

# 1 static 关键字

1: 如果没有 static 会怎样?

1: 定义 Person 类

1: 姓名、年龄、国籍, 说话行为

2: 多个构造, 重载形式体现

2: 中国人的国籍都是确定的

1: 国籍可以进行显示初始化

```
class Person {
    String name;
    int age;
    String gender;
    String country = "CN";

    Person() {

    }

    Person(String name, int age, String gender, String country) {
        this.name = name;
        this.age = age;
        this.gender = gender;
        this.country = country;
    }

    void speak() {
        System.out.println("国籍:" + country + " 姓名:" + name + " 性别:"
            + gender + " 年龄:" + age + " 哈哈!!!");
    }
}
```

3: new Person 对象

1: 分析内存

2: 每个对象都维护实例变量国籍也是。

```
public class PersonDemo {
    public static void main(String[] args) {
        Person p1 = new Person("jack", 20, "男");
        p1.speak();

        Person p2 = new Person("rose", 18, "女");
        p2.speak();
    }
}
```

```
}  
}
```

#### 4: 内存分析

1: 栈, 堆、共享区

2: Demo.class 加载至共享区

1: Demo 类的 main 方法进栈

2: Person p1=new Person();

1: Person.class 加载进方法区

2: 堆内存开辟空间, 实例变量进行默认初始化, 显示初始化。

3: 内存地址传给变量 p1, 栈和堆建立连接

3: person p2=new Person();

1: 堆内存开辟空间, 实例变量进行默认初始化, 显示初始化。

2: 内存地址传给变量 p2, 栈和堆建立连接

4: 如果建立多个 Person 对象发现问题

1: 每个对象都维护有国籍。

5: 解决问题, 内存优化

1: 为了让所有 Person 对象都共享一个 country, 可以尝试将 country 放入共享区。

2: country 变量如何放入共享区? 对象如何访问?

1: 使用 static

2: static

1: 为了实现对象之间重复属性的数据共享

3: static 使用

1: 主要用于修饰类的成员

1: 成员变量

1: 非静态成员变量: 需要创建对象来访问

2: 静态成员变量: 使用类名直接调用, 也可以通过对象访问

```
public static void main(String[] args) {
```

```
    //访问静态成员
```

```
    //直接通过类名来调用
```

```
    String country=Person.country;
```

```
    System.out.println(country);
```

```
    //通过对象.成员的形式访问
```

```
    Person p1 = new Person("jack", 20, "男");
```

```
    p1.country="US";
```

```
    p1.speak();
```

```
}
```

```
class Person {
```

```
    String name;
```

```

    int age;
    String gender;
    //static 修饰成员变量
    static String country = "CN";

    Person() {

    }

    Person(String name, int age, String gender) {
        this.name = name;
        this.age = age;
        this.gender = gender;
    }

    void speak() {

        System.out.println("国籍:" + country + " 姓名: " + name + " 性别: " + gender
            + " 年龄: " + age + " 哈哈!!!");
    }
}

```

## 2: 成员方法

可以使用类名直接调用

### 1: 静态函数:

- 1: 静态函数中不能访问非静态成员变量，只能访问静态变量。
- 2: 静态方法不可以定义 this, super 关键字。
- 3: 因为静态优先于对象存在，静态方法中更不可以出现 this

2: 非静态函数: 非静态函数中可以访问静态成员变量

```

class Person {
    String name;
    int age;
    String gender;
    //static 修饰成员变量
    static String country = "CN";

    Person() {

    }

    Person(String name, int age, String gender) {

```

```

        this.name = name;
        this.age = age;
        this.gender = gender;
    }
    //非静态方法
    void speak() {
        //非静态方法可以访问静态成员
        System.out.println("国籍:" + country );

        System.out.println("国籍:" + country + " 姓名: " + name + " 性别: " + gender
            + " 年龄: " + age + " 哈哈!!!");
    }
    //静态方法
    static void run() {
        //静态方法只能访问静态成员变量。
        System.out.println("国籍:"+country);

        //静态方法访问非静态成员变量, 编译报错。
        System.out.println(" 姓名: " + name);

        //静态方法中不可以出现this, 编译报错
        this.speak();
    }
}

```

## 2: 细节:

- 1: 静态函数中不能使用非静态变量
- 2: 非静态函数可以访问静态变量

## 3: 为什么静态函数中不能访问非静态成员

- 1: static 修饰的成员在共享区中。优先于对象存在
- 2: 验证

### 1: 使用静态代码块验证

#### 1: 静态代码块

```

static{
    静态代码块执行语句;
}

```

#### 1: 静态代码块特点

随着类的加载而加载。只执行一次, 优先于主函数。用于给类进行初始化。

```

public class PersonDemo {

```

```

public static void main(String[] args) {

    // 访问静态成员
    // 直接通过类名来调用
    String country = Person.country;
    System.out.println(country);

    // 通过对象.成员的形式访问
    Person p1 = new Person("jack", 20, "男");
    p1.country = "US";
    p1.speak();

}

}

class Person {
    String name;
    int age;
    String gender;
    // static 修饰成员变量
    static String country = "CN";
    static {
        System.out.println("这是静态代码块");
    }

    {
        System.out.println("这是构造代码块");
    }

    Person() {
        System.out.println("无参数构造");
    }

    Person(String name, int age, String gender) {
        this.name = name;
        this.age = age;
        this.gender = gender;
        System.out.println(" 有参数构造");
    }

    // 非静态方法
    void speak() {

```

```

// 非静态方法可以访问静态成员
System.out.println("国籍:" + country);

System.out.println("国籍:" + country + " 姓名:" + name + " 性别:"
+ gender
+ " 年龄:" + age + " 哈哈!!!");
// 非静态方法可以调用静态方法。
run();
}

// 静态方法
static void run() {
    // 静态方法只能访问静态成员变量。
    System.out.println("国籍:" + country);
}
}

```

#### 4: static 特点

- 1 随着类的加载而加载，静态会随着类的加载而加载，随着类的消失而消失。说明它的生命周期很长。
- 2 优先于对象存在。-->静态是先存在，对象是后存在。
- 3 被所有实例(对象)所共享。
- 4 可以直接被类名调用

#### 5: 静态变量（类变量）和实例变量的区别：

- 1 存放位置
  - 1: 类变量随着类的加载而加载存在于方法区中。
  - 2: 实例变量随着对象的建立而存在于堆内存中。
- 2 生命周期
  - 1: 类变量生命周期最长,随着类的消失而消失。
  - 2: 实例变量生命周期随着对象的消失而消失。

#### 6: 静态优缺点

- 1: 优点:对对象的共享数据进行单独空间的存储,节省空间 例如 Person 都有国籍。该数据可以共享可以被类名调用
- 2: 缺点: 生命周期过长  
访问出现局限性。(静态只能访问静态)

#### 7: 什么时候定义静态变量

- 1:静态变量（类变量）当对象中出现共享数据  
例如: 学生的学校名称。学校名称可以共享  
对象的数据要定义为非静态的存放在堆内存中(学生的姓名,学生的年龄)

#### 8: 什么时候定义静态函数

如果功能内部没有访问到非静态数据(对象的特有数据。那么该功能就可以定义为

静态)

## 9: 静态的应用

自定义数组工具类

```
/*
    定义数组工具类
    1: 定义一个遍历数组的函数
    2: 定义一个求数组和的功能函数    1. 遍历    2. 两两相加
    3: 定义一个获取数组最大值的功能函数
    4: 定义一个获取数组最大值角标的功能函数
    5: 定义一个返回指定数在指定数组中包含的角标的功能函数
    6: 定义一个可以用于排序int数组的函数
        1: 冒泡
        2: 选择

    定义自己的工具类
*/
class Arrays {

    private Arrays() {

    }

    // 1: 定义一个遍历数组的函数
    public static void print(int[] arr) {
        for (int x = 0; x < arr.length; x++) {
            if (x != (arr.length - 1)) {
                System.out.print(arr[x] + ",");
            } else {
                System.out.print(arr[x]);
            }
        }
    }

    // 2: 定义一个求数组和的功能函数
    public static int getSum(int[] arr) {
        int sum = 0;
        for (int x = 0; x < arr.length; x++) {
            sum += arr[x];
        }
        return sum;
    }
}
```

```

// 3:定义一个获取数组最大值的功能函数
public static int getMax(int[] arr) {
    int max = 0;
    for (int x = 0; x < arr.length; x++) {
        if (arr[max] < arr[x]) {
            max = x;
        }
    }
    return arr[max];
}

// 4:定义一个获取数组最大值角标的功能函数
public static int getIndexMax(int[] arr) {
    int max = 0;
    for (int x = 0; x < arr.length; x++) {
        if (arr[max] < arr[x]) {
            max = x;
        }
    }
    return max;
}

// 5:定义一个返回 指定数在指定数组中包含的角标的功能函数
public static int getIndex(int[] arr, int src) {
    int index = -1;
    for (int x = 0; x < arr.length; x++) {
        if (arr[x] == src) {
            index = x;
        }
    }
    return index;
}

// 冒泡
public static void test(int[] arr) {
    for (int x = 0; x < arr.length - 1; x++) {
        if (arr[x] > arr[x + 1]) {
            int temp = arr[x + 1];
            arr[x + 1] = arr[x];
            arr[x] = temp;
        }
    }
}

```



// 选择排序

```
public static void selectSort(int[] arr) {
    for (int x = 0; x < arr.length - 1; x++) {
        for (int y = 1 + x; y < arr.length; y++) {
            if (arr[x] > arr[y]) {
                int temp = arr[y];
                arr[y] = arr[x];
                arr[x] = temp;
            }
        }
    }
}
```

// 7: 定义一个可以将整数数组进行反序的功能函数。

```
public static void reverseSort(int[] arr) {
    int start = 0;
    int end = arr.length - 1;
    for (int x = 0; x < arr.length; x++) {
        if (start < end) {
            int tem = arr[start];
            arr[start] = arr[end];
            arr[end] = tem;
        }
        start++;
        end--;
    }
}
```

// 折半查找

```
public static int halfSearch(int key, int[] arr) {
    int min = 0;
    int max = arr.length - 1;
    int mid = 0;

    while (min < max) {
        mid = (min + max) / 2;
        if (key > arr[mid]) {
            min = mid + 1;
        } else if (key < arr[mid]) {
            max = mid - 1;
        } else {
            return mid;
        }
    }
}
```

```

        }
    }
    return -1;
}

}

class Demo6 {

    public static void main(String[] args) {
        int[] arr = { 3, 4, 5, 2, 3, 7, 4 };
        Arrays.print(arr);
        System.out.println();
        Arrays.selectSort(arr);
        Arrays.print(arr);
    }
}

```

练习：统计创建对象的人数

```

class Person
{
    public String name;
    public int age;
    static public long all_count;
    public Person() {
        all_count++;
    }
    public Person( String name , int age ){
        all_count++;
        this.name = name;
        this.age = age;
    }
    // 统计人数的函数
    public long getCount() {
        return all_count;
    }
    // 应该具备找同龄人的功能
    public boolean isSameAge( Person p1 ){
        return this.age == p1.age;
    }
}

class Demo9
{
    public static void main(String[] args)
    {
        Person p1 = new Person( "jame" , 34 );
        Person p2 = new Person( "lucy" , 34 );

        Person p3 = new Person( "lili" , 34 );
        Person p4 = new Person();
        System.out.println( p1.getCount() + " " + p2.getCount() + " " +
p3.getCount() );
        System.out.println( p1.isSameAge( p2 ) );
        System.out.println( p1.isSameAge( p3 ) );
    }
}

```

## 1.1 main 方法详解

主函数是静态的

```
public static void main(String[] args){  
  
}
```

主函数是什么：主函数是一个特殊的函数，作为程序的入口，可以被 jvm 识别。

主函数的定义：

public : 代表该函数的访问权限是最大的。

static : 代表主函数随着类的加载，就已经存在了。

void: 主函数没有具体的返回值

main : 不是关键字，是一个特殊的单词可以被 jvm 识别。

(String[] args) 函数的参数，参数类型是一个数组，该数组中的元素是字符串。字符串类型的数组。

主函数的格式是固定的：jvm 能够识别

jvm 在调用函数是，传入的是 new String[0];

可以在 dos 窗口中执行 java Demo5 hello world 给类 Demo5 的 main 方法传递 2 个参数，参数与参数之间通过空格隔开。

```
class Demo5 {  
  
    public static void main(String[] args) {  
  
        // 获取String[] args 数组长度  
        System.out.println(args.length);  
  
        // 变量args数组  
        for (int x = 0; x < args.length; x++) {  
            System.out.println(args[x]);  
        }  
    }  
}
```

```
class MainTest {  
  
    public static void main(String[] args) {  
        // 字符串数组  
        String[] arr = { "good", "study", "java" };  
  
        // 调用Demo5类的main方法，传递参数。  
        Demo5.main(arr);  
    }  
}
```

```
}
```

## 2 单例设计模式

一些人总结出来用来解决特定问题的固定的解决方案。

解决一个类在内存中只存在一个对象，想要保证对象的唯一。

- 1 为了避免其他程序过多的建立该类对象。禁止其他程序建立该类对象。
- 2 为了其他程序可以访问该类对象，在本类中自定义一个对象。
- 3 方便其他程序对自定义类的对象的访问，对外提供一些访问方式。

代码：

- 1 将构造函数私有化
- 2 在类中创建一个私有的本类对象
- 3 提供一个用类名调用的公有方法获取该对象。

```
class Single {  
  
    private static Single s = new Single(); // 恶汉式  
  
    private Single() {  
  
    }  
  
    public static Single getInstance() {  
        return s;  
    }  
}  
  
class Single2 {  
    private static Single2 s = null; // 懒汉  
  
    private Single2() {  
  
    }  
  
    public static Single2 getInstance() {  
        if (s == null) {  
            s = new Single2();  
        }  
        return s;  
    }  
}
```

## 3 继承

### 3.1 类和类之间的常见关系。

1: 既然继承是描述类和类之间的关系，就需要先来了解类和类之间的常见关系

#### 3.1.1 现实生活的整体与部分

举例说明

1: 现实生活

- 1: 学生 是人
- 2: 狗 是动物
- 3: 球队 包含 球员 整体与部分的关系，部分可以删除和增加
- 4: 笔记本包含 cpu 整体与部分的关系，部分不可以删除和增加
- 5: 航母编队 包含（航母 护卫舰 驱逐舰 舰载机 核潜艇）

#### 3.1.2 java 中的类与类关系

java 中的类关系

1: is a 关系 (学生是人)

2: has a 整体与部分

```
class Person{
    String name;
    int age;
    Address add;

    Person(){

    }

    Person(String name,int age,Address add){

        this.name=name;
        this.age=age;
        this.add=add;

    }

    void speak(){
        System.out.println("姓名: "+name+" 年龄: "+age+" "+add.print());
    }
}
```

```

class Address{
    String country;
    String city;
    String street;

    Address(){

    }
    Address(String country,String city,String street){
        this.country=country;
        this.city=city;
        this.street=street;
    }

    String print(){
        return "地址: "+country+" "+"城市: "+city+" 街道: "+street;
    }
}

class Demo3{

    public static void main(String[] args){

        Address add=new Address("中国","广州","棠东东路");
        Person p=new Person("jack",27,add);
        p.speak();

        System.out.println();
    }
}

```

## 3.2 继承

- 1: 描述一个学生类
  - 1: 姓名年龄学号属性, 学习的方法
- 2: 描述一个工人类
  - 1: 姓名年龄工号属性, 工作的方法
- 3: 描述一个人类
  - 1: 姓名年龄属性, 说话的方法。
- 4: 发现学生类和人类天生有着联系, 学生和工人都是人。所以人有的属性和行为学生和工人都会有。出现类代码重复

```

class Person {

```

```

String name;
int age;

// 静态变量（类变量）对象和对象之间的代码重复使用静态变量
static String country = "CN";

Person() {

}

void speak() {
    System.out.println(name + ":哈哈，我是人!!!");
}

}

// 让学生类和人类产生关系，发现学生is a 人，就可以使用继承
class Student {

    String name;
    int age;

    Student() {

    }

    void study() {
        System.out.println("姓名: " + name + "年纪: " + age + ":好好学习");
    }

}

class Worker {
    String name;
    int age;

    void work() {
        System.out.println(name + ":好好工作，好好挣钱。");
    }

}

class Demo1 {

```



```

    public static void main(String[] args) {
        Student s = new Student();
        s.name = "jack";
        s.age = 20;
        s.study();

        Worker w = new Worker();
        w.name = "rose";

        w.work();
    }
}

```

5: 问题:

- 1: 如果没有继承, 出现类和类的关系无法描述
- 2: 如果没有继承, 类和类之间有关系会出现类和类的描述代码的重复。

### 3.3 继承特点

- 1: 描述类和类之间的关系
- 2: 降低类和类之间的重复代码
- 1: 降低对象和对象之间的代码重复使用静态变量
- 2: 降低类和类之间的代码重复使用就继承

### 3.4 extends 关键字

继承使用 extends 关键字实现

- 1: 发现学生是人, 工人是人。显然属于 is a 的关系, is a 就是继承。
- 2: 谁继承谁?

学生继承人, 发现学生里的成员变量, 姓名和年龄, 人里边也都进行了定义。有重复代码将学生类的重复代码注释掉, 创建学生类对象, 仍然可以获取到注释的成员。这就是因为继承的关系, 学生类(子类)继承了人类(父类)的部分

```

class Person {
    String name;
    int age;

    // 静态变量(类变量) 对象和对象之间的代码重复使用静态变量
    static String country = "CN";

    Person() {

    }
}

```

```

        void speak() {
            System.out.println(name + ":哈哈, 我是人!!!");
        }

    }

    // 让学生类和人类产生关系, 发现学生is a 人, 就可以使用继承
    class Student extends Person {

        Student() {

        }

        void study() {
            System.out.println("姓名: " + name + "年纪: " + age + ":好好学习");
        }

    }

    class Worker extends Person {

        void work() {
            System.out.println(name + ":好好工作, 好好挣钱。");
        }

    }

    class Demo1 {

        public static void main(String[] args) {
            Student stu = new Student();
            stu.name = "jack";
            stu.age = 20;
            stu.study();
            stu.speak();
            System.out.println(stu.country);
            System.out.println(Student.country);

            Worker worker = new Worker();
            worker.name = "rose";
            System.out.println(worker.country);
            worker.work();
            worker.speak();
        }
    }

```

```

        System.out.println();
    }
}

```

继承细节;

- 1: 类名的设定, 被继承的类称之为父类(基类), 继承的类称之为子类
- 2: 子类并不能继承父类中所有的成员
  - 1: 父类定义完整的成员 静态成员, 非静态, 构造方法。静态变量和静态方法都可以通过子类名.父类静态成员的形式调用成功。
  - 2: 所有的私有成员不能继承,private 修饰的成员。
  - 3: 构造函数不能被继承
- 3: 如何使用继承
  - 1: 不要为了使用继承而继承。工人和学生都有共性的成员, 不要为了节省代码, 让工人继承学生。

```

/*
如何使用继承: 验证是否有 is a 的关系
例如: 学生是人, 小狗是动物
注意: 不要为了使用某些功能而继承, java只支持单继承
*/
class DK {

    void Ip4S() {
        System.out.println("好玩");
    }
}

class BGir extends DK {

}

class Demo {

    public static void main(String[] args) {

        new BGir().Ip4S();

    }
}

```

### 3.5 super 关键字

- 1: 定义 Father(父类)类

```

    1: 成员变量 int x=1;
    2: 构造方法无参的和有参的，有输出语句
2: 定义 Son 类 extends Father 类
    1: 成员变量 int y=1;
    2: 构造方法无参和有参的。有输出语句
        1: this.y=y+x;
3: 创建 Son 类对象
    Son son=new Son(3);
    System.out.println(son.y); //4
class Father {
    int x = 1;

    Father() {
        System.out.println("这是父类无参构造");
    }

    Father(int x) {

        this.x = x;
        System.out.println("这是父类有参构造");
    }

    void speak() {
        System.out.println("我是父亲");
    }
}

class Son extends Father {
    int y = 1;

    Son() {
        System.out.println("这是子类的无参构造");
    }

    Son(int y) {

        this.y = y + x;
        System.out.println("这是子类的有参构造");
    }

    void run() {
        super.speak(); // 访问父类的函数
        System.out.println("我是儿子");
    }
}

```

```

}

class Demo6 {

    public static void main(String[] args) {
        Son s = new Son(3);
        System.out.println(s.y); // 4
    }
}

```

4: 子类对象为什么可以访问父类的成员。

1: `this.y=y+x`; 有一个隐式的 `super` `super.x`

5: `super` 关键字作用

- 1: 主要存在于子类方法中, 用于指向子类对象中父类对象。
- 2: 访问父类的属性
- 3: 访问父类的函数
- 4: 访问父类的构造函数

6: `super` 注意

`this` 和 `super` 很像, `this` 指向的是当前对象的调用, `super` 指向的是当前调用对象的父类。Demo 类被加载, 执行 `main` 方法, `Son.class` 加载, 发现有父类 `Father` 类, 于是 `Father` 类也加载进内存。类加载完毕, 创建对象, 父类的构造方法会被调用 (默认自动无参), 然后执行子类相应构造创建了一个子类对象, 该子类对象还包含了一个父类对象。该父类对象在子类对象内部。`this` `super` 只能在有对象的前提下使用, 不能在静态上下文使用。

2: 子类的构造函数默认第一行会默认调用父类无参的构造函数, 隐式语句

```
super();
```

1: 父类无参构造函数不存在, 编译报错。

```

Son(int y) {
    //super(); 隐式语句
    this.y = y + x;
    System.out.println("这是子类的有参构造");
}

```

3: 子类显式调用父类构造函数

在子类构造函数第一行通过 `super` 关键字调用父类任何构造函数。如果显式调用父类构造函数, 编译器自动添加的调用父类无参数的构造就消失。构造函数间的调用只能放在第一行, 只能调用一次。`super()` 和 `this()` 不能同时存在构造函数第一行。

```

Son(int y) {
    super(y); // 子类显式调用父类构造函数
    this.y = y + x;
    System.out.println("这是子类的有参构造");
}

```

```

Son(int y) {
    this(); //不能同时存在构造函数第一行
    super(y);
    this.y = y + x;
    System.out.println("这是子类的有参构造");
}

```

#### 4: super 思考

如果开发者自定义了一个类，没有显示的进行类的继承，那么该类中成员函数是否可以使用 super 关键字？可以使用，继承了 Object 类，Object 类是所有类的父类。

```

class Demo7 {
    public void print(){
        System.out.println(super.toString());
    }
    public static void main(String[] args){
        new Demo7().print();
        System.out.println();
    }
}

```

#### 5: 继承练习

#### 7: 重写 (Override)

##### 1: 定义 Father 类

1: 姓名，吃饭方法，吃窝窝头。

##### 2: 定义 Son 类，继承 Father

1: Son 类中不定义任何成员，子类创建对象，仍然可以调用吃饭的方法。

2: 父类的吃饭的方法，Son 不愿吃。Son 自己定义了吃饭的方法。

1: 此时父类中有一个吃饭的方法，子类中有 2 个吃饭的方法，一模一样，只是方法体不一样。

2: 一个类中两个函数一模一样，是不允许的。

1: 编译运行，执行了子类的方法。

2: 使用父类的方法，在子类方法中，使用 super.父类方法名。

```

class Father {
    String name;

    void eat() {
        System.out.println("吃窝窝");
    }
}

class Son extends Father {

```

```

    public void eat() { // 继承可以使得子类增强父类的方法
        System.out.println("来俩小菜");
        System.out.println("来两杯");
        System.out.println("吃香喝辣");
        System.out.println("来一根");
    }
}

class Demo8 {

    public static void main(String[] args) {
        Son s = new Son();
        //执行子类的方法
        s.eat();

    }
}

```

### 3: 该现象就叫做重写 (覆盖 override)

- 1: 在继承中，子类可以定义和父类相同的名称且参数列表一致的函数，将这种函数称之为函数的重写。

### 4: 前提

- 1: 必须要有继承关系

### 5: 特点

- 1: 当子类重写了父类的函数，那么子类的对象如果调用该函数，一定调用的是重写后的函数。

可以通过 super 关键字进行父类的重写函数的调用。

- 2: 继承可以使得子类增强父类的方法

### 6: 细节

- 1: 函数名必须相同
- 2: 参数列表必须相同
- 3: 子类重写父类的函数的时候，函数的访问权限必须大于等于父类的函数的访问权限否则编译报错
- 4: 子类重写父类的函数的时候，返回值类型必须是父类函数的返回值类型或该返回值类型的子类。不能返回比父类更大的数据类型：如子类函数返回值类型是 Object

- 1: 定义 A B C 类 B extends A

- 2: Father 类中定义 A getA();

- 3: Son 类中重写 getA(); 方法，尝试将返回值修改为 B, C ,Object

- 1: B 编译通过

- 2: C 编译失败 ,没有继承关系

- 3: Object 编译失败,比父类的返回值类型更大

```

class A {

```

```

}

class B extends A {

}

class C {

}

class Father {
    String name;

    void eat() {
        System.out.println("吃窝窝");
    }

    // 定义一个函数，获取A类的对象，
    A getA() {
        return new A();
    }
}

class Son extends Father {

    public void eat() { // 继承可以使得子类增强父类的方法
        System.out.println("来两杯");
        System.out.println("来俩小菜");
        super.eat();
        System.out.println("来一根");
    }

    // B类是A类的子类
    B getA() {
        return new B();
    }
}

class Demo8 {

    public static void main(String[] args) {
        Son s = new Son();
        s.eat();
    }
}

```



```
}  
}
```

#### 7: 子类对象查找属性或方法时的顺序:

##### 1: 原则: 就近原则。

如果子类的对象调用方法, 默认先使用 `this` 进行查找, 如果当前对象没有找到属性或方法, 找当前对象中维护的 `super` 关键字指向的对象, 如果还没有找到编译报错, 找到直接调用。

#### 8: 重载和重写的不同

##### 1: 重载(overload):

1: 前提: 所有的重载函数必须在同一个类中

2: 特点:

函数名相同, 参数列表不同, 与其他的无关(访问控制符、返回值类型)

3: 不同:

个数不同、顺序不同、类型不同

##### 2: 重写(override):

1: 前提: 继承

2: 特点:

函数名必须相同、参数列表必须相同。

子类的返回值类型要等于或者小于父类的返回值

#### 9: 重写练习

描述不同的动物不同的叫法

1: 定义动物类

有名字, 有吃和叫的方法

2: 定义狗继承动物重写父类吃和叫的方法

3: 定义猫继承动物重写父类吃和叫的方法

```
class Animal{  
    int x=1;  
    String name;  
  
    void eat(){  
        System.out.println("吃东西");  
    }  
    void shout(){  
        System.out.println("我是动物");  
    }  
}  
  
class Dog extends Animal{  
  
    void eat(){  
        System.out.println("啃骨头");  
    }  
}
```

```

    }
    void shout() {
        System.out.println("旺旺");
    }
    void eat(String food) {
        System.out.println("吃: "+food);
    }
}
class Cat extends Animal{

    void eat() {
        System.out.println("吃老鼠");
    }
    void shout() {
        System.out.println("喵喵");
    }
}

class Demo9{

    public static void main(String[] args){
        Dog d=new Dog();
        d.shout();
        d.eat();

        Cat c=new Cat();
        c.shout();
        c.eat();
        System.out.println();
    }
}

```

## 3.6 instanceof 关键字

1: 快速演示 instanceof

```

Person p=new Person();
System.out.println( p instanceof Person);

```

2: instanceof 是什么?

- 1: 属于比较运算符:
- 2: instanceof 关键字: 该关键字用来判断一个对象是否是指定类的对象。
- 3: 用法:

对象 instanceof 类;

该表达式是一个比较运算符，返回的结果是 boolean 类型 true|false

注意：使用 instanceof 关键字做判断时，两个类之间必须有关系。

### 3: 案例

定义一个功能函数，根据传递进来的对象的做不同的事情，如果是狗让其看家，如果是猫让其抓老鼠

- 1: 定义动物类
- 2: 定义狗类继承动物类
- 3: 定义猫类继承动物类
- 4: 定义功能根据传入的动物，执行具体的功能
- 5: instanceof 好处
  - 1: 可以判断对象是否是某一个类的实例

```
package oop01;

/*
instanceof
比较运算符
检查是否是类的对象
    1: 可以判断对象是否是某一个类的实例
    用法
    对象 instanceof 类;

案例
定义一个功能函数，根据传递进来的对象的做不同的事情
    如果是狗让其看家，如果是猫让其抓老鼠
1: 定义动物类
2: 定义狗类继承动物类
3: 定义猫类继承动物类
4: 定义功能根据传入的动物，执行具体的功能
*/

class Animal {

    String name;

    void eat() {
        System.out.println("吃东西");
    }

    void shout() {
        System.out.println("我是动物");
    }
}

class Dog extends Animal {
```

```

    void eat() {
        System.out.println("啃骨头");
    }

    void shout() {
        System.out.println("旺旺");
    }
}

class Cat extends Animal {

    void eat() {
        System.out.println("吃老鼠");
    }

    void shout() {
        System.out.println("喵喵");
    }
}

class Demo11 {

    public static void main(String[] args) {

        Demo11 d = new Demo11();

        // 对象 instanceof 类;
        System.out.println(d instanceof Demo11);

        d.doSomething(new Dog());
        d.doSomething(new Cat());
    }

    // 定义一个功能函数，根据传递进来的对象的做不同的事情
    // 如果是狗让其看家，如果是猫让其抓老鼠
    // 对象 instanceof 类;
    void doSomething(Animal a) {
        if (a instanceof Dog) {
            a.eat();
            a.shout();
            System.out.println("小狗看家");
        } else if (a instanceof Cat) {

```

```

        a.eat();
        a.shout();
        System.out.println("抓老鼠");
    }
}

```

练习:

```

byte[] bs = new byte[] { 1, 2, 3 };
int[] is = new int[] { 1, 2, 3 };
String[] ss = new String[] { "jack", "lucy", "lili" };
System.out.println(bs instanceof byte[]); // true
System.out.println(is instanceof int[]); // true
System.out.println(ss instanceof String[]); // true
// System.out.println(bs instanceof int[]); // 不可转换的类型

```

## 3.7 final 关键字

- 1: 定义静态方法求圆的面积
- 2: 定义静态方法求圆的周长
- 3: 发现方法中有重复的代码，就是 PI，圆周率。
  - 1: 如果需提高计算精度，就需要修改每个方法中圆周率。
- 4: 描述一个变量
  - 1: 方法都是静态的，静态只能访问静态，所以变量也定义为静态的。

```

public static double PI=3.14;

1: 如果定义为 public 后，新的问题，类名.PI=300; 改变了 PI 的值。
2: 修改为 private, 修改为 private 后进行了封装，需要 getset 公共访问方法。
3: 现有的知识不能解决这样的问题了。可以使用 final

class Demo12 {

    public static final double PI = 3.14; // 静态常量

    public static double getArea(double r) {
        return PI * r * r;
    }

    public static double getLength(double r) {
        return PI * r * 2;
    }
}

```

```

public static void main(String[] args) {

    // Demo12.PI=300; 无法为最终变量 PI 指定值
    System.out.println(Demo12.PI);

}
}

```

## 5: 使用 final

1: final 关键字主要用于修饰类、类成员、方法、以及方法的形参。

2: final 修饰成员属性:

1: 说明该成员属性是常量, 不能被修改。

```
public static final double PI=3.14;
```

1: public : 访问权限最大

2: static : 内存中只有一份

3: final : 是一个常量

4: 常量名大写

5: 必须初赋值。

2: 使用类名.成员。修改该成员的值, 报错。--常量不能被修改

1: 基本数据类型, final 使值不变

2: 对象引用, final 使其引用恒定不变, 无法让其指向一个新的对象, 但是对象自身却可以被修改。

3: 该关键字一般和 static 关键字结合使用

1: 常量可以优先加载, 不必等到创建对象的时候再初始化。

4: final 和 static 可以互换位置

5: 常量一般被修饰为 final

3: final 修饰类:

1: 该类是最终类, 不能被继承。

1: 将父类加 final 修饰, 子类继承, 就会报错。

2: 查看 api 文档发现 String 类是 final 的。Integer 类也是 final 的

1: 为了防止代码功能被重写

2: 该类没有必要进行扩展

4: final 修饰方法:

1: 该方法是最最终方法, 不能被重写

2: 当一个类被继承, 那么所有的非私有函数都将被继承, 如果函数不想被子类继承并重写可以将该函数 final 修饰

3: 当一个类中的函数都被修饰为 final 时, 可以将类定义为 final 的。

```

class Father2{
    final void eat(){
        System.out.println("eating...");
    }
}

```

```

    }
}

class Son2 extends Father2{
    //该方法是最终方法，不能被重写
    void eat(){
        System.out.println("eating....");
    }
}

class Demo12 {

    public static void main(String[] args) {

        // Demo12.PI=300; 无法为最终变量 PI 指定值
        System.out.println(Demo12.PI);
        Son2 s=new Son2();
        s.eat();

    }
}

```

#### 5: final 关键字修饰形参

- 1: 当形参被修饰为 final, 那么该形参所属的方法中不能被篡改。
- 2: 项目中主要用于一些只用来遍历未知数据的函数。将未知变量声明为 final 的。增强数据的安全性。

```

class Demo14 {

    public static void main(String[] args) {

        System.out.println();
        String[] arr = { "think in java", "java就业教程", "java核心技术" };

        print(arr);

    }

    // 该方法，打印书名。
    public static void print(final String[] arr) {
        //arr = null; ,无法重新赋值

        for (int x = 0; x < arr.length; x++) {
            System.out.println(arr[x]);
        }
    }
}

```

```
}
```

#### 10: 思考

为什么子类一定要访问父类的构造函数呢

1: 子类继承了父类的属性, 如果要使用父类的属性必须初始化, 创建子类对象, 必须先初始化父类属性

必须调用父类的构造方法。

2: 为什么调用父类无参的构造函数

设计 java 语言之时, 只知道编译器会默认添加无参的构造函数, 有参的无法确定。

但是可以通过 `super` 关键字显式调用父类指定构造函数。

3: 为什么 `super()` `this()` 语句要放在构造函数的第一行

子类可能会用到父类的属性, 所以必须先初始化父类。

## 4 作业

1. 静态和非静态的区别。说一下内存。
2. 成员变量和静态变量的区别?
3. 静态的特点以及注意事项?
4. 什么时候使用静态?
5. 继承的好处?
6. java 改良多继承的原因?
7. 当使用一个已存在的继承体系时, 该如何更快应用
8. 什么时候用继承?
9. `super` 和 `this` 的特点?
10. 覆盖的特点, 何时应用, 注意事项?
11. 子类的实例化过程? 为什么是这样的实例化过程?
12. `super` 语句, 和 `this` 语句为什么不能同时存在, `super` 为什么要定义在第一行?