age = 18;
    public String name = "张三";

    public void show() {
        System.out.println("我叫" + name + ", 今年" + age + "岁");
    }
}

class Test {
    public static void main(String[] args) {
        Person person = new Person();
        person.show();
    }
}

// 执行结果
我叫张三，今年18岁
```

此处的 show 方法, 表示 Person 这个对象具有一个 "展示自我" 的行为.

这样的 show 方法是和 `person` 实例相关联的. 如果创建了其他实例, 那么 show 的行为就会发生变化

```
Person person2 = new Person();
person2.name = "李四";
person2.age = 20;
person2.show()

// 执行结果
我叫李四，今年20岁
```

方法中还有一种特殊的方法称为 **构造方法 (construction method)**

在实例化对象的时候会被自动调用到的方法, 方法名字和类名相同, 用于对象的初始化.

虽然我们前面已经能将属性就地初始化, 但是有些时候可能需要进行一些更复杂的初始化逻辑, 那么就可以使用构造方法.

后面我们会详细介绍构造方法的语法.

3.3 static 关键字

> 1、修饰属性
>
> 2、修饰方法
>
> 3、代码块(本课件中会介绍)
>
> 4、修饰类(后面讲内部类会讲到)

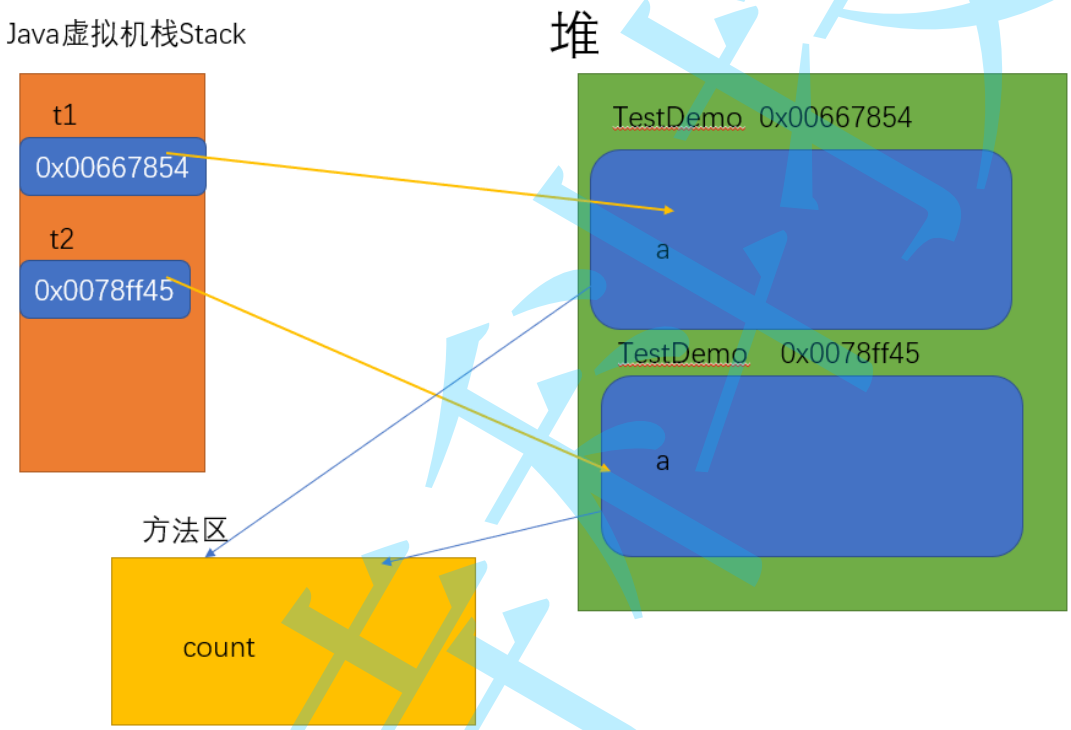
a) 修饰属性，Java静态属性和类相关, 和具体的实例无关. 换句话说, 同一个类的不同实例共用同一个静态属性.

```
class TestDemo{
    public int a;
    public static int count;
}

public class Main{

    public static void main(String[] args) {
        TestDemo t1 = new TestDemo();
        t1.a++;
        TestDemo.count++;
        System.out.println(t1.a);
        System.out.println(TestDemo.count);
        System.out.println("============");
        TestDemo t2 = new TestDemo();
        t2.a++;
        TestDemo.count++;
        System.out.println(t2.a);
        System.out.println(TestDemo.count);
    }
}
```

输出结果为：

```
1
1
============
1
2
```

示例代码内存解析：count被static所修饰，所有类共享。且不属于对象，访问方式为：类名 . 属性。



b) 修饰方法

如果在任何方法上应用 `static` 关键字，此方法称为静态方法。

- 静态方法属于类，而不属于类的对象。
- 可以直接调用静态方法，而无需创建类的实例。
- 静态方法可以访问静态数据成员，并可以更改静态数据成员的值。

```
class TestDemo{
    public int a;
    public static int count;

    public static void change() {
        count = 100;
        //a = 10; error  不可以访问非静态数据成员
    }
}

public class Main{
```

```
    public static void main(String[] args) {
        TestDemo.change(); //无需创建实例对象 就可以调用
        System.out.println(TestDemo.count);
    }
}
```

输出结果：

```
100
```

注意事项1: 静态方法和实例无关, 而是和类相关. 因此这导致了两个情况:

- 静态方法不能直接使用非静态数据成员或调用非静态方法(非静态数据成员和方法都是和实例相关的).
- this和super两个关键字不能在静态上下文中使用(this 是当前实例的引用, super是当前实例父类实例的引用, 也是和当前实例相关).

注意事项2

- 我们曾经写的方法为了简单, 都统一加上了 static. 但实际上一个方法具体要不要带 static, 都需要是情形而定.
- main 方法为 static 方法.

3.4 小结

观察以下代码, 分析内存布局.

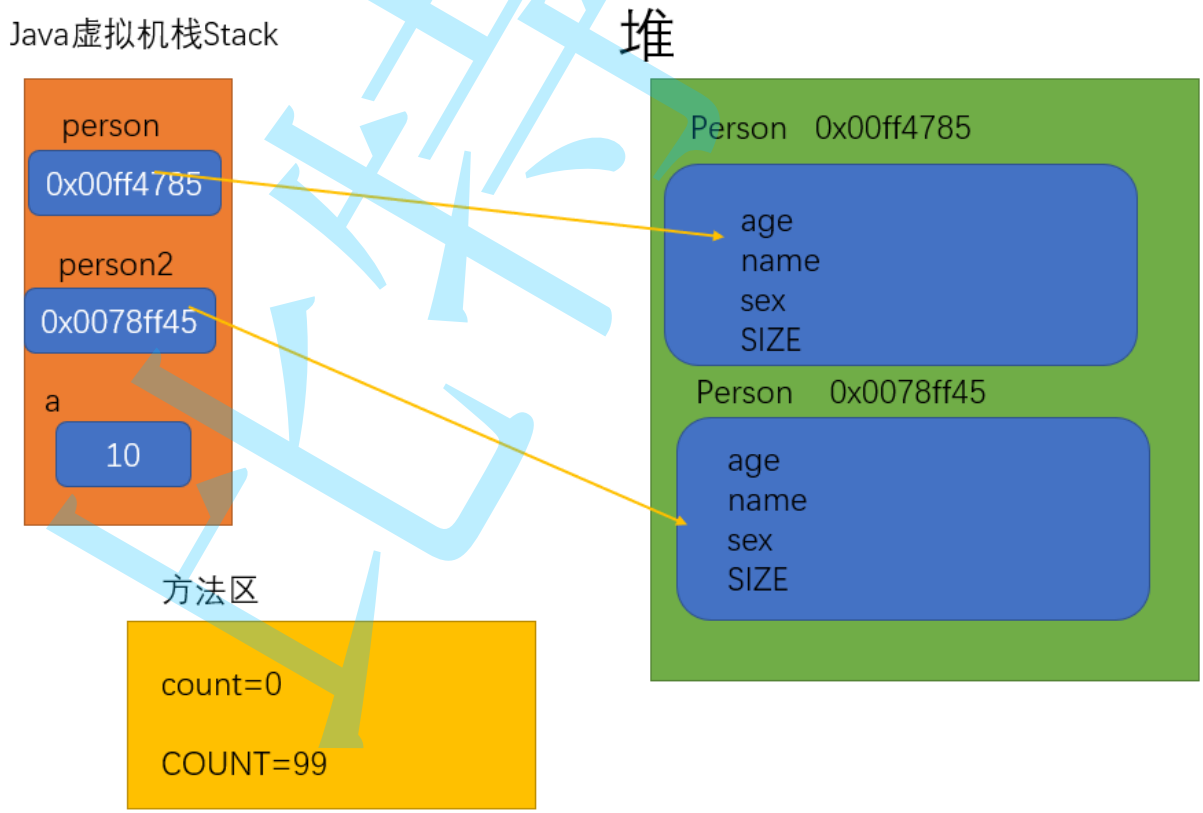
```
class Person {
    public int age;//实例变量     存放在对象内
    public String name;//实例变量
    public String sex;//实例变量
    public static int count;//类变量也叫静态变量，编译时已经产生，属于类本身，且只有一份。存放在方法区
    public final int SIZE = 10;//被final修饰的叫常量，也属于对象。 被final修饰，后续不可更改
    public static final int  COUNT = 99;//静态的常量，属于类本身，只有一份  被final修饰，后续不可更
改
    //实例成员函数
    public void eat() {
       int a = 10;//局部变量
       System.out.println("eat()!");
    }
    //实例成员函数
    public void sleep() {
       System.out.println("sleep()!");
    }
    //静态成员函数
     public static void staticTest(){
         //不能访问非静态成员
        //sex = "man"; error
         System.out.println("StaticTest()");
     }
}
public class Main{
    public static void main(String[] args) {
```

```
        //产生对象 实例化对象
        Person person = new Person();//person为对象的引用
        System.out.println(person.age);//默认值为0
        System.out.println(person.name);//默认值为null
        //System.out.println(person.count);//会有警告!
        //正确访问方式:
        System.out.println(Person.count);
        System.out.println(Person.COUNT);
        Person.staticTest();
        //总结:所有被static所修饰的方法或者属性,全部不依赖于对象。
        person.eat();
        person.sleep();
    }
}
```

输出结果为:

```
0
null
0
99
StaticTest()
eat()!
sleep()!
```

数据属性的内存布局:



4. 封装

> **什么叫封装?**
>
> <<代码大全>> 开篇就在讨论一个问题: 软件开发的本质就是对程序复杂程度的管理. 如果一个软件代码复杂程度太高, 那么就无法继续维护. 如何管理复杂程度? 封装就是最基本的方法.
>
> 在我们写代码的时候经常会涉及两种角色: **类的实现者**和**类的调用者**.
>
> 封装的本质就是让类的调用者不必太多的了解类的实现者是如何实现类的, 只要知道如何使用类就行了.
>
> 这样就降低了类使用者的学习和使用成本, 从而降低了复杂程度.

4.1 private实现封装

private/ public 这两个关键字表示 "访问权限控制" .

- 被 public 修饰的成员变量或者成员方法, 可以直接被类的调用者使用.
- 被 private 修饰的成员变量或者成员方法, 不能被类的调用者使用.

> 换句话说, 类的使用者根本不需要知道, 也不需要关注一个类都有哪些 private 的成员. 从而让类调用者以更低的成本来使用类.

直接使用 public

```
class Person {
    public String name = "张三";
    public int age = 18;
}

class Test {
    public static void main(String[] args) {
        Person person = new Person();
        System.out.println("我叫" + person.name + ", 今年" + person.age + "岁");
    }
}

// 执行结果
我叫张三，今年18岁
```

- 这样的代码导致类的使用者(main方法的代码)必须要了解 Person 类内部的实现, 才能够使用这个类. 学习成本较高
- 一旦类的实现者修改了代码(例如把 name 改成 myName), 那么类的使用者就需要大规模的修改自己的代码, 维护成本较高.

范例：使用 private 封装属性, 并提供 public 方法供类的调用者使用.

```
class Person {
    private String name = "张三";
    private int age = 18;

    public void show() {
        System.out.println("我叫" + name + ", 今年" + age + "岁");
    }
}

class Test {
    public static void main(String[] args) {
        Person person = new Person();
        person.show();
    }
}

// 执行结果
我叫张三, 今年18岁
```

- 此时字段已经使用 private 来修饰. 类的调用者(main方法中)不能直接使用. 而需要借助 show 方法. 此时类的使用者就不必了解 Person 类的实现细节.
- 同时如果类的实现者修改了字段的名字, 类的调用者不需要做出任何修改(类的调用者根本访问不到 name, age 这样的字段).

> 那么问题来了~~ 类的实现者万一修改了 public 方法 show 的名字, 岂不是类的调用者仍然需要大量修改代码嘛?
>
> 这件事情确实如此, 但是一般很少会发生. 一般类的设计都要求类提供的 public 方法能比较稳定, 不应该频繁发生大的改变. 尤其是对于一些基础库中的类, 更是如此. 每次接口的变动都要仔细考虑兼容性问题.

注意事项

- private 不光能修饰字段, 也能修饰方法
- 通常情况下我们会把字段设为 private 属性, 但是方法是否需要设为 public, 就需要视具体情形而定. 一般我们希望一个类只提供 "必要的" public 方法, 而不应该是把所有的方法都无脑设为 public.

4.2 getter和setter方法

当我们使用 private 来修饰字段的时候, 就无法直接使用这个字段了.

代码示例

```
class Person {
    private String name = "张三";
    private int age = 18;

    public void show() {
        System.out.println("我叫" + name + ", 今年" + age + "岁");
    }
}
```

```
class Test {
    public static void main(String[] args) {
        Person person = new Person();
        person.age = 20;
        person.show();
    }
}

// 编译出错
Test.java:13: 错误: age可以在Person中访问private
        person.age = 20;
              ^
1 个错误
```

此时如果需要获取或者修改这个 private 属性, 就需要使用 getter / setter 方法.

代码示例

```
class Person {
    private String name;//实例成员变量
    private  int age;

    public void setName(String name){
        //name = name;//不能这样写
        this.name = name;//this引用, 表示调用该方法的对象
    }
    public String getName(){
        return name;
    }

    public void show(){
        System.out.println("name: "+name+" age: "+age);
    }
}
public static void main(String[] args) {
    Person person = new Person();
    person.setName("caocao");
    String name = person.getName();
    System.out.println(name);
    person.show();
}

// 运行结果
caocao
name: caocao age: 0
```

注意事项

- getName 即为 getter 方法, 表示获取这个成员的值.
- setName 即为 setter 方法, 表示设置这个成员的值.

- 当set方法的形参名字和类中的成员属性的名字一样的时候，如果不使用this, 相当于自赋值. this 表示当前实例的引用.
- 不是所有的字段都一定要提供 setter / getter 方法, 而是要根据实际情况决定提供哪种方法.
- 在 IDEA 中可以使用 alt + insert (或者 alt + F12) 快速生成 setter / getter 方法. 在 VSCode 中可以使用鼠标右键菜单 -> 源代码操作 中自动生成 setter / getter 方法.

5. 构造方法

5.1 基本语法

构造方法是一种特殊方法, 使用关键字new实例化新对象时会被自动调用, 用于完成初始化操作.

new 执行过程

- 为对象分配内存空间
- 调用对象的构造方法

语法规则

- 1.方法名称必须与类名称相同
- 2.构造方法没有返回值类型声明
- 3.每一个类中一定至少存在一个构造方法（没有明确定义，则系统自动生成一个无参构造）

注意事项

- 如果类中没有提供任何的构造函数，那么编译器会默认生成一个不带有参数的构造函数
- 若类中定义了构造方法，则默认的无参构造将不再生成.
- 构造方法支持重载. 规则和普通方法的重载一致.

代码示例

```
class Person {

    private String name;//实例成员变量
    private  int age;
    private String sex;
    //默认构造函数     构造对象
    public Person() {
        this.name = "caocao";
        this.age = 10;
        this.sex = "男";
    }
    //带有3个参数的构造函数
    public Person(String name,int age,String sex) {
        this.name = name;
        this.age = age;
        this.sex = sex;
```

```
    }

    public void show(){
        System.out.println("name: "+name+" age: "+age+" sex: "+sex);
    }

}
public class Main{
    public static void main(String[] args) {
        Person p1 = new Person();//调用不带参数的构造函数   如果程序没有提供会调用不带参数的构造函数
        p1.show();
        Person p2 = new Person("zhangfei",80,"男");//调用带有3个参数的构造函数
        p2.show();
    }
}

// 执行结果
name: caocao age: 10 sex: 男
name: zhangfei age: 80 sex: 男
```

5.2 this关键字

this表示当前对象引用(**注意不是当前对象**). 可以借助 this 来访问对象的字段和方法.

```
class Person {
    private String name;//实例成员变量
    private  int age;
    private String sex;

    //默认构造函数     构造对象
    public Person() {
        //this调用构造函数
        this("bit", 12, "man");//必须放在第一行进行显示
    }

    //这两个构造函数之间的关系为重载。
    public Person(String name,int age,String sex) {
        this.name = name;
        this.age = age;
        this.sex = sex;
    }

    public void show() {
        System.out.println("name: "+name+" age: "+age+" sex: "+sex);
    }
}
public class Main{
    public static void main(String[] args) {
        Person person = new Person();//调用不带参数的构造函数
        person.show();
    }
```

```
}

// 执行结果
name: bit age: 12 sex: man
```

> 我们会发现在构造函数的内部，我们可以使用this关键字，构造函数是用来构造对象的，对象还没有构造好，我们就使用了this，那this还代表当前对象吗？当然不是，this代表的是当前对象的引用。

6. 认识代码块

字段的初始化方式有：

1. 就地初始化
2. 使用构造方法初始化
3. 使用代码块初始化

前两种方式前面已经学习过了，接下来我们介绍第三种方式，使用代码块初始化.

6.1 什么是代码块

使用`{}`定义的一段代码.

根据代码块定义的位置以及关键字，又可分为以下四种：

- 普通代码块
- 构造块
- 静态块
- 同步代码块（后续讲解多线程部分再谈）

6.2 普通代码块

普通代码块：定义在方法中的代码块.

```
public class Main{
    public static void main(String[] args) {
        { //直接使用{}定义，普通方法块
            int x = 10 ;
            System.out.println("x1 = " +x);
        }
        int x = 100 ;
        System.out.println("x2 = " +x);
    }
}

// 执行结果
x1 = 10
x2 = 100
```

这种用法较少见/.

6.3 构造代码块

构造块：定义在类中的代码块(不加修饰符)。也叫：**实例代码块**。构造代码块一般用于初始化实例成员变量。

```
class Person{
    private String name;//实例成员变量
    private  int age;
    private String sex;

    public Person() {
        System.out.println("I am Person init()!");
    }

    //实例代码块
    {
        this.name = "bit";
        this.age = 12;
        this.sex = "man";
        System.out.println("I am instance init()!");
    }

    public void show(){
        System.out.println("name: "+name+" age: "+age+" sex: "+sex);
    }

}

public class Main {
    public static void main(String[] args) {
        Person p1 = new Person();
        p1.show();
    }
}

// 运行结果
I am instance init()!
I am Person init()!
name: bit age: 12 sex: man
```

注意事项: 实例代码块优先于构造函数执行。

6.4 静态代码块

使用static定义的代码块。**一般用于初始化静态成员属性。**

```
class Person{
```

```
    private String name;//实例成员变量
    private  int age;
    private String sex;
    private static int count = 0;//静态成员变量     由类共享数据   方法区

    public Person(){
        System.out.println("I am Person init()!");
    }

     //实例代码块
    {
        this.name = "bit";
        this.age = 12;
        this.sex = "man";
        System.out.println("I am instance init()!");
    }

    //静态代码块
    static {
        count = 10;//只能访问静态数据成员
        System.out.println("I am static init()!");
    }

    public void show(){
        System.out.println("name: "+name+" age: "+age+" sex: "+sex);
    }

}

public class Main {
    public static void main(String[] args) {
        Person p1 = new Person();
         Person p2 = new Person();//静态代码块是否还会被执行?
    }
}
```

注意事项

- 静态代码块不管生成多少个对象，其只会执行一次，且是最先执行的。
- 静态代码块执行完毕后，实例代码块（构造块）执行，再然后是构造函数执行。

7. 补充说明

7.1 toString方法

我们刚刚注意到，我们在把对象的属性进行打印的时候，都自己实现了show函数比如：示例8代码，其实，我们大可不必。接下来我们看一些示例代码：

代码示例：

```
class Person {
    private String name;
    private int age;
    public Person(String name,int age) {
        this.age = age;
        this.name = name;
    }
    public void show() {
        System.out.println("name:"+name+" " + "age:"+age);
    }
}

public class Main {

    public static void main(String[] args) {
        Person person = new Person("caocao",19);
        person.show();
        //我们发现这里打印的是一个地址的哈希值  原因：调用的是Object的toString方法
        System.out.println(person);
    }
}

// 执行结果
name: caocao age: 19
Person@1c168e5
```

可以使用 toString 这样的方法来将对象自动转成字符串.

代码示例:

```
class Person {
    private String name;
    private int age;
    public Person(String name,int age) {
        this.age = age;
        this.name = name;
    }
    public void show() {
        System.out.println("name:"+name+" " + "age:"+age);
    }
    //重写Object的toString方法
    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}

public class Main {

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

```
        Person person = new Person("caocao",19);
        person.show();
        System.out.println(person);
    }
}

// 执行结果
name: caocao age: 19
Person{name='caocao', age=19}
```

注意事项:

- toString 方法会在 println 的时候被自动调用.
- 将对象转成字符串这样的操作我们称为 **序列化**.
- toString 是 Object 类提供的方法, 我们自己创建的 Person 类默认继承自 Object 类, 可以重写 toString 方法实现我们自己版本的转换字符串方法. (关于继承和重写这样的概念, 我们后面会重点介绍).
- @Override 在 Java 中称为 "注解", 此处的 @Override 表示下面实现的 toString 方法是重写了父类的方法. 关于注解后面的课程会详细介绍.
- IDEA快速生成Object的toString方法快捷键：alt+f12(insert)

7.2 匿名对象

匿名只是表示没有名字的对象.

- 没有引用的对象称为匿名对象.
- 匿名对象只能在创建对象时使用.
- 如果一个对象只是用一次, 后面不需要用了, 可以考虑使用匿名对象.

代码示例:

```
class Person {
    private String name;
    private int age;
    public Person(String name,int age) {
        this.age = age;
        this.name = name;
    }
    public void show() {
        System.out.println("name:"+name+" " + "age:"+age);
    }
}

public class Main {

    public static void main(String[] args) {
        new Person("caocao",19).show();//通过匿名对象调用方法
    }
}

// 执行结果
name: caocao age: 19
```

内容重点总结

- 一个类可以产生无数的对象，类就是模板，对象就是具体的实例。
- 类中定义的属性，大概分为几类：类属性，对象属性。其中被static所修饰的数据属性称为类属性, static修饰的方法称为类方法，**特点是不依赖于对象，我们只需要通过类名就可以调用其属性或者方法**。
- 静态代码块优先实例代码块执行，实例代码块优先构造函数执行。
- this关键字代表的是当前对象的引用。并不是当前对象。

课后作业

- 编写一个类Calculator,有两个属性num1,num2,这两个数据的值，不能在定义的同时初始化，最后实现加减乘除四种运算.
- 设计一个包含多个构造函数的类，并分别用这些构造函数来进行实例化对象。
- 实现交换两个变量的值。要求：需要交换实参的值。