# 随机点名案例

功能需求:

存储学生

预览学生

随机点名

分析:

1. 创建学生类

学号

/ 姓名 / 年龄

2） 将学生进行存储

3） 将存储的学生进行打印

4） 随机获取

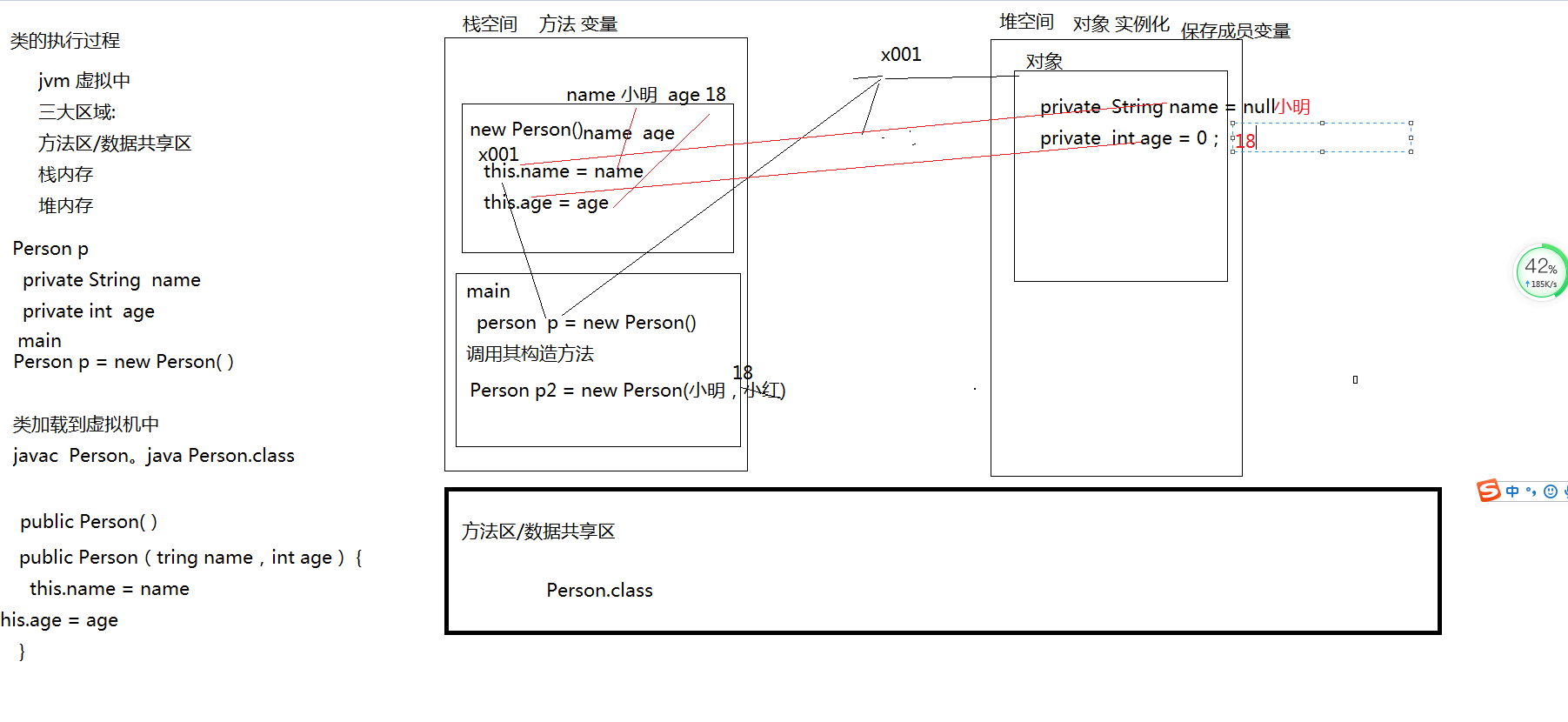
# 封装性

面向对象的第一大特性；将属性私有保护起来 对外不公开 好处，不可以在外面随意更改属性值

如何访问到属性呢:

对封装的属性 对外提供 共有的方法 利用方法为属性赋值

# 类的执行过程



# 构造的创建过程

例如:

class Person{

public Person(){

}

}

Person p = new Person();

构造函数查找过程

创建对象，找该类中是否有构造函数，如果没有构造函数的话，jvm会为这个类自动创建一个无参数的构造函数

如果这个类中存在了构造函数，那么系统就不会帮助我们再去创建构造函数

构造函数的作用:

1. 创建对象 new 构造函数的名字
2. 为成员变量赋值

# 继承

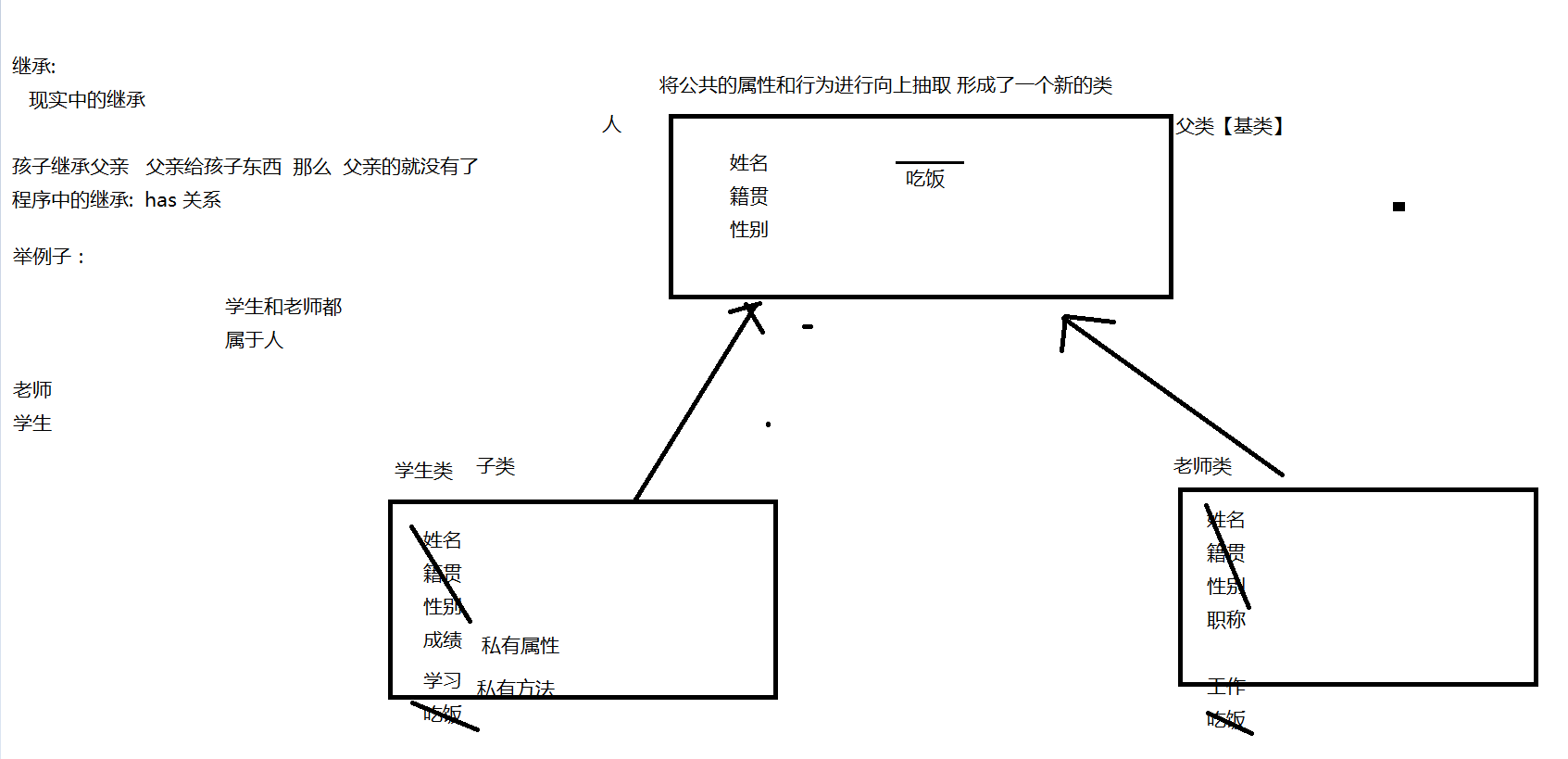
子类的公共的属性/方法 向上抽取成为一个父类

子类享有父类中的所有的属性和方法

语法

权限修饰符 class 子类 extends 父类 {

}



继承写法:

## 基类

public class Person {  
  
 public String name;  
  
 public String address;  
  
 public int age;  
  
 public void eat() {  
 System.*out*.println("在吃饭.....");  
 }  
}

## 子类

public class Teacher extends Person {  
  
  
  
 public String job;  
  
 */\*\*  
 \* 学习  
 \*/* public void work() {  
 System.*out*.println("老师在工作......");  
 }  
  
  
  
  
}

# 继承中的注意事项

1. 继承只支持单继承

public class A extends B {

}

多继承 错误写法 不能这么写

public class A extends B,C {

}

1. 多个类可以继承同一个父类

public class A {

}

public class B extends A {

}

public class C extends A {

}

B类和C类同时继承了 A

1. 继承的传递

解决单继承的问题

错误写法

public class A extends B,C {

}

将上述代码转换为

public class A extends B {

}

public class B extends C {

}

综上 A 的对象就拥有 B 和C 中的所有的属性和方法

# 继承中属性和函数的查找规则

就近原则

属性先查找 局部变量

局部变量不存在 查找当前类的成员变量

当前类的成员变量不存在 查找父类的成员变量

# 关键字

## this

this 代指当前类的对象

## super

super 代指 父类的对象

子类

*/\*\*  
 \* @Title: Son  
 \* @Description:  
 \* @Auther:  
 \* @Version: 1.0  
 \* @create 2020/6/16 0016 14:32  
 \*  
 \* 1) 类中变量的查找关系  
 \* 本着就近原则查找  
 \* 找局部变量  
 \* 局部变量不存在 找 成员变量  
 \* 成员变量不存在 找 父类  
 \* 强行查找:  
 \* 局部变量 访问变量的时候 前面什么都不用加  
 \* 成员变量 变量前面添加 this  
 \* 父级变量 变量强前面添加 super  
 \*  
 \* 方法的查找也是如此  
 \* 1） 方法中不能够嵌套方法  
 \*  
 \*  
 \*/*public class Son extends Father{  
  
 //public String name = "我是子类";  
  
 public void fun(String n) {  
 System.*out*.println(name);  
 // this 本类对象  
 System.*out*.println(this.name);  
 // super 代指父类对象  
 System.*out*.println(super.name);  
 fun2();  
 this.fun2();  
 super.fun2();  
 }  
  
 public void fun2() {  
 System.*out*.println("我是子类方法.....");  
 }  
  
  
 public static void main(String[] args) {  
 // 1 调用子类的 fun 函数  
 Son son = new Son();  
 son.fun("我是局部变量");  
 }  
}

父类

*/\*\*  
 \* @Title: Father  
 \* @Description:  
 \* @Auther:  
 \* @Version: 1.0  
 \* @create 2020/6/16 0016 14:32  
 \*/*public class Father {  
  
 public String name = "我是父类";  
 public void fun2() {  
 System.*out*.println("我是父类方法.....");  
 }  
}

# 方法的重载

同一个类中

方法名相同

参数列表不同

与返回值和修饰符无关

# 方法重写、覆盖

在有继承关系的类中

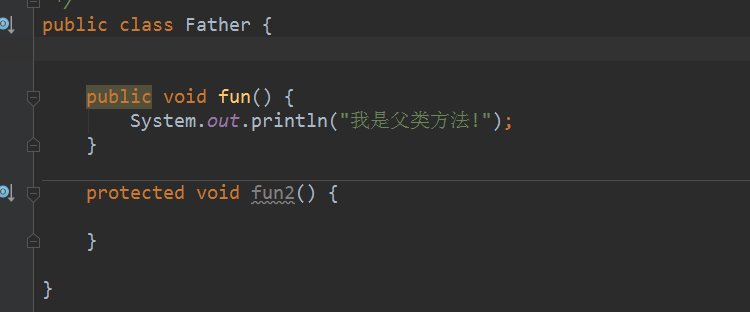
子类的方法名 /返回值类型/参数列表 和父类都一致

注意事项；

如果存在方法的覆盖 那么子类方法的权限修饰符必须要大于等于父类的权限修饰符

权限修饰符

public > protected > 默认>private

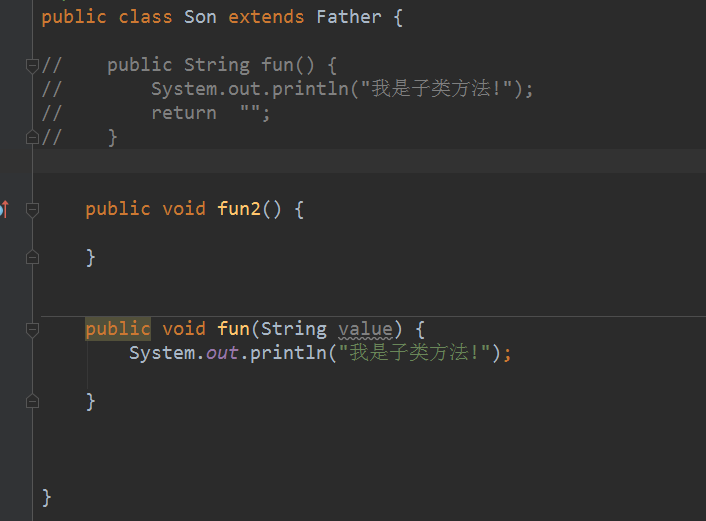


方法覆盖/重写

子类方法的权限 大于等于

父类方法的权限

公共 > 受保护> 默认 > 私有的



# 抽象

特点:

1. 抽象方法必须要存在于抽象类中
2. 抽象方法没有方法的实现，等待继承者重写该方法
3. 抽象方法不可以用private 进行修饰
4. 非抽象类继承抽象类 必须要重写抽象类中所有的抽象方法
5. 抽象类继承抽象类 可以不用重写 其中的抽象方法
6. 抽象类不可以被实例化
7. 抽象类中可以有非抽象方法，抽象方法不可以存在于普通类中

即为:综上所述:

之所以要定义抽象类 就是等待别人继承的

之所有要定义抽象方法 就是要等待继承的类 去重写该方法

重要一点:

开发可以不使用 抽象类和抽象方法

但是 抽象类和抽象方法可以对程序有良好的设计作用

作业：

抽象的代码 写一遍

银行案例【写完】

扩展功能: 每一次取款都记录一个日志， 日志最多能存储3条数【记录近期的3条取款日志】

打印取款日志

继承案例

某IT公司有多名员工，按照员工负责的工作不同，进行了部门的划分（研发部员工、维护部员工）。研发部根据所需研发的内容不同，又分为JavaEE工程师、Android工程师；维护部根据所需维护的内容不同，又分为网络维护工程师、硬件维护工程师。

公司的每名员工都有他们自己的员工编号、姓名，并要做它们所负责的工作。

* 工作内容
  + JavaEE工程师：员工号为xxx的 xxx员工，正在研发淘宝网站
  + Android工程师：员工号为xxx的 xxx员工，正在研发淘宝手机客户端软件
  + 网络维护工程师：员工号为xxx的 xxx员工，正在检查网络是否畅通
  + 硬件维护工程师：员工号为xxx的 xxx员工，正在修复打印机

请根据描述，完成员工体系中所有类的定义，并指定类之间的继承关系。进行XX工程师类的对象创建，完成工作方法的调用。

现有交通工具体系，总类为交通工具，交通工具分为汽车，船只，飞机，而汽车又分为轿车和公交车，

每个类中都有show方法，说明自己是什么交通工具，