

## 目录

一、面向对象思想 .....	2
(一) 面向对象思想引入 .....	2
(二) 面向对象思想的概述 .....	2
(三) 面向对象思想的特点 .....	2
二、面向对象开发、设计、特征 .....	3
三、类与对象 .....	3
(一) 成员变量与局部变量的区别（同类比：静态变量与成员变量的区别） .....	5
(二) 形式参数的问题 .....	6
(三) 匿名对象的概述和应用 .....	6
四、封装 .....	6
(一) PRIVATE 关键字 .....	6
(二) THIS 关键字 .....	8
(三) 构造方法及其重载（同比与：方法重写） .....	9
(四) STATIC 关键字 .....	10
(五) 静态变量与成员变量的区别（同类比：成员变量与局部变量的区别） .....	12
(六) MAIN 格式及详细解释 .....	13
(七) 代码块 .....	14
五、制作和使用帮助文档 .....	15
(一) 制作说明书 .....	15
(二) 使用说明书 .....	16
(三) 使用 JDK 提供的 API 帮助文档 .....	16
六、继承 .....	17
(一) 引出继承的概念 .....	17
(二) 继承的利与弊 .....	18
(三) 继承的特点及注意事项 .....	18
(四) 继承中的不同关系 .....	18
一) 继承中成员变量的关系 .....	18
二) 继承中构造方法的关系 .....	19
三) 继承中成员方法的关系 .....	20
(五) THIS 与 SUPER 的区别 .....	21
(六) 方法重写（同比与：方法重载） .....	21
(七) FINAL 关键字 .....	22
附件 .....	24
案例 1.猜数字小游戏 .....	24
案例 2.有关继承的代码练习（读代码写结果） .....	25

## 一、面向对象思想

### (一) 面向对象思想引入

通过举例数组来讲述面向对象思想的引入。当有多个数组都需要遍历时，可以将遍历的代码封装到方法中，需要遍历时就调用相应的方法即可，提高代码的复用性。在对数组遍历的基础上继续增加需求，比如获取最值、数值逆序等，同样需要将这些功能封装到相应的方法中，这样继续封装会发现方法越来越多，于是就想能不能将这些方法继续进行封装呢？

通过其那面的讲解知道类是可以存放方法的，所以就考虑使用类来封装这些方法，将来再做数组的操作时，不用去找具体的方法，先找到这个类，然后使用这个类中的方法，这就是**面向对象思想的编程方式**。

完成一个需求的步骤：①搞清楚要做什么；②再分析怎么做；③再一步步代码实现；具体的每一步都需要程序员去实现和操作，这些步骤相互调用和协作来完成需求。每一个具体的步骤中程序员都是参与者，并且需要面对具体的每一个步骤和过程，这就是**面向过程最直接的体现**。

那么什么是面向过程开发呢？面向过程开发就是面向着具体的每一个步骤和过程，把每一个步骤和过程完成，然后有这些功能相互调用来完成需求。

面向过程的代表语言，即 C 语言。

### (二) 面向对象思想的概述

**面向对象是基于面向过程的编程思想。**

面向过程，强调的是每一个功能的步骤。

面向对象，强调的是对象，然后由对象去调用功能来完成。

### (三) 面向对象思想的特点

- ① 是一种更符合我们思想习惯的思想；
- ② 可以将复杂的事情简单化，将我们从执行者变成为指挥者；

Eg:

把大象装进冰箱问题

面向过程：打开冰箱门---大象装进冰箱---关闭冰箱门

```
/*把大象装进冰箱的代码实现*/
class Demo
{
    public static void main(String[] args)
    {
        open();
        in();
        close();
    }
    public static void open()
    {
        //打开冰箱门;
    }
    public static void in()
    {
        //把大象装进冰箱;
    }
    public static void close()
    {
        //关闭冰箱门;
    }
}
```

面向对象：把大象装进冰箱的分析？（如何分析有哪些类？采用 UML、名词提取法）

A．有哪些类呢？

大象 冰箱 Demo

B．每个类有哪些东西呢？

大象：进去 冰箱：开门、关门 Demo：main 方法

C．类与类直接的关系是什么呢？

Demo 中使用大象和冰箱类的功能。

```
/*把大象装进冰箱的代码实现*/
class 大象
{
    public static void in()
    {
        //把大象装进冰箱;
    }
}
class 冰箱
{
    public static void open()
    {
        //打开冰箱门;
    }
    public static void close()
    {
        //关闭冰箱门;
    }
}
class Demo
{
    public static void main(String[] args)
    {
        冰箱调用开门;
        大象调用进去;
        冰箱调用关门;
    }
}
//虽然看起来比较复杂，但对于后期的维护非常便利
```

## 二、面向对象开发、设计、特征

**面向对象开发**，就是不断的创建对象，使用对象，指挥对象做事情。

**面向对象设计**，就是在管理和维护对象之间的关系。

**面向对象特征**，包括封装(encapsulation)、继承(inheritance)、多态(polymorphism)。

## 三、类与对象

现实世界中是如何描述一个事物的呢？

通过事物的属性和行为。而学习编程语言，是为了模拟现实世界的事物，把事物通过类来体现出来，从而得到了现实世界事物和类的对应关系（如下表）。

事物	类
属性，事物的信息描述	成员变量
行为，事物的功能	成员方法

**类**是一组相关属性和行为的集合(成员变量、成员方法、[构造方法](#))。例如：学生等。

**对象**是该类事物的具体体现。例如：班长等。

Eg :

学生事物

学生类

属性：姓名、年龄、地址

把事物转换成对应的类

成员变量：姓名、年龄、地址

行为：学习、吃饭、睡觉

成员方法：学习、吃饭、睡觉

成员方法，和以前的方法定义是一样的格式，但是把关键字 `static` 去掉；

如何使用呢？创建对象。

如何创建对象呢？

**格式：**类名 对象名 = `new` 类名();  
`Student S = new Student();`

如何使用成员变量呢？对象名.变量名

**如何使用成员方法呢？**对象名.方法名();

成员变量 和以前变量的定义是一样的格式，但是位置不同，在类中方法外；

Eg:

/\*学生类定义的代码实现\*/

```
class Student
{
    //姓名
    String name;
    //年龄
    int age;
    //地址
    String address;

    public void study()
    {
        //学习的方法
    }
    public void eat()
    {
        //吃饭的方法
    }
    public void sleep()
    {
        //睡觉的方法
    }
}
```

//学生测试类的代码实现(引用定义的学生类)

```
class StudentDemo
{
    public static void main(String[] args)
    {
        Student S = new Student();//创建对象

        //输出成员变量值
        System.out.println(S.name+"---"+S.age+"---"+S.address);
        //null-0-null
        S.name = "GHB";
        S.age = 24;
        S.address = "China";
        System.out.println(S.name+"---"+S.age+"---"+S.address);
        //GHB-24-China

        //输出成员方法
        S.study();
        S.eat();
        S.sleep();
    }
}
```

运行结果：

```
GHB-PC:Desktop Spinach$ ls
Student.java
GHB-PC:Desktop Spinach$ javac Student.java
GHB-PC:Desktop Spinach$ ls
Student.class      Student.java      StudentDemo.class
GHB-PC:Desktop Spinach$ java StudentDemo
null--0--null
GHB--24--China
GHB-PC:Desktop Spinach$
```

通过两次 `ls` 可以看出 `javac` 编译后会出现两个 `.class` 文件。

**提出问题：** `Student S = new Student();` 在内存中做了什么事情？

- 答：
1. 把 `Student.class` 文件加载到内存；
  2. 在栈内存给 `S` 变量开辟一个空间；
  3. 在堆内存为学生对象申请一个空间；
  4. 对学生对象的成员变量进行默认初始化。

**(一) 成员变量与局部变量的区别** (同类比: [静态变量与成员变量的区别](#))

- ① 在类中的位置不同  
成员变量：在类中方法外  
局部变量：在方法定义中或方法声明上
- ② 在内存中的位置不同  
成员变量：在堆内存  
局部变量：在栈内存
- ③ 生命周期不同  
成员变量：随着对象的创建而存在，随着对象的消失而消失  
局部变量：随着方法的调用而存在，随着方法的调用完毕而消失
- ④ 初始化值不同  
成员变量：有默认初始化值  
局部变量：没有默认初始化值，必须定义、赋值后才能使用

Eg:

```
class Student
{
    String name; //成员变量

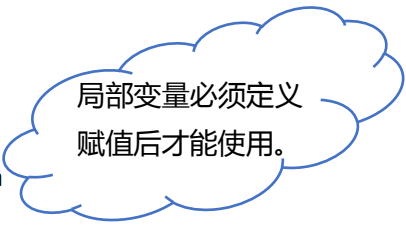
    public void study()
    {
        int num; //局部变量
        /*学习方法*/
        System.out.println(num);
    }
}

class StudentDemo
{
    public static void main(String[] args)
    {
        Student S = new Student();
        System.out.println(S.name);
        S.study();
    }
}
```

运行结果：

```
[GHB-PC:Desktop Spinach$ javac Student.java
Student.java:9: 错误: 可能尚未初始化变量 num
    System.out.println(num);
                    ^
```

1 个错误



局部变量必须定义  
赋值后才能使用。

## (二) 形式参数的问题

基本类型：形式参数的改变不影响实际参数。【值传递】

引用类型：形式参数的改变直接影响实际参数。【地址传递】( 示例参考 [匿名对象](#) )

## (三) 匿名对象的概述和应用

匿名对象：没有名字的对象。

匿名对象的应用场景：A. 调用方法，仅仅只调用一次方法的时候；

B. 可以作为实际参数传递；

Eg:

```
/*匿名对象的代码实现*/
class Student{
    public void show(){
        System.out.println("I am GHB.");
    }
}
class StudentDemo{
    public void method(Student s){
        s.show();
    }
}
class NoNameDemo{
    public static void main(String[] args) {
        Student S = new Student();//带名字的调用
        S.show();
        //匿名对象
        new Student();
        //匿名对象的调用方法
        new Student().show();
        //匿名对象作为实际参数传递
        new StudentDemo().method(new Student());
    }
}
```

## 四、封装

隐藏对象的属性和实现细节，仅对外提供公共访问方式，即为**封装**。

优点：A. 隐藏实现细节，提供公共的访问方式；

B. 提高了代码的复用性；

C. 提高安全性；

将不需要对外提供的内容都隐藏起来，把属性隐藏，提供公共方法对齐访问便是**封装原则**。

### (一) private 关键字

private 是一个权限修饰符；

可以修饰成员（成员变量和成员方法）；

被 private 修饰的成员只在类中才能访问。

private 仅仅是封装的一种体现，类和方法其实也是封装体。

private 最常用的应用：把成员变量、成员方法用 private 修饰（如 Eg1）；

提供对应 getXxx()/setXxx()方法（如 Eg2）；

一个标准的案例的使用。

Eg1:

```
/*private关键字的应用代码实现*/
class Demo{
    private int num = 10;          //用private修饰
    public void show(){
        System.out.println("I am GHB");
    }

    private void method(){        //用private修饰
        System.out.println("Hello World");
    }
    public void show1(){
        method();
    }
}

class PrivateDemo{
    public static void main(String[] args) {
        Demo S = new Demo();
        //System.out.println(S.num); //不能访问私有的成员变量
        S.show(); //运行结果: I am GHB

        //S.method(); //不能访问私有的成员方法
        S.show1(); //运行结果: Hello World
    }
}
```

Eg2:

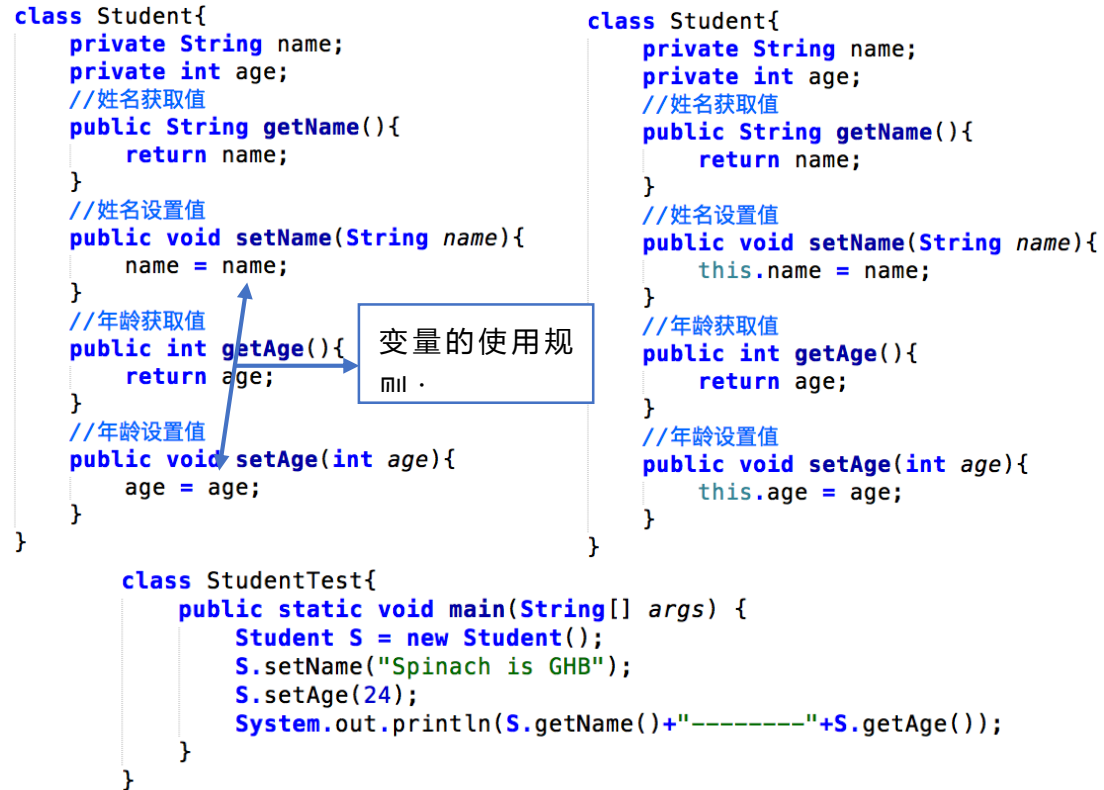
```
/*封装和private的应用*/
class Student{
    private String name;
    private int age;
    //姓名获取值
    public String getName(){
        return name;
    }
    //姓名设置值
    public void setName(String n){
        name = n;
    }
    //年龄获取值
    public int getAge(){
        return age;
    }
    //年龄设置值
    public void setAge(int m){
        age = m;
    }
}

class StudentTest{
    public static void main(String[] args) {
        Student S = new Student();
        System.out.println(S.getName()+"--"+S.getAge());
        //运行结果: null--0
        S.setName("Spinach");
        S.setAge(24);
        System.out.println(S.getName()+"--"+S.getAge());
        //运行结果: Spinach--24
    }
}
```

## (二) this 关键字

曾针对代码变量名称有一句话, : “起名字要做到见名知意”。见(一)private 关键字  
Eg2 中 `public void setName(String n){name = n;}`和 `public void setAge(int m){age = m;}`对于 n、m 的参数变量的起名并不是见名知意, 因此对后期会造成不便。

Eg :



```

class Student{
    private String name;
    private int age;
    //姓名获取值
    public String getName(){
        return name;
    }
    //姓名设置值
    public void setName(String name){
        name = name;
    }
    //年龄获取值
    public int getAge(){
        return age;
    }
    //年龄设置值
    public void setAge(int age){
        age = age;
    }
}

class StudentTest{
    public static void main(String[] args) {
        Student S = new Student();
        S.setName("Spinach is GHB");
        S.setAge(24);
        System.out.println(S.getName()+"-----"+S.getAge());
    }
}

```

```

class Student{
    private String name;
    private int age;
    //姓名获取值
    public String getName(){
        return name;
    }
    //姓名设置值
    public void setName(String name){
        this.name = name;
    }
    //年龄获取值
    public int getAge(){
        return age;
    }
    //年龄设置值
    public void setAge(int age){
        this.age = age;
    }
}

```

变量的使用规范  
mll .

运行结果 :

```
GHB-PC:Desktop Spinach$ java StudentTest
null-----0
```

运行结果 :

```
GHB-PC:Desktop Spinach$ java StudentTest
Spinach is GHB-----24
```

第一种运行结果是由于变量遵循就近原则, 故只是自身的赋值而已, 为了解决此问题, 引出 this 关键字 (哪个对象调用那个方法, this 就代表那个对象), 则出现第二种运行结果。



### (三) 构造方法及其重载 (同比与: [方法重写](#))

构造方法就是给对象的数据进行初始化。

格式: A. 方法名与类名相同;

B. 没有返回值类型, 连 void 都没有;

C. 没有具体的返回值;

**注意事项:** A. 如果程序员没有给出构造方法, 系统会自动提供一个无参构造方法(通过反编译工具查看.class 文件可看到系统提供的无参构造方法);

B. 如果程序员给出构造方法(无论带参还是不带参), 系统将不再提供默认  
的无参构造方法, 如果还想使用无参构造方法, 就只能自己给出;

```
/*构造方法及其重载*/
class ConstructStudent{
    private String name;
    private int age;

    public ConstructStudent(){
        System.out.println("这是无参构造方法");
    }

    //构造方法的重载格式
    public ConstructStudent(String name){
        System.out.println("这是带一个String类型的构造方法");
        this.name = name;
    }
    public ConstructStudent(int age){
        System.out.println("这是带一个int类型的构造方法");
        this.age = age;
    }
    public ConstructStudent(String name,int age){
        System.out.println("这是带String类型和int类型的构造方法");
        this.name = name;
        this.age = age;
    }

    public void show(){
        System.out.println(name+"-----"+age);
    }
}

class ConstructDemo{
    public static void main(String[] args) {
        ConstructStudent S1 = new ConstructStudent();
        S1.show();
        System.out.println("-----");

        ConstructStudent S2 = new ConstructStudent("GHB");
        S2.show();
        System.out.println("-----");

        ConstructStudent S3 = new ConstructStudent(24);
        S3.show();
        System.out.println("-----");

        ConstructStudent S4 = new ConstructStudent("Spinach",24);
        S4.show();
    }
}
```

**运行结果：**

```

[GHB-PC\Desktop Spinach$ javac ConstructDemo.java
[GHB-PC\Desktop Spinach$ java ConstructDemo
这是无参构造方法
null----0
-----
这是带一个String类型的构造方法
GHB----0
-----
这是带一个int类型的构造方法
null----24
-----
这是带String类型和int类型的构造方法
Spinach----24

```

关于构造方法及其重载的解释（如上面代码）：①若符合注意事项 A 则为构造方法，若符合注意事项 B 则为构造方法的重载（**方法重载 Overload 即为类名相同，但参数不同的构造方法，与返回值类型无关**）；②可看出此代码符合注意事项 B，public ConstructStudent()为无参构造方法，由于 public ConstructStudent(String name)、public ConstructStudent(String name,int age)、public ConstructStudent(int age)带参数构造方法的出现，从而构成了构造方法的重载；③其中 public void show()为成员方法，private String name、private int age 为成员变量。

**(四) static 关键字**

static 关键字(可以修饰成员变量也可以修饰成员方法)特点：

- A. 随着类的加载而加载；
- B. 优先于对象存在；
- C. 被类的所有对象共享(若某成员变量是被所有对象共享的，则应定义为静态变量)；
- D. 可以通过类名调用(也可通过对象名调用)。

Eg1:

```

/*static关键字的使用*/
class StaticDmeo{
    String name;
    int age;
    static String country;

    public StaticDmeo(){ }
    public StaticDmeo(String name,int age){
        this.name = name;
        this.age = age;
    }
    public StaticDmeo(String name,int age,String country){
        this.name = name;
        this.age = age;
        this.country = country;
    }

    public void show(){
        System.out.println("姓名"+name+", 年龄"+age+", 国籍"+country);
    }
}

class GHBTes{
    public static void main(String[] args) {
        StaticDmeo SD1 = new StaticDmeo();
        SD1.show();//姓名null, 年龄0, 国籍null
        StaticDmeo SD2 = new StaticDmeo("GHB",24,"China");
        SD2.show();//姓名GHB, 年龄24, 国籍China
        StaticDmeo SD3 = new StaticDmeo("Spinach",23);
        SD3.show();//姓名Spinach, 年龄23, 国籍China
        //country在SD2中被赋值, 由于其为静态变量, 所以在SD3中会打印出来。
    }
}

```

可分析出，针对多个对象有共同的成员变量值时，可采用 `static` 关键字作为修饰符，可以避免堆内存开辟过多的空间。

**注意事项：**A. 在静态方法中是没有 `this` 关键字的；

(如何理解？`static` 是随着类的加载而加载，`this` 是随着对象的创建而存在，静态比对象先存在，只能后存在的能访问先存在的。)

B. 静态方法只能访问静态的成员变量和静态的成员方法(如 Eg2 的对比)。

Eg2：

#### 1. 函数体中程序顺序执行

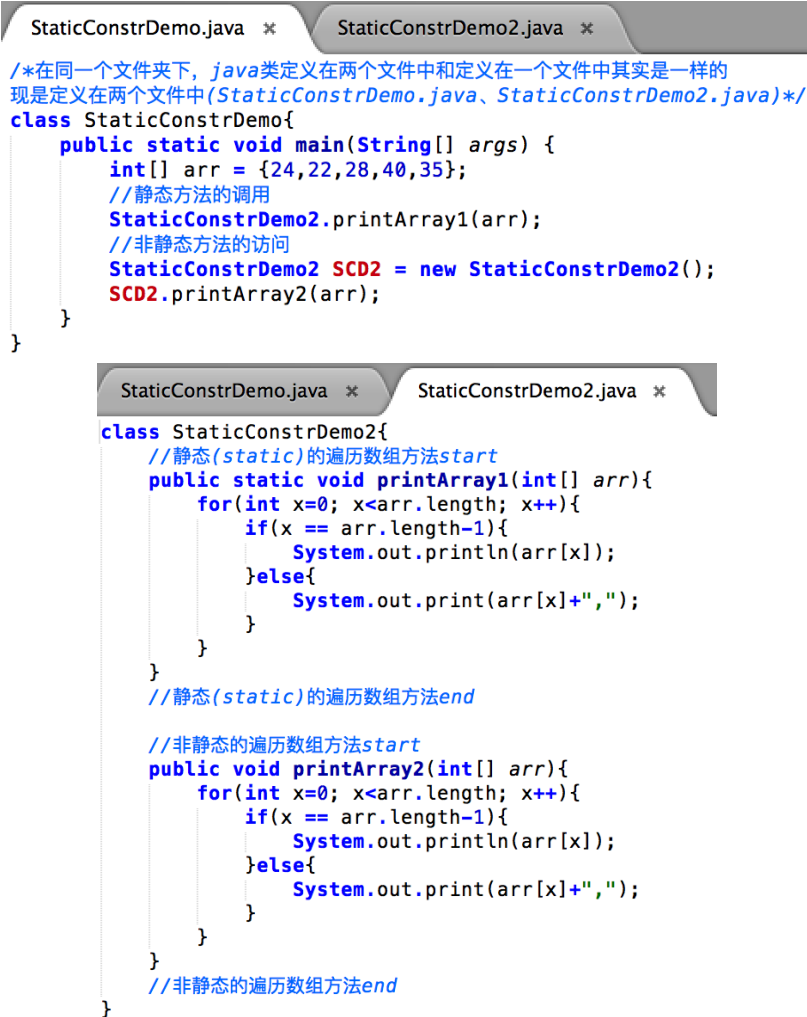
```
class StaticConstrDemo{
    public static void main(String[] args) {
        int[] arr = {24,22,28,40,35};
        //遍历数组start
        for(int x=0; x<arr.length; x++){
            if(x == arr.length-1){
                System.out.println(arr[x]);
            }else{
                System.out.print(arr[x]+"，");
            }
        }
        //遍历数组end
    }
}
```

#### 2. 同一文件中，对静态和非静态两个不同调用方式进行对照

```
class StaticConstrDemo{
    public static void main(String[] args) {
        int[] arr = {24,22,28,40,35};
        //静态方法的调用
        printArray1(arr); //由于静态方法可以直接调用静态的
        //非静态方法的访问
        StaticConstrDemo SCD = new StaticConstrDemo();
        SCD.printArray2(arr); //非静态方法需要通过创建对象来调用
    }
    //静态(static)的遍历数组方法start
    public static void printArray1(int[] arr){
        for(int x=0; x<arr.length; x++){
            if(x == arr.length-1){
                System.out.println(arr[x]);
            }else{
                System.out.print(arr[x]+"，");
            }
        }
    }
    //静态(static)的遍历数组方法end

    //非静态的遍历数组方法start
    public void printArray2(int[] arr){
        for(int x=0; x<arr.length; x++){
            if(x == arr.length-1){
                System.out.println(arr[x]);
            }else{
                System.out.print(arr[x]+"，");
            }
        }
    }
    //非静态的遍历数组方法end
}
```

## 3. 同一文件夹下的两个文件中，对静态和非静态两个不同调用方式进行对照



```
StaticConstrDemo.java × StaticConstrDemo2.java ×
/*在同一个文件夹下，java类定义在两个文件中和定义在一个文件中其实是一样的
现是定义在两个文件中(StaticConstrDemo.java、StaticConstrDemo2.java)*/
class StaticConstrDemo{
    public static void main(String[] args) {
        int[] arr = {24,22,28,40,35};
        //静态方法的调用
        StaticConstrDemo2.printArray1(arr);
        //非静态方法的访问
        StaticConstrDemo2 SCD2 = new StaticConstrDemo2();
        SCD2.printArray2(arr);
    }
}

StaticConstrDemo.java × StaticConstrDemo2.java ×
class StaticConstrDemo2{
    //静态(static)的遍历数组方法start
    public static void printArray1(int[] arr){
        for(int x=0; x<arr.length; x++){
            if(x == arr.length-1){
                System.out.println(arr[x]);
            }else{
                System.out.print(arr[x]+",");
            }
        }
    }
    //静态(static)的遍历数组方法end

    //非静态的遍历数组方法start
    public void printArray2(int[] arr){
        for(int x=0; x<arr.length; x++){
            if(x == arr.length-1){
                System.out.println(arr[x]);
            }else{
                System.out.print(arr[x]+",");
            }
        }
    }
    //非静态的遍历数组方法end
}
```

**(五) 静态变量与成员变量的区别** (同类比：[成员变量与局部变量的区别](#))

- ① 所属不同  
静态变量：属于类，又称为类变量  
成员变量：属于对象，又称为实例变量（对象变量）
- ② 在内存中的位置不同  
静态变量：存储于方法区的静态区  
成员变量：存储于堆内存
- ③ 内存出现时间不同  
静态变量：随着类的加载而加载，随着类的消失而消失  
成员变量：随着对象的创建而存在，随着对象的消失而消失
- ④ 调用不同  
静态变量：可以通过类名调用，也可以通过对象调用  
成员变量：只能通过对象名调用

## (六) main 格式及详细解释

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

main 方法格式的详细解释	
public	公共的，访问权限是最大的。由于 main 方法是被 jvm 调用，故权限够大。
static	静态的，不需要创建对象，通过类名调用，方便被 jvm 调用。
main	一个常见的方法入口，大多数语言都是以 main 作为程序入口的。
String[] args	一个字符串的数组，args 是一个字符串数组的变量名，不是关键字，是 arguments 的缩写，只是一个默认名，一般都习惯性照写（变量名可以任意起名）。
{... ...}	代码块或函数体。

提出问题：String[] args 到底有什么作用呢？怎么给值呢？值又去哪里了？

答：早期是为了接收键盘录入的数据，其格式为 java 文件名 Hello World Java

Eg1：

```
class MainDemo{
    public static void main(String[] args) {
        System.out.println(args);
        System.out.println(args.length);
        System.out.println(args[0]);
    }
}
```

运行结果：

```
[GHB-PC:Desktop Spinach$ javac MainDemo.java
[GHB-PC:Desktop Spinach$ java MainDemo
[Ljava.lang.String;@7852e922
0
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
at MainDemo.main(MainDemo.java:6)
```

Eg2：

```
class MainDemo{
    public static void main(String[] args) {
        System.out.println(args);
        System.out.println(args.length);
        for(int x=0; x<args.length; x++){
            System.out.println(args[x]);
        }
    }
}
```

运行结果：

```
[GHB-PC:Desktop Spinach$ javac MainDemo.java
[GHB-PC:Desktop Spinach$ java MainDemo Hello World Java
[Ljava.lang.String;@7852e922
3
Hello
World
Java
```

## (七) 代码块

在 java 中用{}括起来的代码被称为代码块。根据其位置和声明的不同，可以分为局部代码块、构造代码块、静态代码块。

局部代码块：局部位置，用于限定变量的生命周期，以早释放，提高内存利用率。

构造代码块：在类中的成员位置，用{}括起来的代码，每次调用构造方法执行前，都会先执行构造代码块。其作用：可以把多个构造方法中的共同代码放在一起，对对象进行初始化。

静态代码块：在类中的成员位置，用{}括起来的代码，只不过它用了 static 修饰。其作用：一般是对类进行初始化。

Eg1 :

```
class Code{
    //静态代码块
    static {
        int a = 23;
        System.out.println(a);
    }
    //构造代码块
    {
        int x = 22;
        System.out.println(x);
    }
    //构造方法
    public Code(){
        System.out.println("Code");
    }
    //构造代码块
    {
        int y = 24;
        System.out.println(y);
    }
    //静态代码块
    static {
        int b = 25;
        System.out.println(b);
    }
}

class CodeDemo{
    public static void main(String[] args) {
        Code C1 = new Code();
        System.out.println("-----");
        Code C2 = new Code();
        System.out.println("-----");
        Code C3 = new Code();
    }
}
```

运行结果：

```
[GHB-PC:Desktop Spinach$ javac Code.java
[GHB-PC:Desktop Spinach$ java CodeDemo
23
25
22
24
Code
-----
22
24
Code
-----
22
24
Code
```

引出面试题：

构造方法、构造代码块、静态代码块的执行顺序？

答：执行顺序为先静态代码块，然后构造代码块，最后构造方法。

## 五、制作和使用帮助文档

### (一) 制作说明书

如何制作说明书？

- A. 写一个工具类；
- B. 对这个类加入文档注释（注释格式和内容在例 Eg1 中体现）；
- C. 用工具解析文档注释（javadoc 工具，如 Eg2）；
- D. 格式（如 Javadoc -d 目录 -author -version StaticConstrDemo.java）。

Eg1：

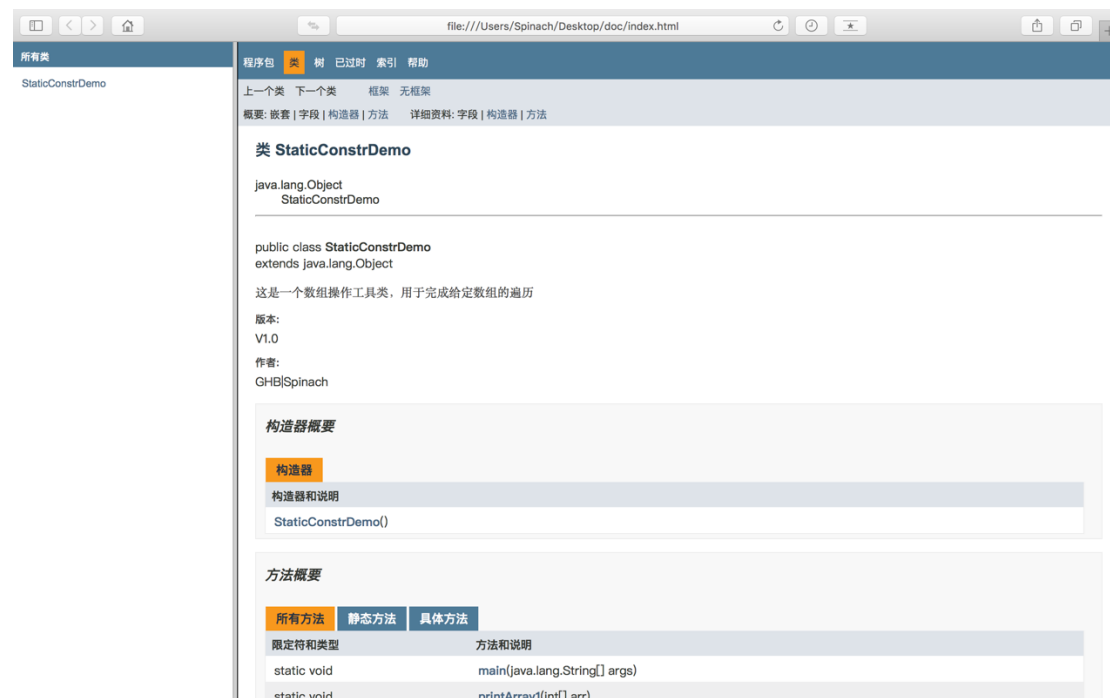
```
/**
 *这是一个数组操作工具类，用于完成给定数组的遍历
 *@author   GHB|Spinach
 *@version   V1.0
 */
public class StaticConstrDemo{
    public static void main(String[] args) {
        int[] arr = {24,22,28,40,35};
        printArray1(arr);
    }
    /**
     *静态遍历数组的方法，遍历后的格式为元素1，元素2，元素3，...
     *@param arr  被遍历的数组
     */
    public static void printArray1(int[] arr){
        for(int x=0; x<arr.length; x++){
            if(x == arr.length-1){
                System.out.println(arr[x]);
            }else{
                System.out.print(arr[x]+",");
            }
        }
    }
}
```

Eg2：

```
[GHB-PC:Desktop Spinach$ javadoc -d doc -author -version StaticConstrDemo.java ]
正在加载源文件StaticConstrDemo.java...
正在构造 Javadoc 信息...
正在创建目标目录："doc/"
标准 Doclet 版本 1.8.0_91
正在构建所有程序包和类的树...
正在生成 doc/StaticConstrDemo.html...
正在生成 doc/package-frame.html...
正在生成 doc/package-summary.html...
正在生成 doc/package-tree.html...
正在生成 doc/constant-values.html...
正在构建所有程序包和类的索引...
正在生成 doc/overview-tree.html...
正在生成 doc/index-all.html...
正在生成 doc/deprecated-list.html...
正在构建所有类的索引...
正在生成 doc/allclasses-frame.html...
正在生成 doc/allclasses-noframe.html...
正在生成 doc/index.html...
正在生成 doc/help-doc.html...
GHB-PC:Desktop Spinach$ █
```

则在桌面上生成一个 doc 文件夹，里面便保存着生成的对该类的说明书。

在生成的文件中，点击 index.html，查看如下：



## (二) 使用说明书

java 中有很多像这种类似的类供使用，但如何使用呢？

只需把所在类的 .Class 文件与想用该类的文件放于同一文件夹下，并通过查看说明书的来得知类中方法的功能，即可直接调用该类中的方法。

## (三) 使用 JDK 提供的 API 帮助文档

API(application programming interface)应用程序编程接口，又称帮助文档。

步骤：1. 打开帮助文档；

2. 点击显示，找到索引，找到输入框；

3. 知道需要找类名（例如 Scanner）；

4. 在输入框里面输入 Scanner，然后回车；

5. 看所属包，所属 java.util.Scanner

（注：java.lang 包下的类不需要导入，其他的全部需要导入）；

6. 查看类的解释和说明，以及给来的版本；

7. 查看类的结构；

成员变量——字段摘要

成员方法——方法摘要

构造方法——构造方法摘要

8. 学习构造方法；

若有构造方法，就直接创建对象

若无构造方法，由于成员都是静态，则可直接被调用

9. 看成员方法；

是否静态、返回值类型、方法名、参数列表

【注】此部分的使用和实践可以参考附件—[案例 1. 猜数字小游戏](#)来理解。



## 六、继承

## (一) 引出继承的概念

```

/*例如：学生类和教师类*/
class Student{
    String name;
    int age;
    public void eat(){
        System.out.println("我要吃饭");
    }
    public void sleep(){
        System.out.println("我要呼噜噜");
    }
}

class Teacher{
    String name;
    int age;
    public void eat(){
        System.out.println("我要吃饭");
    }
    public void sleep(){
        System.out.println("我要呼噜噜");
    }
}

//测试类
class TestDemo{
    public static void main(String[] args) {
        Student S = new Student();
        S.eat();
        S.sleep();
        System.out.println("-----");
        Teacher T = new Teacher();
        T.eat();
        T.sleep();
    }
}

```

如左例，发现学生类和教师类都有同样的成员变量(String name、int age)和成员方法(public void eat、public void sleep)，如果再继续定义类(例如工人类、军人类等)，依然会出现相同的成员变量和成员方法，因此就造成了代码的重复，为了解决这种情况的问题，java 就提供了一种技术叫做继承。

多个类中存在相同属性和行为时，将这些内容抽取到单独一个类中，那么多个类无需再定义这些属性和行为，只要继承那个类即可，而实现类与类的继承通过 extends 关键字，其格式：`class 子类名 extends 父类名 {}`，那么单独的这个类成为父类、基类或超类，这多个类则成为子类或派生类。

以上代码使用继承后代码如下：

```

/*使用继承后*/
class Person{
    String name;
    int age;
    public void eat(){
        System.out.println("我要吃饭");
    }
    public void sleep(){
        System.out.println("我要呼噜噜");
    }
}

class Teacher extends Person {}
class Student extends Person {}

class TestDemo2{
    public static void main(String[] args) {
        Student S = new Student();
        S.eat();
        S.sleep();
        System.out.println("-----");
        Teacher T= new Teacher();
        T.eat();
        T.sleep();
    }
}

```

## (二) 继承的利与弊

- 利：1.提高代码的复用性；  
2.提高代码的维护性；  
3.让类与类之间产生了关系，是多态的前提。  
(其实也是继承的一个弊端：类的耦合性很强)
- 弊：1.类的耦合性变强；  
2.打破了封装性。

## (三) 继承的特点及注意事项

### 特点：

- 1.Java 只支持单继承，不支持多继承，一个类只能有一个父类，不可以有多个父类。

Eg1: `class Student extends Person{}` //正确的  
`class Student extends Person1,Person2, ...` //错误的

- 2.Java 支持多层继承 (继承系)。

Eg2: `class A {}`  
`class B extends A {}`  
`class C extends B {}`

### 注意事项：

- 1.子类只能继承父类的所有非私有的成员。  
(其实这也体现了继承的另一个弊端：打破了封装性)
- 2.子类不能继承父类的构造方法，但可以通过 [super 关键字](#)去访问父类的构造方法。
- 3.不要为了部分功能而去继承。

## (四) 继承中的不同关系

### 一) 继承中成员变量的关系

如下例，对两种情况进行讨论：

- 1.子类中的成员变量和父类中的成员变量名称不一样
- 2.子类中的成员变量和父类中的成员变量名称一样

```
class Father{
    public int num = 10;
    public void method(){
        int num = 20;
    }
}

class Son extends Father {
    int num = 30;
    int num2 = 40;
    public void Show(){
        int num = 50;
        System.out.println(num);
        System.out.println(num2);
    }
}

class Extends3Demo{
    public static void main(String[] args) {
        Son S = new Son();
        S.Show();
    }
}
```

运行结果：

50  
40

解释分析：

针对第一种情况，`System.out.println(num)`、`System.out.println(num2)`中的变量直接继承父类中的变量或直接使用子类中的变量；

针对第二种情况，`System.out.println(num)`、`System.out.println(num2)`中的变量采用就近原则进行调用，其在子类方法中访问一个变量（就近原则）的查找顺序：首先在子类方法的局部范围找，有就使用，若没有再在子类的成员范围找，有就使用，若没有再在父类的成员范围找，有就使用，若还未找到就报错。

**提出问题：**除了想输出局部范围的 `num`，还想输出本类范围的 `num`，还想输出父类范围的 `num`，怎么办？

答：输出局部范围的 `num` 采用变量的就近原则，而输出本类范围的 `num` 采用 `this` 关键字，但输出父类范围的 `num`，怎么办呢？Java 提供了 [super 关键字](#) 以解决这一问题。

## 二) 继承中构造方法的关系

特点：1. 子类中所有的构造方法默认都会访问父类中无参构造方法。（为什么呢？因为子类会继承父类中的数据，可能还会使用父类的数据，所以子类初始化之前，一定要先完成父类的数据的初始化。）

2. 子类每一个构造方法的第一条语句默认都是：`super()`；。

```
class Father {
    public Father(){
        System.out.println("父类的无参构造方法");
    }
    public Father(String name){
        System.out.println("父类的带参构造方法");
    }
}
class Son extends Father {
    public Son(){
        super();//默认是有这条语句的，写与不写都一样
        System.out.println("子类的无参构造方法");
    }
    public Son(String name){
        super();//默认是有这条语句的，写与不写都一样
        System.out.println("子类的带参构造方法");
    }
}
class ContrExtendDemo{
    public static void main(String[] args) {
        Son S = new Son();
        System.out.println("-----");
        Son S2 = new Son("啾啾菜");
        System.out.println("-----");
    }
}
```

**运行结果：**

```
GHB-PC:Desktop Spinach$ javac ContrExtend.java
GHB-PC:Desktop Spinach$ java ContrExtendDemo
父类的无参构造方法
子类的无参构造方法
-----
父类的无参构造方法
子类的带参构造方法
```

**提出问题：** 如果父类没有无参构造方法，那么子类的构造方法会出现什么现象？如何解决呢？

- 答：
1. 在父类中加一个无参构造方法；
  2. 子类通过 `super` 去显示调用父类其他的带参的构造方法，如 Eg1；
  3. 子类通过 `this` 去调用本类的其他构造方法，如 Eg2（子类中一定要有一个去访问了父类的构造方法，否则父类数据就没有初始化）；

Eg1：

```
class Father {
    public Father(String name){
        System.out.println("父类的带参构造方法");
    }
}
class Son extends Father {
    public Son(){
        super("随便给");
        System.out.println("子类的无参构造方法");
    }
    public Son(String name){
        super("随便给");
        System.out.println("子类的带参构造方法");
    }
}
class ContrExtendDemo{
    public static void main(String[] args) {
        Son S = new Son();
        System.out.println("-----");
        Son S2 = new Son("啦啦菜");
        /*运行结果：父类的带参构造方法
                   子类的无参构造方法
                   -----
                   父类的带参构造方法
                   子类的带参构造方法
        */
    }
}
```

Eg2：

```
class Father {
    public Father(String name){
        System.out.println("父类的带参构造方法");
    }
}
class Son extends Father {
    public Son(){
        super("随便给");
        System.out.println("子类的无参构造方法");
    }
    public Son(String name){
        this();
        System.out.println("子类的带参构造方法");
    }
}
class ContrExtendDemo{
    public static void main(String[] args) {
        Son S = new Son();
        System.out.println("-----");
        Son S2 = new Son("啦啦菜");
        /*运行结果：父类的带参构造方法
                   子类的无参构造方法
                   -----
                   父类的带参构造方法
                   子类的无参构造方法
                   子类的带参构造方法
        */
    }
}
```

注：`this(...)`或`super(...)`必须出现在第一条语句上，如果不放在第一条语句上，可能对父类的数据进行了多次初始化。

### 三) 继承中成员方法的关系

如下例，对两种情况进行讨论：

1. 子类中的方法和父类中的方法声明不一样
2. 子类中的方法和父类中的方法声明一样

/\*继承中成员方法的关系\*/

```
class Father{
    public void show(){
        System.out.println("show Father");
    }
}
class Son extends Father {
    public void method(){
        System.out.println("method Son");
    }
    //针对第二种情况加上此句
    public void show(){
        System.out.println("show Son");
    }
}
class ExtendsDemo{
    public static void main(String[] args) {
        Son S = new Son();
        S.show();
        S.method();
    }
}
```

运行结果：

```
show Son
method Father
```

解释分析：

针对第一种情况，S.show()、S.method()的成员方法直接继承父类中的相应的成员方法或直接使用子类中的相应的成员方法；

针对第二种情况，S.show()、S.method()的成员方法，采取以下顺序进行查找：首先找子类中是否有对应方法，有就使用，若没有再找父类中是否有对应的方法，有就使用，若还未找到就报错。

【注】学完继承的所有知识后可结合附件— [案例 2.关于继承的代码（读代码写结果）](#) 做下练习 和 补充内容 [方法重写](#)

### （五）this 与 super 的区别

this 代表本类对应的应用。

super 代表父类存储空间的标识(可以理解为父类引用，可以操作父类的成员)。

格式：

this.成员变量	super.成员变量
this.成员方法	super.成员方法
this(...)	调用构造方法
super(...)	调用构造方法

```
class Father{
    public int num = 10;
    public void FatherShow(){
        System.out.println("我是父类");
    }
}

class Son extends Father {
    public int num = 20;
    public void SonShow(){
        int num = 30;
        System.out.println(num);
        System.out.println(this.num);
        System.out.println(super.num);
        super.FatherShow();
    }
}

class Extends4Demo{
    public static void main(String[] args) {
        Son S = new Son();
        S.SonShow();
        /*运行结果: 30
                   20
                   10
                   我是父类
        */
    }
}
```

### （六）方法重写（同比与：[方法重载](#)）

方法重写 Override 即为子类中存在和父类中方法声明一模一样的方法，也称方法重写、方法覆盖。

使用特点：1.如果方法名不同，就调用对应的方法；

2.如果方法名相同，最终使用子类中对应的方法。

可参考 [三\) 继承中成员方法的关系](#) 来理解

方法重写的应用：

当子类需要父类的功能，而功能主体子类有自己的特有内容时，可以重写父类中的方法，这样即沿袭了父类的功能，又定义了子类的特有的内容（如下例）。

```

/*方法重写的应用*/
class Phone{
    public void call(String name){
        System.out.println("给"+name+"打电话");
    }
}
class NewPhone extends Phone {
    public void call(String name){
        super.call(name);
        System.out.println("给"+name+"视频聊天");
    }
}
class MethodDemo{
    public static void main(String[] args) {
        NewPhone NP = new NewPhone();
        NP.call("GHB");
    }
}

```

运行结果：

给 GHB 打电话

给 GHB 视频聊天

注意事项：

1. 父类中的私有方法不能被重写（因为父类私有方法，子类根本不能继承）；
2. 子类重写父类时，访问权限不能更低（大于等于父类的访问权限，最好一致）；
3. 父类静态方法，子类也必须通过静态方法进行重写（在多态中会细讲）。

### （七）final 关键字

由于继承中方法有一个现象：方法重写，所以父类的功能就会被子类给覆盖掉，而有时候，我们不想让子类去覆盖掉父类的功能，针对这一种情况，Java 提供了一个关键字：final 关键字。final 常见可以修饰类、方法、变量。

**特点：**final 修饰类，该类不能被继承，由于为最终类。

final 修饰方法，该方法不能被重写（也称为覆盖、复写）。

final 修饰变量，该变量不能重新被赋值，由于这个变量其实就是常量。

Eg1：

```

1  /*final关键字，可以修饰类、方法、变量*/
2  class Fu{
3      public int num = 10;
4      public final int num2 = 20;
5      public void show(){
6          System.out.println(num);
7          System.out.println(num2);
8      }
9  }
10 class Zi extends Fu {
11     public void show(){
12         num = 100;
13         num2 = 200;
14         System.out.println(num);
15         System.out.println(num2);
16     }
17 }
18 class FinalDemo{
19     public static void main(String[] args) {
20         Zi Z = new Zi();
21         Z.show();
22     }
23 }

```

运行结果：

```

[GH-B-PC:Desktop Spinach$ javac FinalDemo.java
FinalDemo.java:15: 错误：无法为最终变量num2分配值
    num2 = 200;
    ^

```

1 个错误

面试题知识点：final 修饰局部变量的问题。

Eg2:

```
/*final修饰局部变量*/
class Student{
    public int age = 24;
}
class FinalDemo2{
    public static void main(String[] args) {
        //局部变量是基本数据类型
        int a = 10;
        a = 100;
        System.out.println(a);

        final int b = 20;
        //b = 200;无法为最终变量b分配值
        System.out.println(b);
        System.out.println("-----");

        //局部变量是引用数据类型
        Student St = new Student();
        System.out.println(St.age);
        St.age = 25;
        System.out.println(St.age);
        System.out.println("-----");

        final Student St2 = new Student();
        System.out.println(St2.age);
        St2.age = 22;
        System.out.println(St2.age);
    }
}
```

运行结果：

```
100
20
-----
24
25
-----
24
22
```

提出问题：针对最后一种情况（final Student St2 = new Student();），为什么“St2.age = 22;”可以正常赋值呢？

答：因为创建的新对象 St2 是被 final 关键字修饰的，即其对象的地址被固定的，而“St2.age = 22;”是给对象中的变量进行赋值，故不影响。假如在其后面加入一句代码 St2 = new Student();，此时就会报错为无法为最终变量 St2 分配值，由于“St2 = new Student();”是为 St2 重新分配一次空间，而 St2 是被 final 修饰的（其对象的地址已被固定了），所以此种情况会报错。

通过上例得出结论：final 修饰基本类型，是值不能被改变；

final 修饰引用类型，是地址值不能被改变。

final 修饰变量的初始化时机：1. 被 final 修饰的变量只能赋值一次

2. 在构造方法完毕前（非静态的变量）



于 2016/8/25 编著完成



## 附件

## 案例 1. 猜数字小游戏

猜数字小游戏（数据在 1-100 之间）。

- 分析：
1. 程序产生一个随机数；
  2. 键盘录入数据，进行猜；
  3. 对录入数据和随机数进行比较，并显示“猜中”、“大了”、“小了”以作提示；
  4. 给出有限次猜的机会，猜中就结束，否则直接结束。

```
/*猜数字小游戏（数据在1-100之间），最多可猜10次*/
import java.util.Scanner;
//import java.lang.Math 由于其属于java.lang包，故不需要导入
class GuessNumgame{
    public static void main(String[] args) {
        //程序产生的一个随机数字
        int number = (int)(Math.random()*100)+1;
        //提示录入数据
        System.out.println("请输入你猜的数字(1-100)");

        Scanner sc = new Scanner(System.in);
        int guessNum;
        int n = 10, flag = 0; //限制次数
        //作比较
        while(n != 0 && flag == 0)
        {
            //录入数据
            guessNum = sc.nextInt();

            if (guessNum < number) {
                System.out.println("小了，请输入大于"+guessNum+"的数");
            } else if (guessNum == number) {
                System.out.println("哇~ 恭喜你猜中数字: "+guessNum);
                flag = 1;
            } else {
                System.out.println("大了，请输入小于"+guessNum+"的数");
            }
            n--;
        }
    }
}
```

## 运行结果：

```
[GHB-PC:Desktop Spinach$ javac GuessNumgame.java
[GHB-PC:Desktop Spinach$ java GuessNumgame
请输入你猜的数字(1-100)
20
小了，请输入大于20的数
50
小了，请输入大于50的数
80
小了，请输入大于80的数
90
小了，请输入大于90的数
100
大了，请输入小于100的数
95
大了，请输入小于95的数
93
大了，请输入小于93的数
91
小了，请输入大于91的数
92
哇~ 恭喜你猜中数字：92
```



## 案例 2. 有关继承的代码练习（读代码写结果）

以下代码是有关继承的代码练习，请读代码写结果。

```
/*读代码写结果*/
class Xx{
    Yy a = new Yy();
    Xx(){
        System.out.print("Xx,");
    }
}

class Yy{
    Yy(){
        System.out.print("Yy,");
    }
}

public class Zz extends Xx{
    Yy b = new Yy();
    Zz(){
        System.out.println("Zz");
    }
    public static void main(String[] args) {
        new Zz();
    }
}
```

运行结果：

```
GHB-PC:Desktop Spinach$ javac Zz.java
GHB-PC:Desktop Spinach$ java Zz
Yy,Xx,Yy,Zz
```

分析：

首先 main 方法执行 Zz()，而 Zz() 的父类是 Xx()，在类 Xx() 时，创建对象 a 需要初始化执行 Yy()，则打印出 Yy，，然后执行 Xx 类中的构造方法 Xx()，则打印出 Xx，，父类初始化完毕后，进入子类 Zz 创建 b 需要执行 Yy()，则又打印出 Yy，，最后执行 Zz 子类中的构造方法 Zz()，则打印出 Zz，，故运行结果为 Yy,Xx,Yy,Zz。