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. 面相对象的初步认知

修改意见：

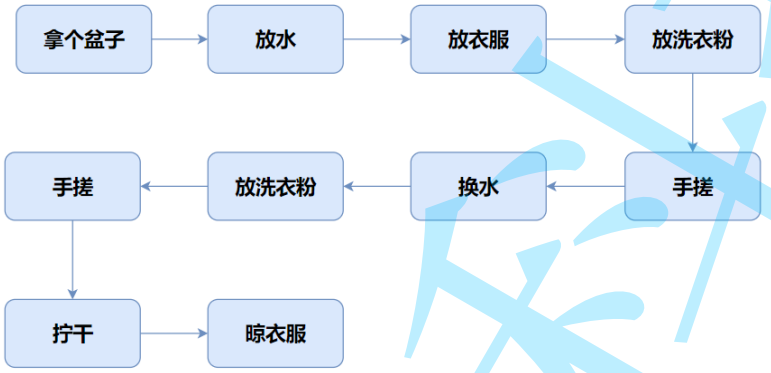
1. 先入为主的问题要，前期要过渡好，待定
2. 这里不谈面向对象和面向过程的区别，可以放在后面准备建模的时候在进行对比
3. 建议：每次在开会时候，把课件章节以及小标题顺序尽量确定下来
4. 建议：课件在写的时候，尽量不要出现太多的文字，讲的时候不好讲
5. 建议：每个知识点在讲的时候最好要聚焦，不要太分散，学生总结能力都不是很好，后期可能面试的时候回答的就不全面
6. 以下内容主要是在高博基础之上进行的修改

1.1 什么是面相对象

Java是一门纯面相对象的语言(Object Oriented Program，继承OOP)，在面相对象的世界里，一切皆为对象。面相对象是解决问题的一种思想，主要依靠对象之间的交互完成一件事情。用面相对象的思想来涉及程序，更符合人们对事物的认知，对于大型程序的设计、扩展以及维护都非常友好。

1.2 面相对象程序的好处

1. **传统洗衣服过程**



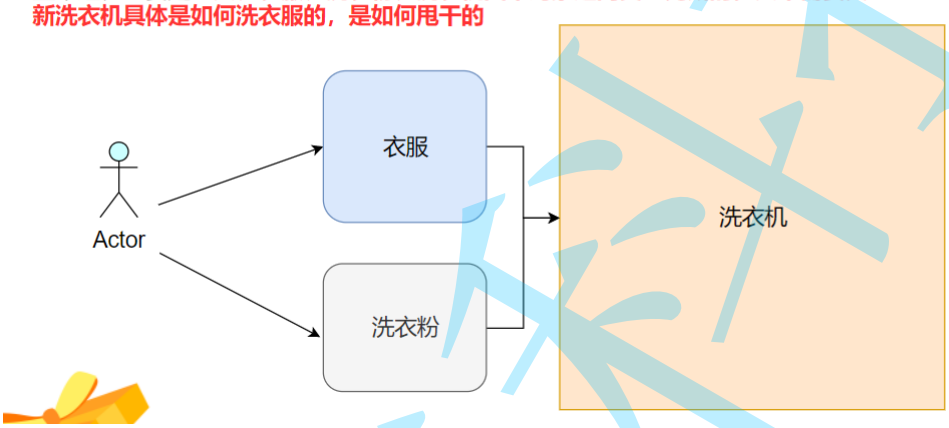
传统的方式：**注重的是洗衣服的过程**，少了一个环节可能都不行。

而且不同衣服洗的方式，时间长度，拧干方式都不同，处理起来就比较麻烦。如果将来要洗鞋子，那就是另一种放方式。

按照该种方式来写代码，将来**扩展或者维护起来会比较麻烦。**

2. **现代洗衣服过程**

总共有四个对象：**人、衣服、洗衣粉、洗衣机**，
整个洗衣服的过程：人将衣服放进洗衣机、倒入洗衣粉，启动洗衣机，洗衣机就会完成洗衣过程并且甩干
整个过程主要是：**人、衣服、洗衣粉、洗衣机四个对象之间交互完成的，人不需要关新洗衣机具体是如何洗衣服的，是如何甩干的**



以**面相对象方式来进行处理，就不关注洗衣服的过程**，具体洗衣机是怎么来洗衣服，如何来甩干的，用户不用去关心，只需要将衣服放进洗衣机，导入洗衣粉，启动开关即可，**通过对象之间的交互来完成的。**

以上就是面相对象的思想来处理，而在java中借助面相对象思想来处理问题，首先需要用类来描述对象。

2. 类定义和使用

修改意见：

1. 类的相关概念明确化，比如，类的概念，类的定义规则，类的使用等
2. 把类的基本语法全部写完，要多多新增例子来进行巩固

2.1 简单认识类

类主要是用来对一个实体(对象)来进行描述的，主要描述该实体(对象)具有哪些属性(外观尺寸等)，哪些工功能(用用来干啥)，描述完成后告诉给计算机的。

比如：上述的洗衣机，洗衣机就是一个实体或者称为对象。但是计算机不认识洗衣机，如果要让计算机认识洗衣机，在java中必须通过类对计算机进行描述。

洗衣机类：

属性：生产厂商，品牌，出厂日期，大小尺寸，颜色...

功能：电源开关，暂时/开始，洗衣，烘干....

2.2 类的定义格式

在java中定义类时需要用到class关键字，具体语法如下

```
// 创建类
class ClassName{
    field;//成员属性
    method;//成员方法
}
```

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

类中包含的内容称为类的成员。

变量信息主要是对类进行描述的，称之为类的成员属性或者类成员变量。

方法主要说明类具有哪些功能，称为类的成员方法。

```
class Person {
    public int age;//成员属性
    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

- **类名注意采用大驼峰定义**
- 成员属性的定义，当前写法统一为public，后期会详细解释，其实也可以是其他的访问修饰限定符。
- 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.

2.3 尝试定义一个类

2.3.1 定义一个猫咪类

```
class Cat {

    public String name;//名字
    public int age;//年龄
    public String color;//颜色
    public double weight;//体重
    public String gender;//性别
    //猫的行为，既方法
    public void barks() {
        System.out.println("mewing,mewing");
    }
    public void jump() {
        System.out.println("jump,jump");
    }
}
```

2.3.2 定义一个电脑类

```
class Computer {

    public String computerName;//品牌名称
    public String cpu;//CPU型号
    public double screenSize;//屏幕尺寸
    public int usbNumbers;//USB接口个数
    public int memorySize;//内存大小
    //可能还有其他属性可以自行添加

    public void openFiles() {
        System.out.println("打开IDEA! ");
    }

    public void shutDown() {
        System.out.println("关机! ");
    }
}
```

3. 类的实例化

3.1 什么是实例化

定义了一个类，就相当于在计算机中定义了一种新的类型，与int，double类似，只不过int和double是java语言自带的内置类型，而类是用户自定义了一个新的类型，比如上述的：Cat类和Computer类。有了这些自定义的类之后，就可以使用这些类来定义实例(或者称为对象)。

用类类型创建对象的过程，称为类的实例化，在java中才用new关键字，配合类名来实例化对象。

```
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.

3.2 类和对象的说明

1. **类只是**一个**模型**一样的东西，用来对一个实体进行描述，限定了类有哪些成员.
2. **类是一种自定义的类型**，可以用来定义变量，但是**在java中用类定义出来的变量我们成为对象**.
3. 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**
4. 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

4. 常见定义类语法错误

4.1 同一个类当中，定义的类，不能和public修饰的类，名称相同

```
class TestDemo {

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

此时会发生编译错误：TestDemo.java:4: 错误: 类重复: TestDemo

4.2 不要轻易去修改public修饰的类的名称

假设：文件名是HelloWorld。

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

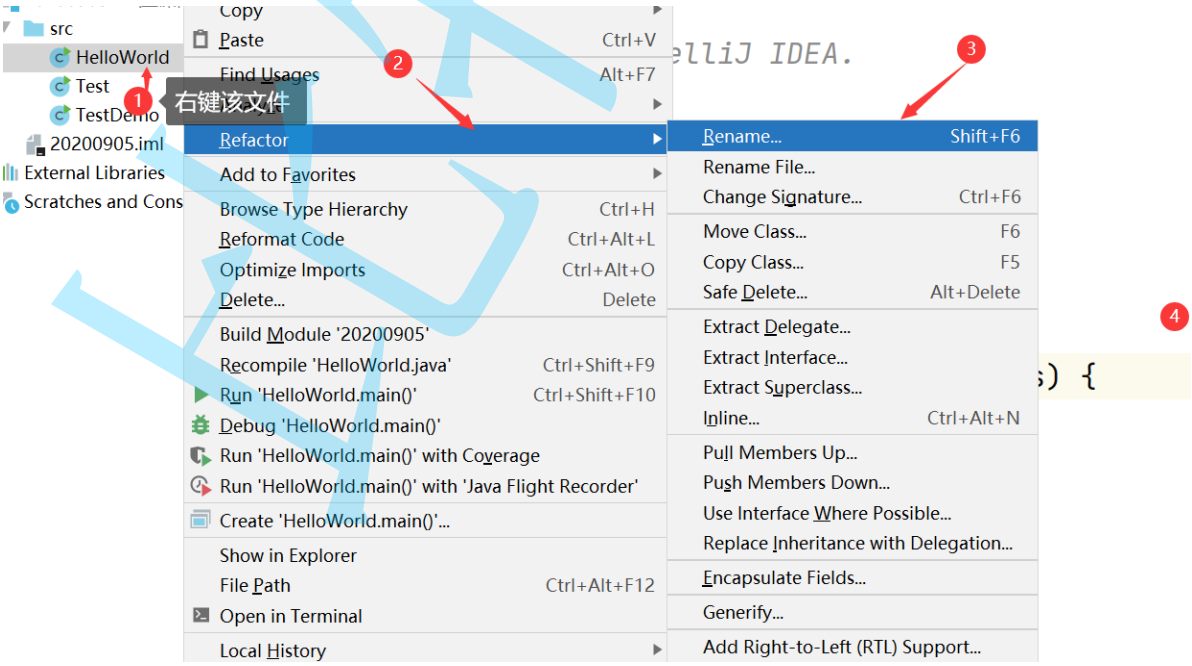
假设你觉得把这个类换个名字，然后直接手动修改为：

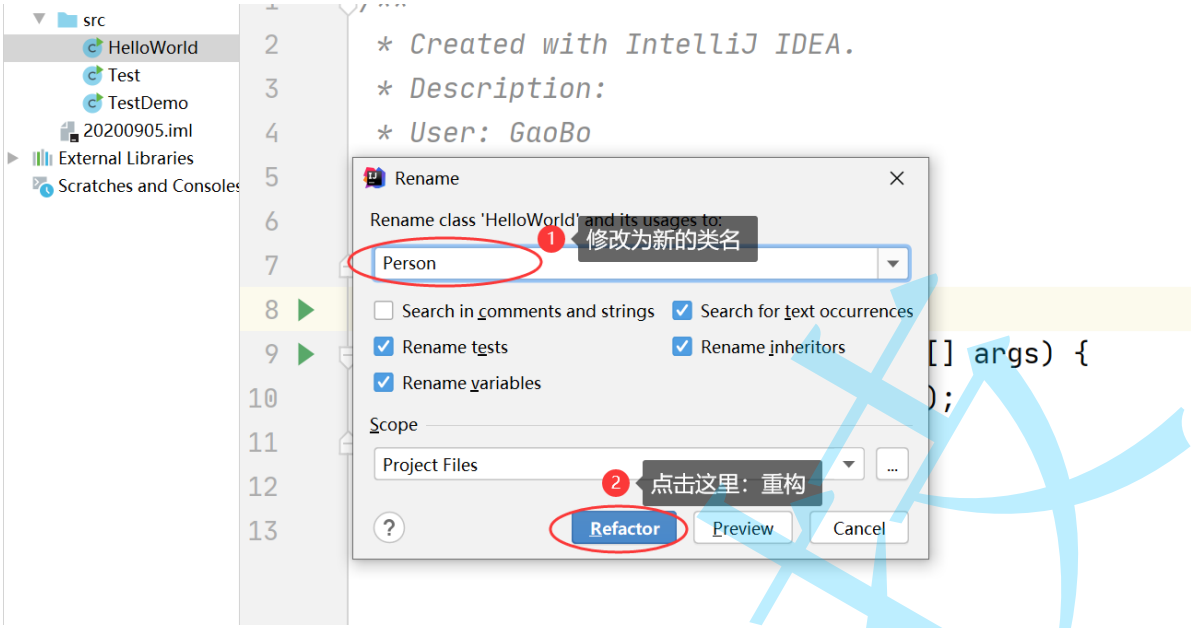
```
public class Person {
    public static void main(String[] args) {
        System.out.println("hello");
    }
}
```

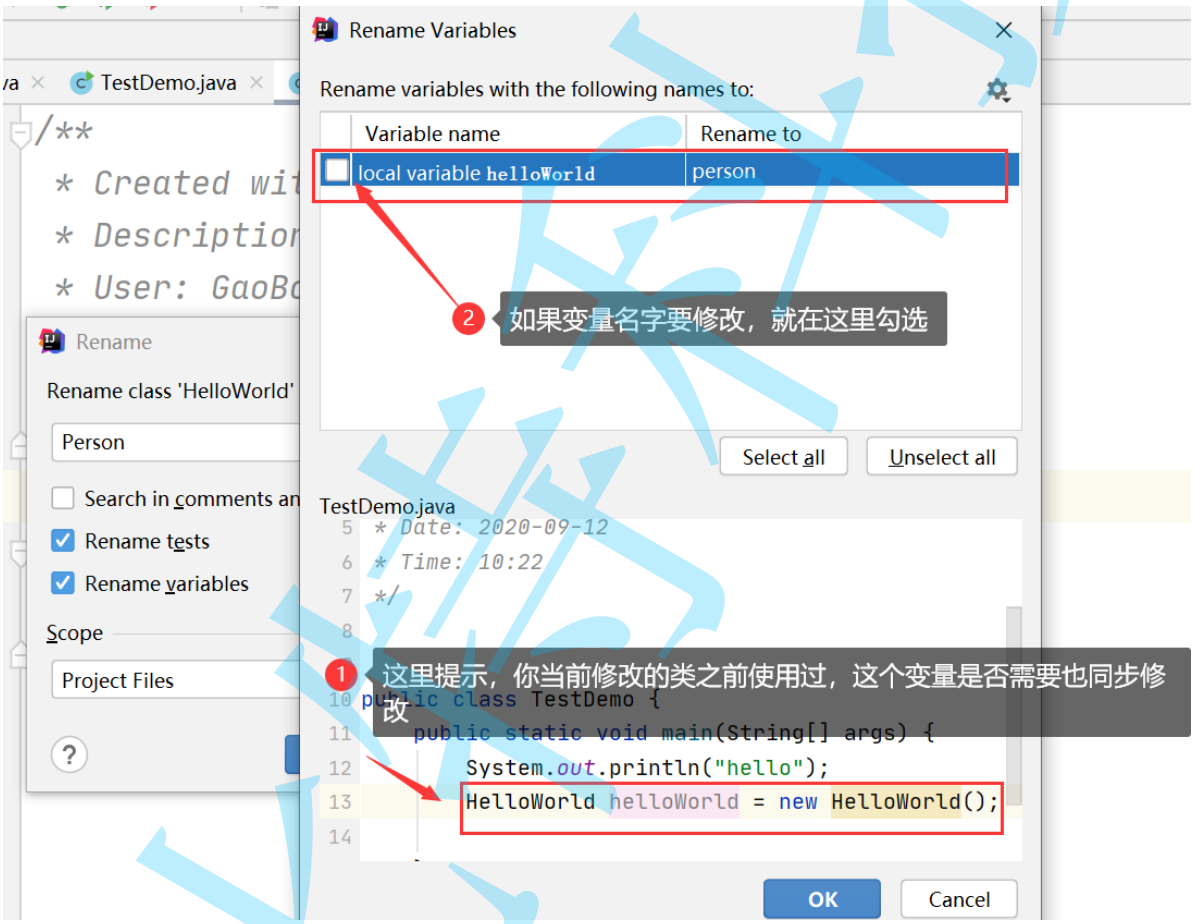
- 此时编译代码：错误: 类Person是公共的, 应在名为 Person.java 的文件中声明

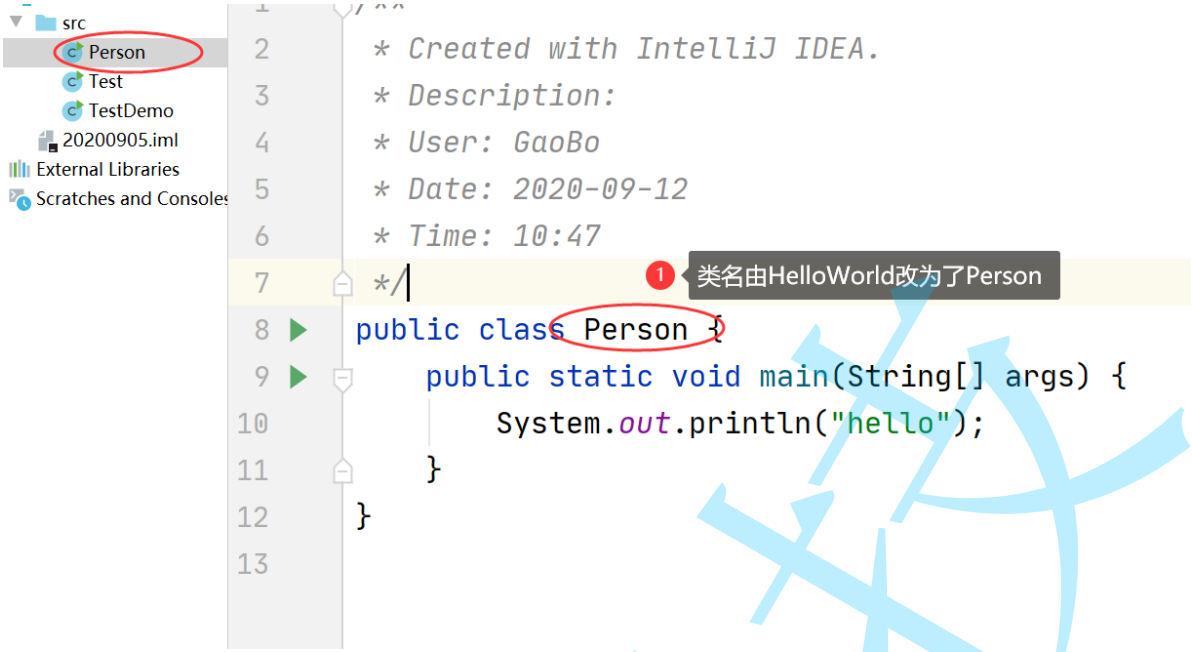
问题解决：

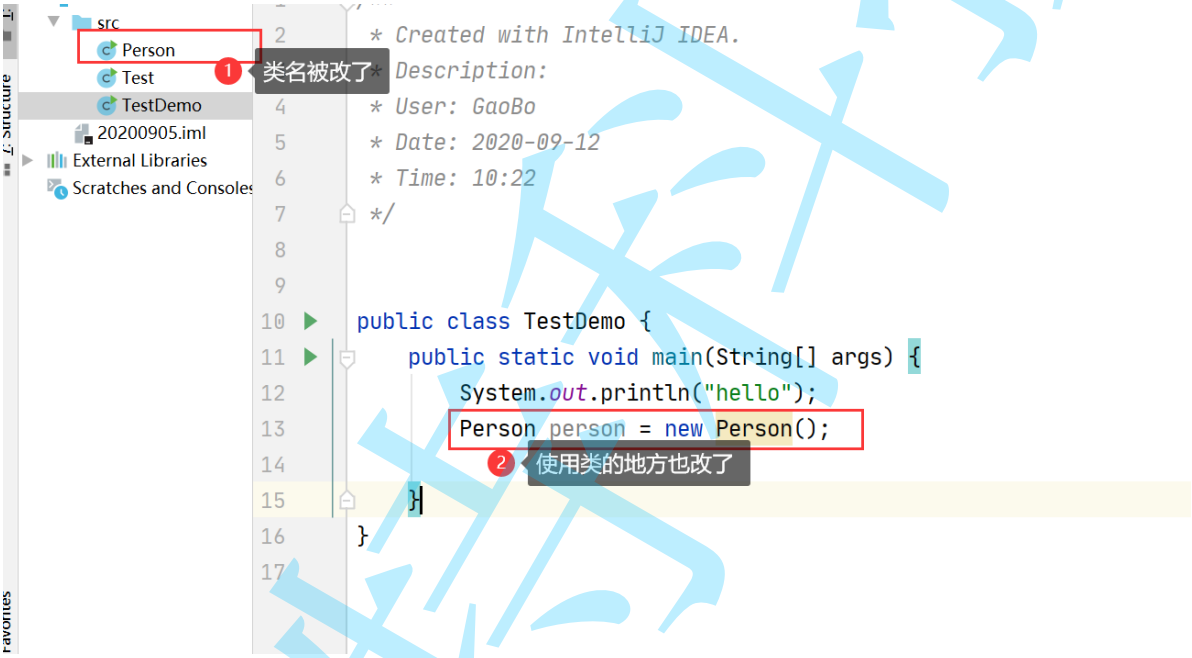
有的时候，这个类可能在被其他类使用，修改当前类名，很容易牵一发而动全身。此时若真有这样的需求，可以利用编译器来进行实现。




src
HelloWorld
Test
TestDemo
20200905.iml
External Libraries
Scratches and Consoles
* Created with IntelliJ IDEA.
* Description:
* User: GaoBo
Rename
Rename class 'HelloWorld' and its usages to:
1 修改为新的类名
Person
Search in comments and strings
Search for text occurrences
Rename tests
Rename inheritors
Rename variables
Scope
Project Files
2 点击这里：重构
Refactor
Preview
Cancel
[] args) {
);


TestDemo.java
/**
* Created wit
* Description
* User: GaoBo
Rename
Rename class 'HelloWorld'
Person
Search in comments an
Rename tests
Rename variables
Scope
Project Files
Rename Variables
Rename variables with the following names to:
Variable name
Rename to
local variable helloWorld
person
2 如果变量名字要修改，就在这里勾选
Select all
Unselect all
TestDemo.java
* Date: 2020-09-12
* Time: 10:22
*/
1 这里提示，你当前修改的类之前使用过，这个变量是否需要也同步修改
public class TestDemo {
public static void main(String[] args) {
System.out.println("hello");
HelloWorld helloWorld = new HelloWorld();
OK
Cancel





3. 类的成员

建议：类的成员没有必要单独列出来了，直接在定类定义的位置就提了---时亮益建议

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-----放在构造函数之前来讲，可以作为构造函数的引子----时亮益建议
>
> - 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----放在引用的位置来讲，类和对象位置不用讲---时亮益建议

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 - 需要单独摘录出来

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

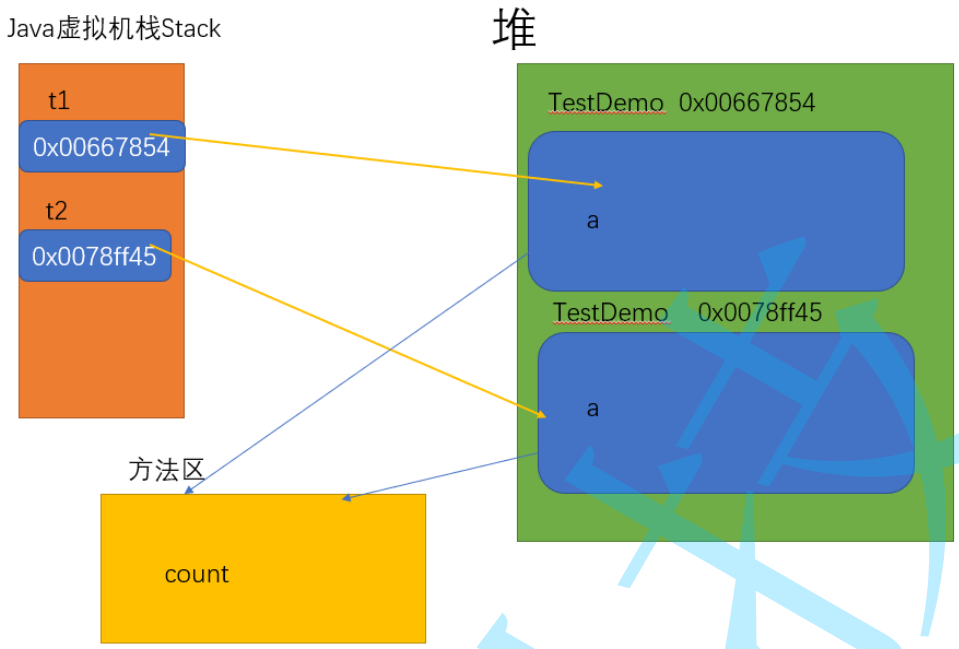
a) 修饰属性，Java静态属性和类相关, 和具体的实例无关. 换句话说, 同一个类的不同实例共用同一个静态属性.

```
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

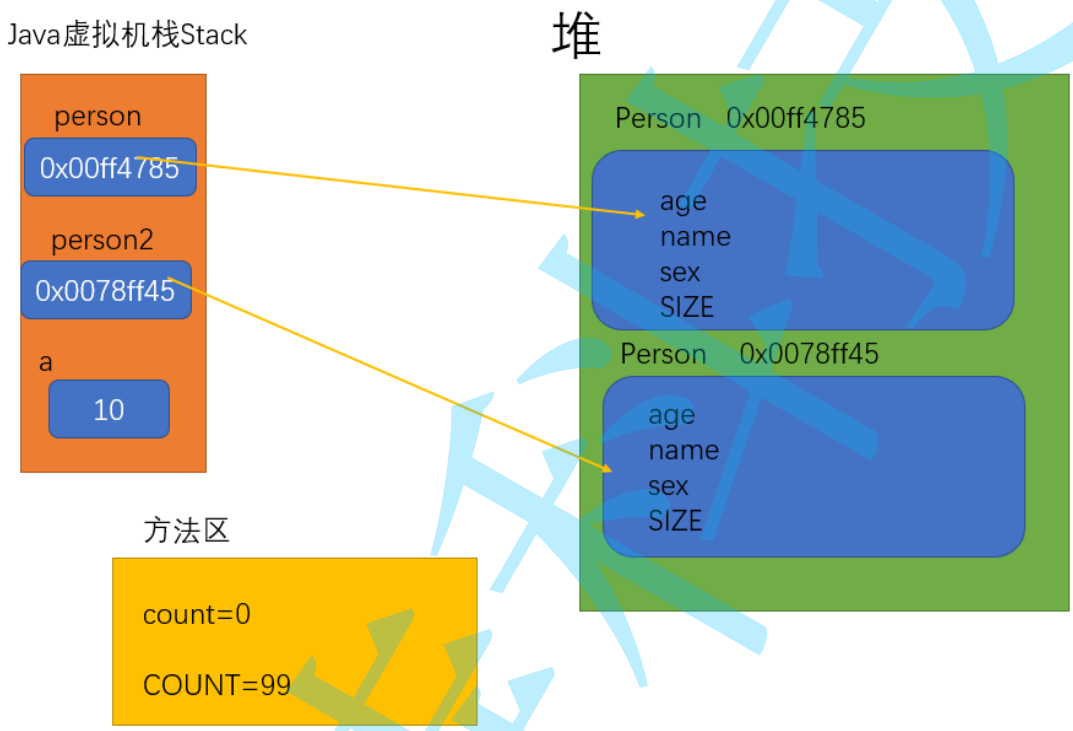
观察以下代码, 分析内存布局.

```
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. this指针

4.1 为什么要有this指针

先看一个日期类的例子：

```java
public class Date {
    public int year;
    public int month;
    public int day;

    public void setDay(int y, int m, int d){
        year = y;
        month = m;
        day = d;
    }

    public void printDate(){
        System.out.println(year + "/" + month + "/" + day);
    }

    public static void main(String[] args) {
        // 构造三个日期类型的对象 d1 d2 d3
```

```
        Date d1 = new Date();
        Date d2 = new Date();
        Date d3 = new Date();

        // 对d1，d2，d3的日期设置
        d1.setDay(2020,9,15);
        d2.setDay(2020,9,16);
        d3.setDay(2020,9,17);

        // 打印日期中的内容
        d1.printDate();
        d2.printDate();
        d3.printDate();
    }
}
```

以上代码定义了一个日期类，然后main方法中创建了三个对象，并通过Date类中的成员方法对对象进行设置和打印，代码整体逻辑非常简单，没有任何问题。

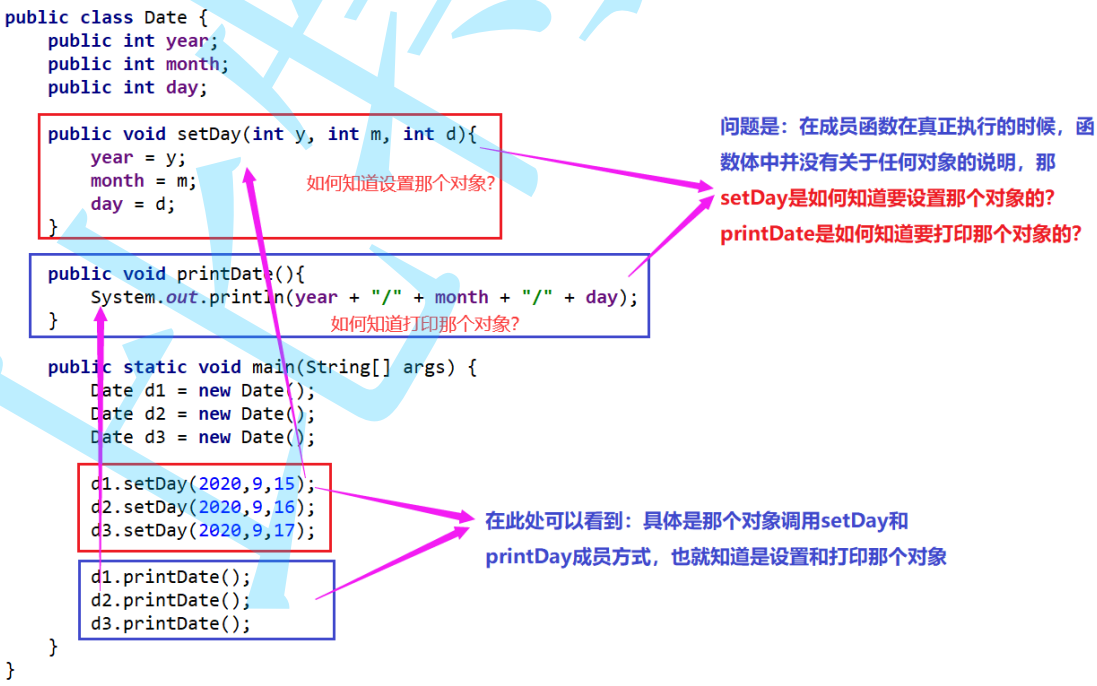
但是细思之下有以下两个疑问：

1. 形参名不小心与成员变量名相同：

   ```
   public void setDay(int year, int month, int day){
       year = year;
       month = month;
       day = day;
   }
   ```

   那函数体中到底是谁给谁赋值？成员变量给成员变量？参数给参数？参数给成员变量？成员变量参数？估计自己都搞不清楚了。

2. 三个对象都在调用setDate和printDate函数，但是这两个函数中没有任何有关对象的说明，setDate和printDate函数如何知道打印的是那个对象的数据呢？



一切让this引用来揭开这层神秘的面纱。

4.2 什么是this引用

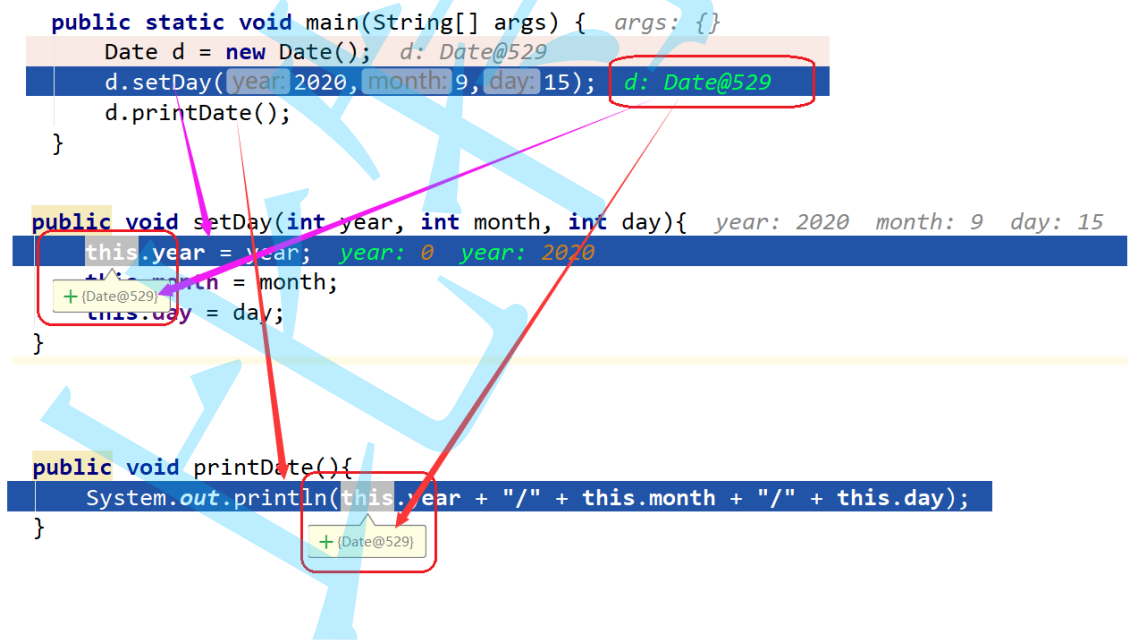
java编译器给每个“成员方法”增加了一个隐藏的引用类型参数，让该引用参数指向当前对象(成员方法运行时调用该成员方法的对象)，在成员方法中中所有成员变量的操作，都是通过该引用去访问。只不过所有的操作对用户是透明的，即用户不需要来传递，编译器自动完成。

```
public class Date {
    public int year;
    public int month;
    public int day;

    public void setDay(int year, int month, int day){
        this.year = year;
        this.month = month;
        this.day = day;
    }

    public void printDate(){
        System.out.println(this.year + "/" + this.month + "/" + this.day);
    }
}
```

this引用是编译器自动添加的，用户在实现代码时一般不需要显式给出。

注意：**this引用的是调用成员方法的对象。**

```
public static void main(String[] args) {
    Date d = new Date();
    d.setDay(2020,9,15);
    d.printDate();
}
```



4.3 this引用的特性

1. this引用的类型：对应类类型引用，即那个对象调用就是那个对象的引用类型
2. this引用只能在"成员方法中使用"
3. this引用具有final属性，在一个成员方法中，不能再去引用其他的对象
4. this引用是成员方法第一个隐藏的参数，编译器会自动传递，在成员方法执行时，编译器会负责将调用成员方法对象的引用传递给该成员方法，this引用负责来接收
5. 在成员函数中，所有成员变量的访问，都会被编译器修改成通过this来访问

大家思考下：this引用可以为空吗?

5. 对象的初始化和构造方法

5.1 对象的初始化

5.2 默认初始化

5.3 就地初始化

5.4 构造方法

5.4.1 构造方法的概念

5.4.2 构造方法的特性

6. static成员

6.1 static修饰成员变量

6.2 static修饰成员方法

7. 封装

7.1 封装的概念

7.2 java中如何实现封装

7.3 访问限定符

8. toString方法

4.1 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() {
        name = "caocao";
        age = 10;
        sex = "男";
    }
    //带有3个参数的构造函数
    public Person(String myName,int myAge,String mySex) {
        name = myName;
        age = myAge;
        sex = mySex;
    }

    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
```

4.2 this关键字

修改意见:

1. 分3部分

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（普通方法和构造方法）.

4.2.1 通过this调用类中的属性

```
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 注意此时形参的命名和属性的命名一样
    public Person(String name,int age,String sex) {
        this.name = name;
        this.age = age;
        this.sex = sex;
    }

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

4.2.2 通过this调用类中的方法

```
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.name = "caocao";
        this.age = 10;
        this.sex = "男";
    }
    //带有3个参数的构造函数 注意此时形参的命名和属性的命名一样
    public Person(String name,int age,String sex) {
        this.name = name;
        this.age = age;
        this.sex = sex;
    }

    public void eat() {
        System.out.println(this.name+" 正在吃饭! ");
    }

    public void show(){
        System.out.println("name: "+this.name+" age: "+this.age+" sex:
"+this.sex);
        //在show方法当中，通过this引用来调用本类的eat方法
        this.eat();
    }

}
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
caocao 正在吃饭!
name: zhangfei age: 80 sex: 男
zhangfei 正在吃饭!
```

4.2.3 通过this调用自身的构造方法

```
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。

注意事项总结：

- 场景：需要在一个构造方法当中，调用当前类的另外一个构造方法的时候，通过this()的形式调用。
- 必须放在第一行，且只能调用一个

```
public Person() {
    this("caocao",12);
    //this("wukong",500,"man"); 此时程序编译错误，当前类只能调用1个构造方法
}
//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;
}
//带有2个参数的构造函数
public Person(String name,int age ) {
    this.name = name;
    this.age = age;
}
```

- 使用this调用构造方法的时候，只能在构造函数当中使用，不能再普通方法当中使用。

5. private关键字

修改意见：

1. 封装放在设计里面
2. private放在后面
3. 建议：封装特性还是放在这个课件中讲，都属于语法行列，下个章节就直接使用了

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 修饰属性, 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();
        //当属性被private修饰之后，类外就不可以访问了。
        //System.out.println("我叫" + person.name + ", 今年" + person.age + "岁");
        person.show();
    }
}

// 执行结果
我叫张三, 今年18岁
```

- 当属性被private修饰之后，类外不可以直接进行访问。 **（后面的课件会讲到访问方式）**
- 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.

6. 补充说明

6.1 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 综合练习

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。**
- 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。