回顾:

1. static 修饰符, 修饰变量=> 全局的变量

1. 方法内局部变量

作用范围： 仅限于方法内

属于方法： 栈内存

1. 成员变量

public class Teacher{

int age;

}

new Teacher().age

作用范围： 与对象一致

属于对象： 堆内存

如果命名冲突：this.成员变量名

1. static 静态成员变量 (全局变量)

作用范围： 如果没有访问修饰符的限制，所有地方都可以引用

属于类： 方法区内存

如果命名冲突：类名.全局变量名

1. 面向对象的基础

阶段1：用java代码来描述和抽象现实世界中的事物，抽象为java类

阶段2：创建类的实例（对象）, 用对象表示具体的一个事物

new 类型()

类 =》 模板 =》 印章

对象 =》 产品 =》 盖的无数个章

2.1 构造方法：

public 类名() {

}

public 类名(参数...) {

}

构造方法之间都属于重载关系：名字一样，参数的个数和类型不一样

this(参数...) 可以用来调用其他构造方法，必须是第一行

2.2 this 代表对象本身

this.成员变量

this.成员方法（）

2.3 成员变量、成员方法

都是从属于某一个对象, 不加static 修饰符

变量 描述属性

方法 描述行为

1. 封装性

public 所有其他类都可以访问

protected 本包内的所有类，以及子类

(default) 本包内的所有类

private 本类可以访问

继承 父类、子类

多态

接口 抽象类

对象创建过程中，执行顺序（成员变量、静态成员变量、构造方法、静态初始化代码块、初始化代码块，父类子类)

# 继承

把多个类公共的特征提取到父类，让这些类（子类）继承父类，子类中就可以直接使用父类共同的特性，不用自己声明了

子类 extends 父类

子类可以从父类继承成员变量，与成员方法

注意：

## 1) 继承会受访问修饰符的影响

父类中如果定义了 private 的成员变量和方法，子类不能访问

父类中的成员变量和方法是default, 同包类可以访问

父类中的成员变量和方法是protected修饰的，同包类和子类可以访问

父类中的成员变量和方法是public，子类不管在哪个包，都能访问

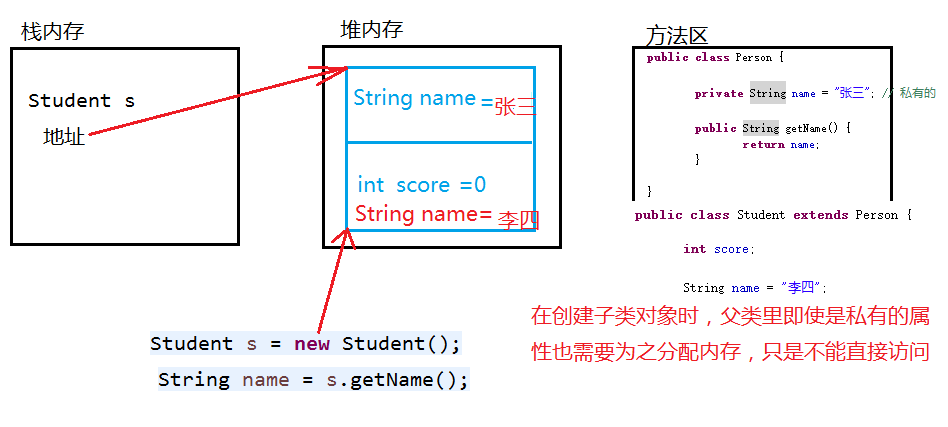
## 2) java 是单继承

子类只能继承一个父类

但整个的继承关系可以形成一个树状结构



## 3) 继承在内存上是如何体现的



## 方法的重写 (override)

是运行时多态的一个前提

区别重载 （overload) 同一个类，方法名一样，参数个数和类型不一样

子类和父类中， 方法的名字一样，并且参数的个数和类型都一样，称之为方法重写(覆盖)

Person

String getName() {

}

Student

String getName() {

}

Student s = new Student();

s.getName();

根据对象的**实际类型** 找到离他最近的重写方法来调用， 见test3包下的例子

## 5) 构造方法 ， 见test4 包下的例子

5.1 如果父类有了带参数的构造方法，子类就必须在自己的构造方法中调用父类的构造，否则就不能完成父类成员变量的初始化

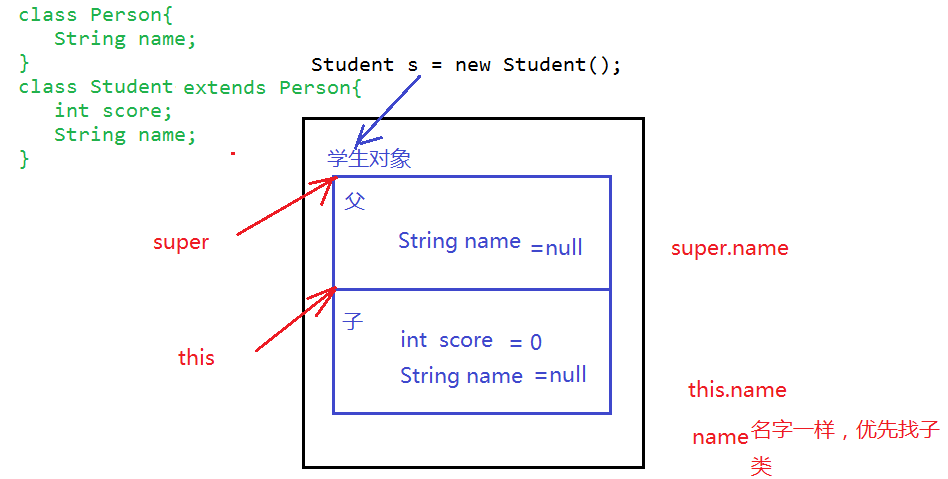
原因：

父类如果加了带参构造，默认无参构造就会消失；

子类的构造都会调用super()， 因为现在父类没有了无参构造，子类就必须显式地通过super调用父类构造方法

5.2 super() 调用父类的构造方法，必须位于第一行

## 6) 使用super 访问父类的成员方法与成员变量 见test5



## 7)方法重写时，访问修饰符的限制，见 test6

public 》 protected 》 default > private

方法重写时，子类的修饰符必须大于或等于父类的修饰符

# 对象构造的执行顺序

## 初始化代码块

代码参考(test7)

语法：

class 类名 {

public 类名() {

}

public void 成员方法() {

}

{

//初始化代码

}

}

会在构造方法之前被执行, 如果有多个初始化代码块，会按照从上到下，依次执行

## 静态初始化代码块 (test8)

语法：

class 类名 {

static {

//静态初始化代码

}

}

作用：给static 成员变量赋初值

如果有多个静态代码块，执行顺序是从上到下依次执行

如果静态变量变量本身也有赋值操作，那么也会与静态代码块一样，有从上到下的执行顺序

## 对象构造的执行顺序

Student s = new Student(); // 继承了Person父类

阶段1：类加载并初始化 (第一次用到该类时执行，只需要执行一次)

1） 检查继承关系, 加载父类字节码到jvm，将类的结构信息放入方法区

2） 给父类的静态成员变量分配空间

3） 执行父类的静态代码块和静态变量的赋值语句（从上至下）

4） 子类字节码加载到jvm 中，将类的结构信息放入方法区

5） 给子类中的所有静态成员变量分配空间 (有一个默认值)

6） 执行本类的静态代码块和静态变量的赋值语句（从上至下）

阶段2：对象的实例化阶段 (每次new 都会执行)

1）找到父类，给父类所有成员变量分配空间（默认值）

2） 找到父类初始化代码块和成员变量的赋值语句，从上至下依次执行

3） 父类构造

4) 找到子类，给子类所有成员变量分配空间（默认值）

5) 找到子类初始化代码块和成员变量的赋值语句，从上至下依次执行

6) 子类构造

# 多态(test9)

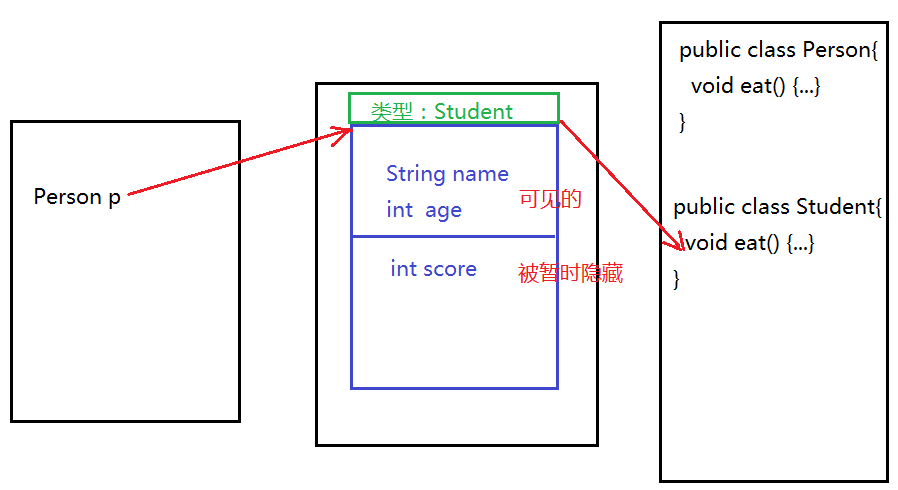
eat方法有重写的情况

Person p = new Student();

p.eat();

p这个变量调用的方法，不是根据p的类型决定的，而是根据p指向对象的实际类型来决定

java能够在创建对象时，就记住对象的实际类型



运行时多态：

在运行期间，判断出变量的实际的手段，称为运行时多态

意义：（test10）

能够以一种一致的方式来实现代码的统一调用， 提高代码的可扩展性

作业：

1. 写出下列代码的运行结果

public class ExerciseTest {

    public static void main(String[] args){

        ExerciseTest f = new ExerciseTest();

     System.out.println(f.add("4", "5"));

    }

    public int add(int x, int y) {

        return x + y;

    }

    public String add(String x,String y) {

        return x + y;

    }

}

2. 关于构造方法，下面说法正确的是

A.构造方法不能带有参数

B.构造方法的名称必须和类名相同

C.构造方法可以定义返回值

D.构造方法不能重载

3. 指出下面代码的编译错误，并说明原因

//哺乳动物

public class Mammals {

}

//鸟类

public class Birds {

}

//蝙蝠

public class Bat extends Mammals,Birds{

}

4. JVM如何管理内存，分成几个部分？分别有什么用途？说出下面代码的内存实现原理：

Foo foo = new Foo();

foo.f();

 5. 说出下面代码的输出结果，并解释原因

public class Sub extends Base {

    String color;

    public Sub(double size, String name, String color) {

        super(size, name);

        this.color = color;

    }

    public static void main(String[] args) {

        Sub s = new Sub(5.6, "测试对象", "红色");

        System.out.println(s.size + "--" + s.name + "--" + s.color);

    }

}

class Base {

    double size;

    String name;

    public Base(double size, String name) {

        this.size = size;

        this.name = name;

    }

}

6. 说出下面代码的输出结果，并解释原因

//鸵鸟

public class Ostrich extends Bird{

    public void fly(){

        System.out.println("我只能在地上奔跑...");

    }

    public static void main(String[] args) {

        Ostrich os=new Ostrich();

        os.fly();

    }

}

class Bird {

    public void fly(){

        System.out.println("我在天空里自由自在的飞翔...");

    }

}

7. 说出下面代码的输出结果，并解释原因

public class SlowPoint extends Point {

    public void move(int dx, int dy) {

        System.out.println("SlowPoint move parameter");

        move();

    }

    public static void main(String[] args) {

        SlowPoint sp=new SlowPoint();

        sp.move(10,20);

    }

}

class Point {

    public void move(int dx, int dy) {

        System.out.println("Point move parameter");

    }

    public void move(){

        System.out.println("Point move ");

    }

}

8.关于package和import语句，下面说法错误的是：

A. package 提供了一种命名机制，用于管理类名空间

B. 定义类时，除了定义类的名称以外，必须要指定一个包名

C. import语句用于导入所需要的类

D.同时使用不同包中相同类名的类，包名不能省略

9. 关于public和private，下面说法错误的是：

A.private修饰的成员变量和方法仅仅只能在本类中访问

B. public修饰的成员变量和方法可以在任何地方访问

C. private修饰的成员变量和方法可以在本类和子类中访问

D.public 修饰的成员变量和方法只能在同一个包中访问

10. 关于protected 关键字，下面说法错误的是：

A. 用protected修饰的成员变量和方法可以被子类及同一个包中的类使用

B.使用protected 关键字修饰的访问控制，即默认访问控制

C.默认访问控制的成员变量和方法可以被同一个包中的类访问

D. 用protected修饰的成员变量和方法只能被子类使用

11. 关于static 关键字，下面说法正确的是：

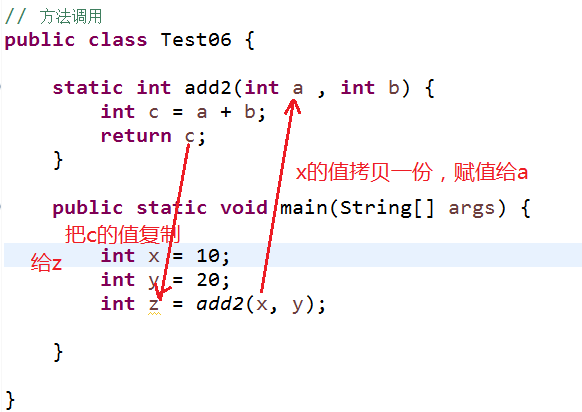
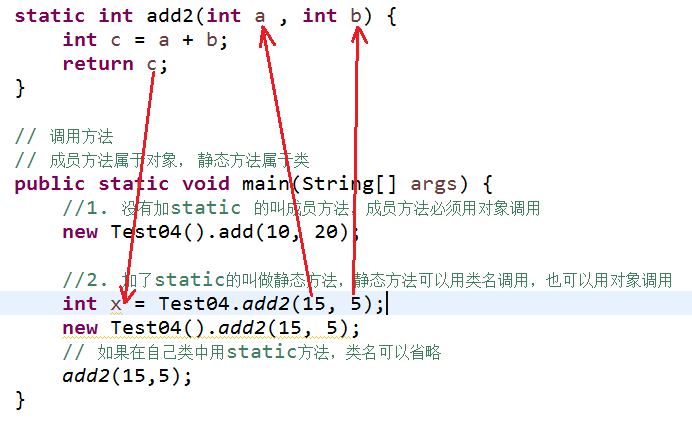
A. 用static修饰的成员变量是属于对象的数据结构

B.在static方法中，可以访问非static成员(对象成员)

C. static成员变量存储在堆中

D. 一个类的static成员变量只有“一份”，无论该类创建了多少对象

方法调用：



设计一个方法，输出的符号是变化的, 每次输出的符号个数不一样

没有结果时，返回值类型是void

第1次调用 \*\*\*\*\*\*\*

第2次调用 %%%%%

第3次调用 ####

## 方法重载与方法重写的区别？

方法重载overload：同一个类，方法名字一样，参数的个数和类型不一样

编译时多态： 在编译期间就能根据方法的参数个数和类型确定调用哪个方法

方法重写override: 父子类之间，子类方法与父类方法名字和参数都一样

覆盖

运行时多态

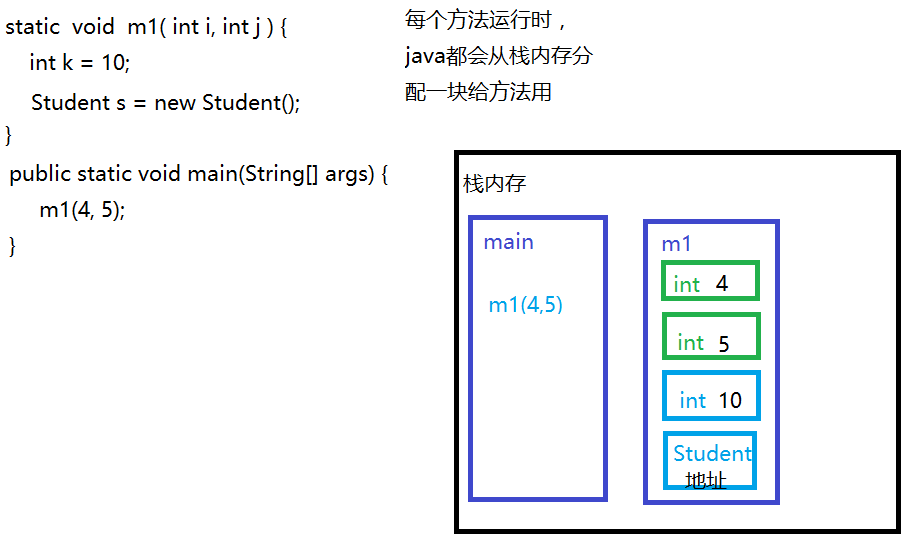
static void m1()

static void m1(int i)

m1()

m1(3)

## 方法执行时，内存的分配



### 方法区内存:

class 的定义（记录了Student、Person，有哪些属性）

以及方法的定义 （方法的代码）

static int i = 存储于方法区

static Student s = new Student(); // s 在方法区, 后面的new Student()在堆里

### 堆内存：

存储对象：（所有通过 new 关键字创建的都是对象， 存储于堆）

## 数组的定义方法：

1） 方法1：

int[] a; // 定义变量

a = new int[10]; // 分配空间， 所有元素有默认值

1. 方法2:

int[] a = new int[10];

1. 方法3：

int[] a = new int[]{1,2,3,4}; // 同时赋初值

1. 方法4：

int[] a = {1,2,3,4};

// 错误

int[] a;

a = {1,2,3,5};

1. int[] a;

int a [];

## 修改、获取数组的值

int[] a = new int[]{1,2,3}

下标值0 1 2

a[1] // 获取了2

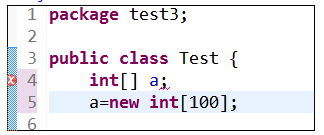
a[1] = 20; // 用20覆盖掉了2

1. length // 获取数组长度

a[3] // 出现错误

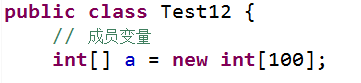
## 二维数组

## 错误:

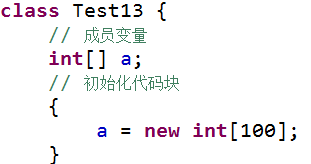


赋值，以及其他运算（+， - \* /）,如果想写在类的定义中是不对的

1）必须把赋值以及其他运算与成员变量的声明放在一起：

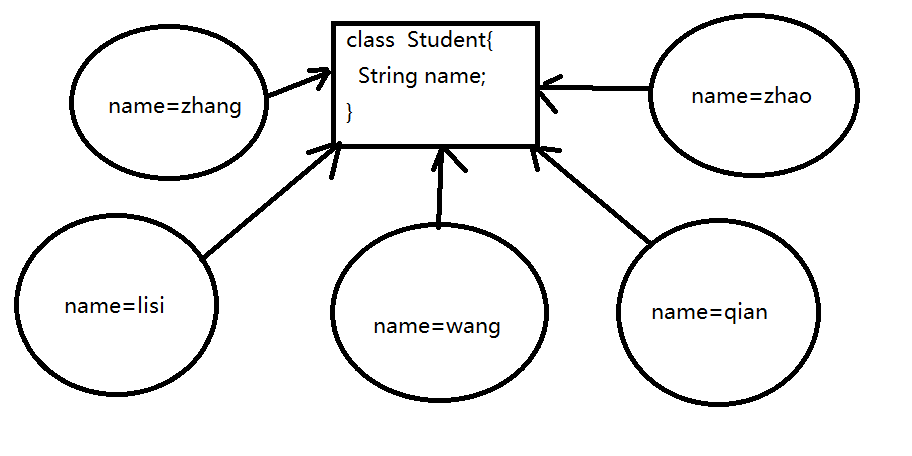


1. 或是将赋值以及其他运算放入初始化代码块



## 数组的冒泡排序

## 类与对象



## 成员变量

public class Student {

String name; // 成员变量、实例变量、实例域（field）、属性

}

## this

// 解释this

class Person {

String name;

void test() {

System.out.println(this);

}

}

public class Test15 {

public static void main(String[] args) {

Person p1 = new Person();

p1.name = "张三";

System.out.println("p1:"+p1);

// 对象的类型@对象的地址, 类型一样，地址一样，代表同一个对象

// buke.Person@6fd46259

Person p2 = new Person();

p2.name = "李四";

System.out.println("p2:"+p2);

p1.test(); // 对应第一个Person对象

p2.test(); // 对应第二个Person对象

// 谁调用了test方法, test方法内的this就代表谁

}

}

this 可以调用构造方法(只能位于构造方法内，只能位于第一行)

public class Test16 {

public Test16() {

System.out.println("1");

}

public Test16(int a, int b) {

System.out.println("3");

}

public Test16(int b) {

this();

System.out.println("2");

}

public static void main(String[] args) {

new Test16(3);

}

}

## 基本类型和引用类型在参数传递时的差异：

class Cat {

String name;

}

public class Test17 {

public static void main(String[] args) {

// 1.基本类型传递参数时，拷贝的是值

// int a = 10;

// test(a);

// System.out.println(a); // 10

// 2.引用类型传递参数是，拷贝的是地址

// Cat c = new Cat(); // 地址6fd46259

// c.name = "白猫";

// test2(c);

// System.out.println(c.name); // 黑猫

// 3.

Cat c = new Cat(); // 地址6fd46259

c.name = "白猫";

test3(c);

System.out.println(c.name); // 白猫

}

static void test(int a) { // 10

a = 20; // 20

}

static void test2(Cat c2) { // c2 = 6fd46259

c2.name = "黑猫";

}

static void test3(Cat c3) { // c3 = 6fd46259

c3 = new Cat(); // c3 = 7489327

c3.name = "黑猫";

}

}

## super

用在子类变量或子类方法与父类命名有冲突的时候

public class Person {

String name = "张三";

void eat() {

System.out.println("父类eat方法");

}

}

public class Student extends Person {

String name = "李四";

void eat() {

System.out.println("子类eat方法");

}

void test() {

// 当子类的变量名与父类的变量名有冲突时，可以用super.变量名 引用父类的变量

// System.out.println(name);

System.out.println(super.name);

// 调用方法

// eat(); // 就近调用子类eat()

super.eat(); //调用父类eat方法

}

public static void main(String[] args) {

new Student().test();

}

}

## super 用在构造时

public class Person {

// public Person() {

// System.out.println("在调用父类无参构造");

// }

public Person(int a) {

}

}

// 1. 子类的所有构造都会由编译器，偷偷加一条super(),调用父类默认构造

// 2. 如果父类没有提供无参构造，子类的构造中都会报错

// 解决方法 a：在父类提供一个无参构造

// 解决方法b: 自己手工调用父类的有参构造

public class Student extends Person {

public Student() {

// super() 调用父类默认构造

super(1);

}

public Student(int a ) {

// super() 调用父类默认构造

super(1);

}

public static void main(String[] args) {

// new Student();

new Student(1);

}

}