OOA：面向对象分析

OOD：面向对象设计

OOP：面向对象编程

**面向对象三大特征：**

**封装性**

**（private):**让内部操作对外部不可⻅（对象不能直接操作属性）

对私有属性进行访问时，需要使用getter及setter方法（setName() / getName())

快捷键：Alt+insert

**继承性（只能继承一个类）**

子类会继承父类除构造方法之外所有的，进行重写。

子类可以进行功能的扩充。

子类对象在进行实例化前⼀定会首先实例化父类对象。先调用父类的构造方法后再调⽤子类构造方法（隐含的super()调用)。

如果父类没有无参构造，那么子类构造必须使⽤**super**明确指出使用父类哪个构造⽅法。

**多态性（方法覆写）**

**方法的多态性：**

①方法的重载：同⼀个方法名称可以根据参数的类型或个数不同调⽤不同的方法体

②方法的覆写：同⼀个方类的方法，可能根据实例化子类的不同也有不同的实现。

**对象的多态性**【抽象类和接口、继承才能体会到实际用处】（前提：方法覆写）：

①对象的向上转型：父类引用 引用子类对象，此时父类引用只能访问父类属性和方法

发生时机：直接赋值、传参、返回值

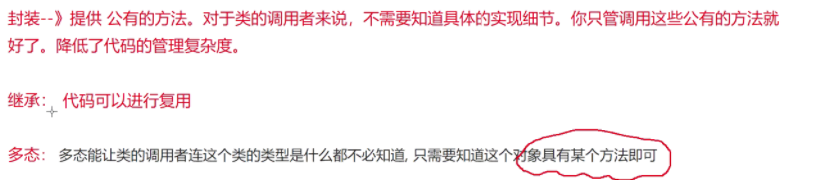
父类 父类对象 = 子类实例。

②对象的向下转型（不安全）：

子类 子类对象 =（子类）父类实例。

**注意：为什么我们需要向下转型？当你需要子类的扩充功能时候**

**向下转型之前需要向上转型，否则在转型时会出现 ClassCastException 。需要进行判断（instanceof)**

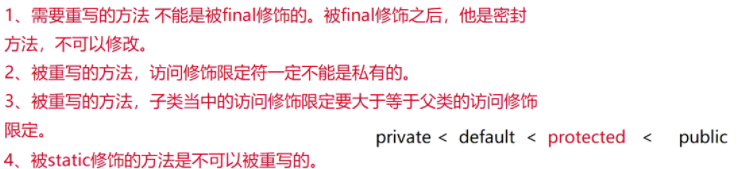


**重写（override）：发生在有继承关系的类之间**

方法的覆写：子类定义了与父类方法名称、参数类型及个数完全相同的方法。但不能够拥有比父类更为严格的访问控制权限（private<default<public）。

如果现在父类方法使⽤private定义，子类中使⽤public覆写，对吗？

答：如果现在父类方法使用了private定义，那么就表示该方法只能被父类使⽤，子类无法使用



**重载与重写区别：**



**类与对象**

**类与对象**：类相当于设计图纸，对象为对象实体

**类的实例化：**一个类可以实例多个对象

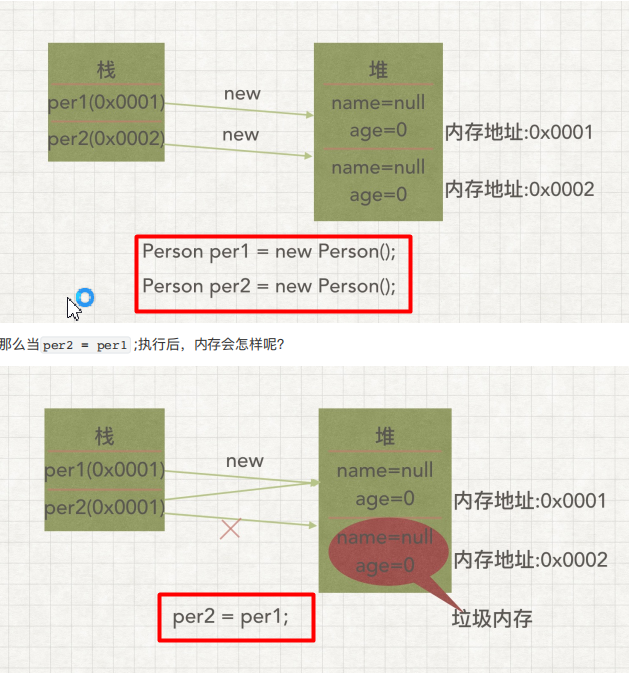
**对象的实例化：**

对象必须在实例化后调用，否则会产⽣ **NullPointerException** （运行错误）

Person per=new Person();只是分配了内存，并没有赋值

实例成员（属性）没有初始化时，默认值为对应的0值。

Person per1=new Person();
Person per2=new Person();
per2=per1;



**局部变量**：方法内部

**成员变量（实例/静态）**：类内部方法外部

静态成员变量存放于方法区中，实例成员变量存放于堆区

静态成员变量只有一份，实例成员只要new 一次啊 就产生一次

访问静态成员变量/方法都是类名.变量/方法

静态方法内部不可以访问非静态成员，实例方法可以访问静态或者非静态成员

**构造方法（可重载）：**就是使用关键字new实例化新对象后来进行调用的操作⽅法。

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.每⼀个类中至少存在⼀个构造方法（没有定义，则系统自动生成成⼀个无参构造）

4.快捷键:Alt+insert

**代码块：静态>实例>构造方法**

**如果都是静态的，执行顺序与定义顺序一致**

实例/普通代码块：

{
}

静态代码块：

静态块优先于构造块执⾏。

无论产生多少实例化对象，静态块都只执行⼀次。

不能访问非静态数据成员

在主类中的静态代码块优先于main方法

static{
}

构造代码块：

定义在类中，优先于构造方法执行，每产⽣⼀个新的对象就调⽤⼀次构造块

public class main{
{
}
}

**内部类：**

1. 内部类⽅法可以访问该类定义所在作⽤域中的数据

2. 内部类可以对同⼀包中的其他类隐藏起来

3. 内部类可以实现 java 单继承的缺陷，实现“多继承”

**内部类与外部类关系：**

对于非静态内部类，**内部类的创建依赖外部类的实例对象**，在没有外部类实例之前是无法创建内部类的

内部类是⼀个相对独⽴的实体，与外部类不是is-a关系

**内部类可以直接访问外部类的元素(包含私有域)**，但是外部类不可以直接访问内部类的元素

**外部类可以通过内部类引⽤间接访问内部类元素**

**在内部类创建内部类对象：**

外部类.内部类 内部类对象 = new 外部类().new 内部类();
Outter.Inner in = new Outter().new Inner();

**在外部类内部创建内部类对象：**

在外部类内部创建内部类，就像普通对象⼀样直接创建
Inner in = new Inner();

**成员内部类**

成员内部类中不能存在任何static的变量和⽅法

成员内部类是**依附于外围类**的，只有先创建了外部类才能够创建内部

**静态内部类**

静态内部类的创建是**不需要依赖于外围类**，可以直接创建

外部类.内部类 内部类对象 = new 外部类().内部类();
Outter.Inner in = new Outter().Inner();

**局部/方法内部类**

局部内部类不允许使⽤访问权限修饰符 public private protected 均不允许

局部内部类只允许创建这个类的⽅法访问

局部内部类要想使用方法形参，该形参必须⽤fifinal声明

class Outter {
private int num;
public void display(int test) {
class Inner {
private void fun() {
num++;
System.out.println(num);
System.out.println(test);
}
}
new Inner().fun();
}
}
public class Test {
public static void main(String[] args) {
Outter out = new Outter();
out.display(20);
}
}

**4.匿名内部类**

没有访问修饰符的。

必须继承⼀个抽象类或者实现⼀个接⼝

不能存在任何静态成员或⽅法

没有构造⽅法的，因为它没有类名。

可以引⽤⽅法形参，此形参也必须声明为 final