继承

继承的特点

1. 子类可以继承父类的属性和行为，但是子类不能继承父类的构造器。
2. Java是单继承模式：一个类只能继承一个直接父类。 单根性
3. Java不支持多继承、但是支持多层继承。 传递性
4. Java中所有的类都直接或间接继承自Object类。
5. 继承具有侵入性.

super关键字

> 在任意一个除Object 以外的类的构造器中 第一行非注释性代码 必定是 super(); 如果没有显式的编写该代码 将由编译器进行补充

super()

在子类的构造方法中 调用父类构造方法 实例化子类对象时 子类构造先调用 后执行 父类构造 后调用先执行

必须注意的面试题

```
package com.hxzy;
//父类
public class Person {
    static {
        System.out.println("Person1");
    }
    public Person(){

        //自己的属性初始化操作
        System.out.println("Person");
    }
    {
        System.out.println("Person2");
    }
}
// 子类
//一个类不显式继承自其他类时 它自动继承自Object
public class Student extends Person {

    {
        System.out.println("Student1");
    }
    static {
        System.out.println("student2");
    }
    public Student(){
        //super();
        //初始化对象的属性值
        System.out.println("Student");
    }
```

```
    public static void main(String[] args) {
        //子类静态代码块>     父类静态代码块>父类代码块>父类构造方法     子类代码块 > 子类构造
器
        Student stu=new Student();
    }
}
```

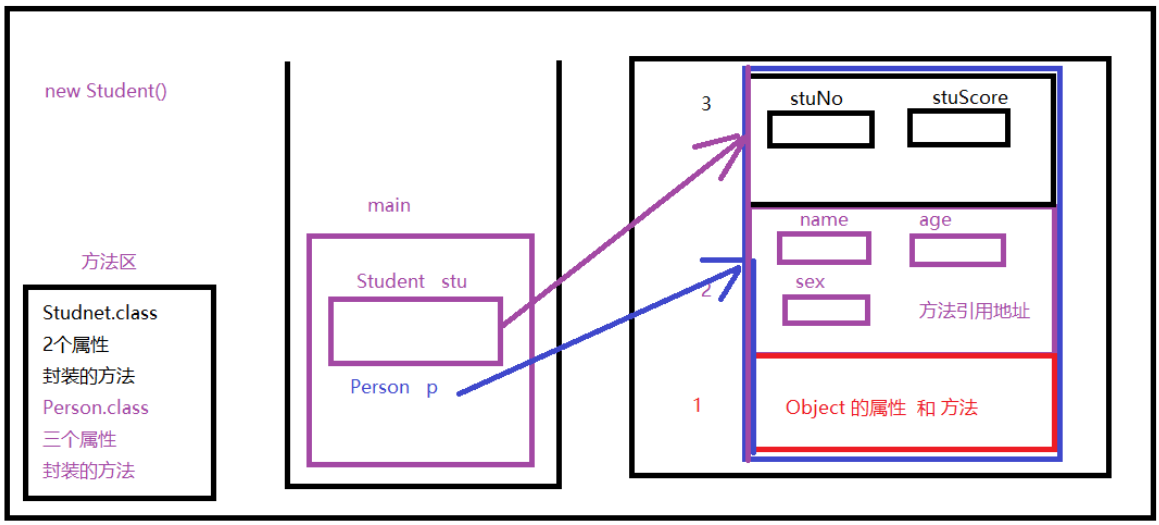
> 默认子类构造调用的都是父类的无参构造 super() 如果想要调用父类的带参构造 则需要显式的代码进行控制 此代码 必须是第一行非注释性代码 **且不能与 this() 共存**

this和super访问变量

```
public class Person {
    int num=30;
}
public class Student extends Person {
    //this 和 super 访问成员变量
    public void show(){
        System.out.println(num);//方法中找不到  本类中找不到  父类中找到了  输出30
        System.out.println(this.num);//本类中找 结果没找到  继续到父类中找 找到了 输出30
        System.out.println(super.num);//直接去父类中找 找到了 输出30
    }
}
```

```
public class Person {
    int num=30;
}
public class Student extends Person {
    int num=20;
    //this 和 super 访问成员变量
    public void show(){
        int num=10;
        System.out.println(num);//10
        System.out.println(this.num);//20
        System.out.println(super.num);//30
    }
}
```

> ***类型1 对象名称= new 类型2(); 类型1 可以和类型2一致 或 满足 类型1是类型2的直接或间接父类 类型2决定了生成的对象 包含哪些成员 类型1 决定了对象可以直接访问哪些成员***




```
public class Manager {
    ArrayList<Student> arr1=new ArrayList();
    ArrayList<Teacher> arr2=new ArrayList();
    public void add(Student stu){

    }
    public void add(Teacher tea){

    }
    //修改删除等方法都需要根据不同的子类类型设计多份
}
```

> 如果 使用子类类型 作为 存储空间的数据类型 和方法参数/返回值的类型 会导致 存储空间的数量 方法的数量 成倍的增加 整个程序不便于扩展和维护(每添加一个子类 就需要额外创建一个新的存储空间 及 一套新方法)

应该选择父类类型作为存储空间的类型声明 作为方法的参数和返回值类型 可以提高程序的可扩展性 易维护性

```
public class Manager {
    //父类类型集合 可以存储各种子类类型的对象
    ArrayList<Person> arr=new ArrayList();
    //父类类型做参数的方法  可以接收各种子类类型对象作为参数
    public void add(Person per){

    }
}
```

装箱操作

```
//装箱操作--> 父类类型的变量 存储子类类型的对象
Person p=new Student();
p.父类属性 父类方法 可以访问
```

> 装箱后 对象只能访问 父类 和 父类的父类中所包含的 成员 子类的特有成员无法直接访问

> 装箱后 如果需要访问子类特有的成员 则需要进行拆箱操作

拆箱操作

1. 需要对已经装箱的对象进行类型推断 **instanceof**

```
if(对象 instanceof 子类类型){
    //拆箱操作
}
Person p=new Student();//装箱
if (p instanceof Student){//类型推断
    System.out.println(((Student) p).getStuNo());//拆箱
}
```

2. 通过强制类型转换器 将父类类型对象转为子类类型

```
((Student) p).getStuNo();
```

方法的重写 override

方法重载 overload 在一个类 或形成继承关系的父子了中 有两个或多个方法 它们方法名称相同 参数列表不同 形成方法重载 参数的数量 类型 或顺序

> 继承+装箱+方法重写 实现了 多态

方法重写的概念(必背)

在形成继承关系的父子类间 子类拥有和父类 同名 同参数列表 同返回值类型的 且访问权限修饰符 不低于父类方法访问权限修饰符的方法 子类方法的主体 对父类方法主体进行覆盖的过程 称为 方法重写

```
public class Bird {
    //属性 和 封装 构造都省略

    public void move(){
        System.out.println("挥动翅膀!飞翔在天空中!");
    }
}
//麻雀
public class Sparrow extends Bird {
    //没有重写 继续沿用父类方法
}
//老鹰
public class Eagle extends Bird {
    //没有重写 继续沿用父类方法
}
//企鹅
public class Penguin extends Bird {
    //重写了父类的运动方法
    @Override
    public void move() {
        System.out.println("在冰面上滑行前进!");
    }
}

public class Test {
    public static void main(String[] args) {
        Bird b1=new Sparrow();
```

```
        Bird b2=new Eagle();
        Bird b3=new Penguin();
        b1.move();//父类方法
        b2.move();//父类方法
        b3.move();//子类重写的父类方法
    }
}
运行结果:
挥动翅膀!飞翔在天空中!
挥动翅膀!飞翔在天空中!
在冰面上滑行前进!
```

> 重写方法 无需使用 拆箱 就可以直接使用父类类型的变量访问到
>
> 子类无法重写父类的静态方法和私有方法
>
> 私有方法子类无权限访问 所以无法注入自身的主体代码 形成覆盖 静态方法无法重写是因为 静态成员依附于类 重写的实现实在实例化子类类型对象时发生

java常用的权限修饰符

主要用于修饰类的成员

修饰符	本类其他成员	本包中的其他类	跨包的子类	跨包非子类
public 公共的	可以	可以	可以	可以
protected 受保护的	可以	可以	可以	不可
[default] 缺省的	可以	可以	不可	不可
private	可以	不可	不可	不可