1、课程名称：面向对象（基础）

2、知识点

2.1、上次课程的主要知识点

1、 类与对象

- 类是一种抽象的表示共性的集合，类是由属性及方法所组成的。
- 对象是类的具体表现，表示一个个个性的产物。
- 对象的产生格式： 类名称 对象名称 = new 类名称() ;
  - |- 声明对象：类名称 对象名称 = null ; à 栈内存中开辟
  - |- 实例化对象：对象名称 = new 类名称() ; à 堆内存中开辟
  - |- 垃圾空间：一个堆内存空间不再有任何的栈内存所指向，那么将称为垃圾，等待被自动回收。

2、 封装性：private

- 在类中可以使用 private 声明或方法，声明之后，这些属性和方法是无法被类的外部所访问的。

- 类中的所有属性都必须封装，被封装的属性只能通过 setter 及 getter 设置和取得。

3、 构造方法：

- 主要的功能是为类中的属性进行初始化。
- 方法名称与类名称相同，且无返回值声明。
- 当使用关键字 new 进行对象实例化的时候调用。
- 当一个类中没有明确的声明的一个构造方法的话，将默认生成一个无参的什么都不做的构造方法，以保证一个类中至少存在一个构造方法。
- 构造方法也可以进行重载。

4、 String 类

- String 有两种实例化方式：直接赋值、调用构造方法（new String(String oth)）
- String 中的比较有两种方式：==和 equals
  - |- ==：比较的是内存地址的值
  - |- equals()：比较的是两个字符串的内容
- 使用直接赋值只会开辟一个堆内存空间，而且如果以后有相同的内容将不再开辟，而指向同一个空间，但是如果使用了构造方法的话，则会开辟两个内存空间，其中有一个将称为垃圾
- 一个字符串就是 String 的一个匿名对象，是在堆内存中保存的。
- 字符串的内容一旦声明之后则不可改变，改变的只是内存地址的指向。
- String 类的方法：
  - |- 与字符相关：toCharArray()、charAt(int ind)、new String(char c[])、new String(char c[],int offset,int length)
  - |- 与字节相关：getBytes()、new String(byte b[])、new String(byte b[],int offset,int length)
  - |- 拆份：public String[] split(String regex)
  - |- 替换：public String replaceAll(String org,String newc)、public String replaceFirst(String org,String newc)
  - |- 截取：public String substring(int offset,int length) 、public String substring(int offset)
  - |- 其他操作：length()、toUpperCase()、toLowerCase()、trim()、equalsIgnoreCase()、

5、 对象数组：对象数组采用动态初始化的时候，里面的所有内容都是 null。

2.2、本次预计讲解的知识点

1、 this 关键字的使用
2、 引用传递的深入掌握
3、 static 关键字的作用
4、 内部类

3、具体内容

3.1、this 关键字（重点）

this 这个关键字个人认为在自学中是最痛苦的，因为市面没有一本书可以把 this 讲明白的，在 Java 中 this 可以作为以下的操作使用：

访问属性、调用方法（包含了构造）、**表示当前对象**。

3.1.1、访问属性

现在，假设有如下的一道程序：

```
class Person {
    private String name ;
    private int age ;
    public Person(String n,int a){
        name = n ;
        age = a ;
    }
    public String getInfo(){
        return "姓名：" + name + "，年龄：" + age ;
    }
    // 编写 setter、getter
};
public class ThisDemo01{
    public static void main(String args[]){
        Person per = new Person("张三",20) ;
        System.out.println(per.getInfo()) ;
    }
};
```

以上程序的功能明确而且简单，但是程序有个问题了，从构造方法上可以发现，要传递姓名和年龄，但是从这两个参数的名称定义上根本就不能体现出来。

```
    public Person(String name,int age){
        name = name ;
        age = age ;
    }
```

以上的操作程序中的 name 和 age 两个参数实际上与类中的属性没有任何的关系，而只是在构造方法中的参数间的互操作而已，那么如果此时非要明确的强调出要把构造方法中的 name 或 age 参数给类中的 name 和 age 属性的话，就需要通过 this 来表示出本类的属性。

```
class Person {
    private String name ;
    private int age ;
    public Person(String name,int age){
        this.name = name ;
        this.age = age ;
    }
    public String getInfo(){
        return "姓名：" + name + "，年龄：" + age ;
    }
    // 编写 setter、getter
};
public class ThisDemo01{
```

```
    public static void main(String args[]){
        Person per = new Person("张三",20) ;
        System.out.println(per.getInfo()) ;
    }
};
```

所以，以后对于 setter 和 getter 方法最好按照如下的方式编写：

```
class Person {
    private String name ;
    private int age ;
    public Person(String name,int age){
        this.name = name ;
        this.age = age ;
    }
    public String getInfo(){
        return "姓名：" + name + "，年龄：" + age ;
    }
    public void setName(String name){
        this.name = name ;
    }
    public void setAge(int age){
        this.age = age ;
    }
    public String getName(){
        return this.name ;
    }
    public int getAge(){
        return this.age ;
    }
};
```

3.1.2、调用方法

早在之前就强调过，如果现在在一个类中调用了本类的方法的话，可以使用“this.方法()”，但是对于方法的调用，使用 this 也可以调用类中的构造方法，但是只局限于在一个构造方法中调用其他构造方法的形式。

```
class Person {
    private String name ;
    private int age ;
    public Person(){
        System.out.println("一个新的对象产生了！") ;
    }
    public Person(String name){
        this() ;            // 调用本类中的无参构造
        this.name = name ;
```

```
    }
    public Person(String name,int age){
        this(name) ;      // 调用有一个参数的构造
        this.age = age ;
    }
    public String getInfo(){
        return "姓名：" + name + "，年龄：" + age ;
    }
};
public class ThisDemo02{
    public static void main(String args[]){
        new Person("张三",20) ;
        new Person("张三") ;
        new Person() ;
    }
};
```

但是，如果现在使用 this 调用其他构造方法的话，有一点需要注意，即：**如果使用 this 调用本类中其他构造方法的话，则此语句必须放在构造方法的首行**。

而且，在使用 this 调用构造方法的时候一定要注意：“**一个类中如果有多个构造方法的话，则肯定保留有一个是不用 this 调用其他构造的情况，以作为出口”**。

```
class Person {
    private String name ;
    private int age ;
    public Person(){
        this("",0) ;
        System.out.println("一个新的对象产生了！") ;
    }
    public Person(String name){
        this() ;          // 调用本类中的无参构造
        this.name = name ;
    }
    public Person(String name,int age){
        this(name) ;      // 调用有一个参数的构造
        this.age = age ;
    }
    public String getInfo(){
        return "姓名：" + name + "，年龄：" + age ;
    }
};
public class ThisDemo03{
    public static void main(String args[]){
        new Person("张三",20) ;
        new Person("张三") ;
        new Person() ;
```

```
    }
};
```

而且当一个类中如果存在多个构造方法的时候，一定要按照以下的顺序进行：

- 将参数少的构造方法放在最前面，参数多的构造方法放在最后面。

3.1.3、表示当前

this 的最大用法实际上就只有一个：“表示当前对象”，包括以上的访问属性或者是调用构造等等，实际上都是此概念的体现。当前调用类中方法的对象称为当前对象。

```
class Person {
    public void fun(){
        System.out.println("当前对象：" + this) ;
    }
};
public class ThisDemo04 {
    public static void main(String args[]){
        Person p1 = new Person() ;
        Person p2 = new Person() ;
        System.out.println("p1 对象：" + p1) ;
        p1.fun() ;
        System.out.println("--------------------------") ;
        System.out.println("p2 对象：" + p2) ;
        p2.fun() ;
    }
};
```

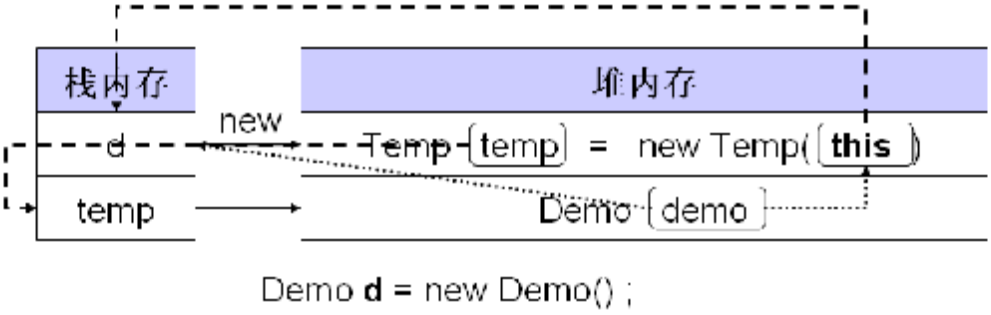
那么按照这种理解，则之前的“this.属性”实际上就意味着表示当前对象的属性。

范例：观察如下程序的输出结果 —— 本程序没有任何实际意义，只是为了加强 this 的理解。

```
class Demo {
    private Temp temp ;
    public Demo(){
        this.temp = new Temp(this) ;
        this.temp.fun() ;
    }
    public void print(){
        System.out.println("Hello World!!!") ;
    }
};
class Temp {
    private Demo demo ;
    public Temp(Demo demo){
        this.demo = demo ;
    }
    public void fun(){
```

```
        this.demo.print() ;
    }
};
public class ThisDemo05 {
    public static void main(String args[]){
        Demo d = new Demo() ;
    }
};
```

将以上的程序代码变为内存关系图。



3.1.4、习题

编写一个公司员工类

- 数据成员：员工号、姓名、薪水、部门
- 方法：
  - |- 利用构造方法完成设置信息：
    - A、单参，只传递员工号，则员工姓名：无名氏，薪水：0，部门：未定
    - B、双参，传递员工号，姓名，则员工薪水为 1000，部门：后勤
    - C、四参，传递员工号，姓名，部门，薪水
    - D、无参，则均为空值
  - |- 显示信息

```
class Emp {
    private int empno ;
    private String ename ;
    private float sal ;
    private String dept ;
    public Emp(){}
    public Emp(int empno){
        this(empno,"无名氏","未定",0.0f) ;
    }
    public Emp(int empno,String ename){
        this(empno,ename,"后勤",1000.0f) ;
    }
    public Emp(int empno,String ename,String dept,float sal){
        this.empno = empno ;
```

```
        this.ename = ename ;
        this.dept = dept ;
        this.sal = sal ;
    }
    public String getInfo(){
        return      "雇员编号：" + this.empno + "\n" +
                    "雇员姓名：" + this.ename + "\n" +
                    "部门：" + this.dept + "\n" +
                    "工资：" + this.sal ;
    }
};
public class ThisDemo06 {
    public static void main(String args[]){
        System.out.println(new Emp(1009,"张三","测试",3000.0f).getInfo()) ;
    }
};
```

3.2、引用传递（重点）

引用传递肯定是 Java 的核心问题，引用传递的操作的核心就是内存地址的传递，那么下面通过几道题目以及一些实际的问题，来分析一下引用传递的作用。

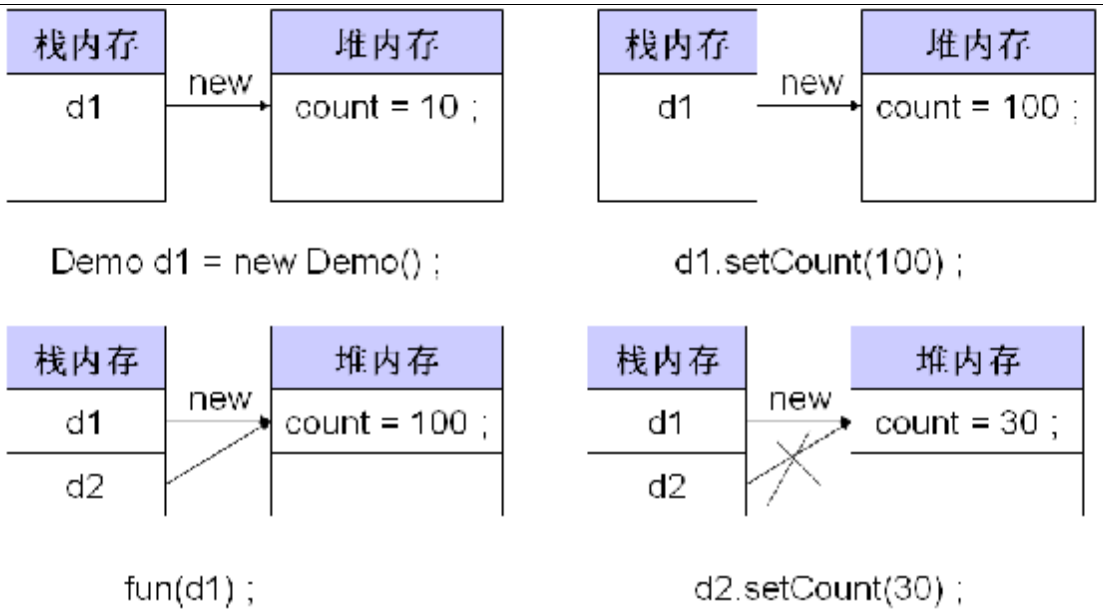
3.2.1、三道引用范例

下面通过三道程序来回顾一下之前学习过的关于引用传递的各个概念。

范例：程序一

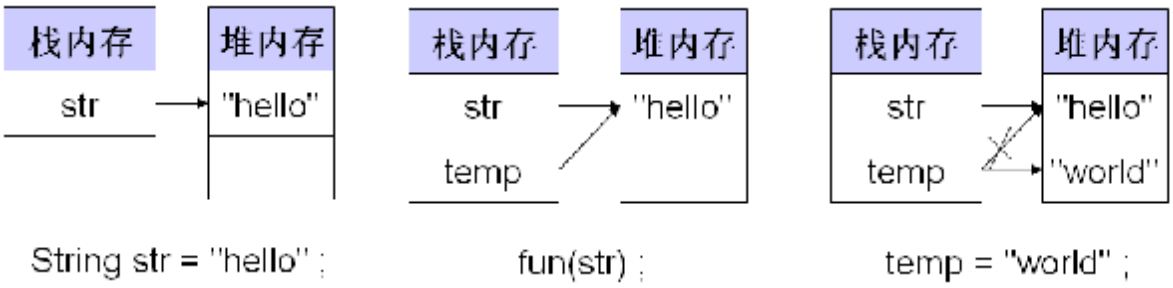
```
class Demo{
    private int count = 10 ;
    public void setCount(int count){
        this.count = count ;
    }
    public int getCount(){
        return this.count ;
    }
};
public class RefDemo01 {
    public static void main(String args[]){
        Demo d1 = new Demo() ;
        d1.setCount(100) ;
        fun(d1) ;
        System.out.println(d1.getCount()) ;
    }
    public static void fun(Demo d2){
```

```
        d2.setCount(30) ;
    }
};
```



范例：第二道程序

```
public class RefDemo02 {
    public static void main(String args[]){
        String str = "hello" ;
        fun(str) ;
        System.out.println(str) ;
    }
    public static void fun(String temp){
        temp = "world" ;
    }
};
```



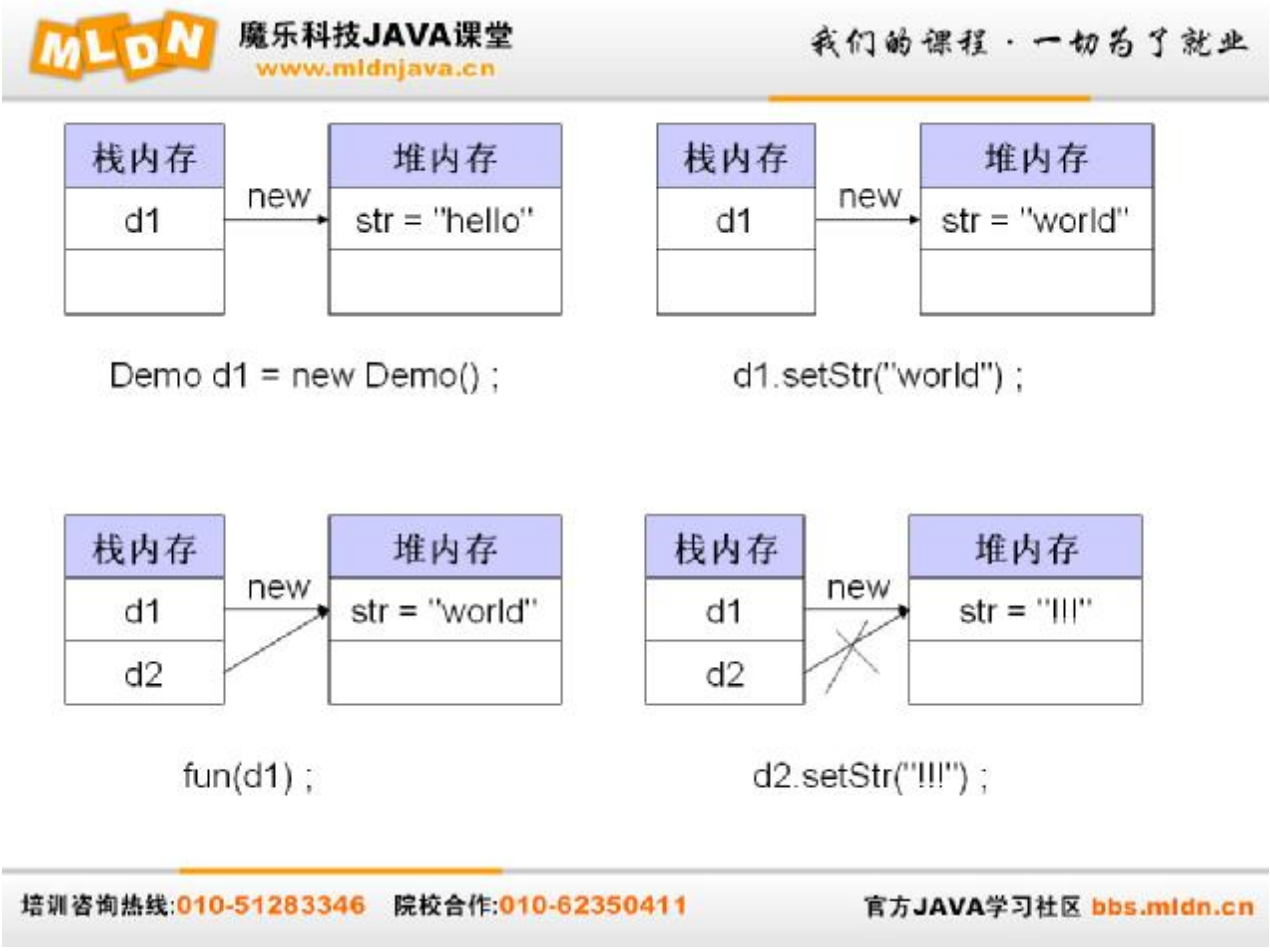
字符串的内容无法改变，改变的只是内存地址的指向。

范例：第三道程序

```
class Demo{
    private String str = "hello" ;
    public void setStr(String str){
        this.str = str ;
    }
    public String getStr(){
```

```
            return this.str ;
        }
};
public class RefDemo03 {
        public static void main(String args[]){
            Demo d1 = new Demo() ;
            d1.setStr("world") ;
            fun(d1) ;
            System.out.println(d1.getStr()) ;
        }
        public static void fun(Demo d2){
            d2.setStr("!!!") ;
        }
};
```

实际上，本程序和第一道程序是完全一样的，下面观察内存关系图



3.2.2、实际问题（理解）

面向对象实际上可以对现实的世界进行抽象，也就是说，现实世界的一切物质都可以通过面向对象的关系来表示出来，例如：各个不同的物种或者是不同的产品，一切的一切都可以进行抽象。

那么，下面通过几个实际的问题来观察。

一个人有一部汽车，此时肯定可以进行面向对象分析，很明显现在应该有两种类。

```
class Person {
	private String name ;
	private Car car ; // 表示一个人可以有一部车
	public Person(String name){
		this.name = name ;
	}
	public String getName(){
		return this.name ;
	}
	public void setCar(Car car){
		this.car = car ;
	}
	public Car getCar(){
		return this.car ;
	}
};
class Car {
	private String brand ;
	private Person person ;// 表示一部车属于一个人
	public Car(String brand){
		this.brand = brand ;
	}
	public String getBrand(){
		return this.brand ;
	}
	public void setPerson(Person person){
		this.person = person ;
	}
	public Person getPerson(){
		return this.person ;
	}
};
public class RefDemo04 {
	public static void main(String args[]){
		Person per = new Person("张三") ;
		Car car = new Car("BENZ") ;
		per.setCar(car) ;
		car.setPerson(per) ;
		System.out.println(per.getCar().getBrand()) ;
		System.out.println(car.getPerson().getName()) ;
	}
};
```

一个人可能还有后代，后代可能还有车。

```
class Person {
	private String name ;
	private Car car ; // 表示一个人可以有一部车
	private Person child ;
	public Person(String name){
		this.name = name ;
	}
	public String getName(){
		return this.name ;
	}
	public void setCar(Car car){
		this.car = car ;
	}
	public Car getCar(){
		return this.car ;
	}
	public void setChild(Person child){
		this.child = child ;
	}
	public Person getChild(){
		return this.child ;
	}
};
class Car {
	private String brand ;
	private Person person ;// 表示一部车属于一个人
	public Car(String brand){
		this.brand = brand ;
	}
	public String getBrand(){
		return this.brand ;
	}
	public void setPerson(Person person){
		this.person = person ;
	}
	public Person getPerson(){
		return this.person ;
	}
};
public class RefDemo05 {
	public static void main(String args[]){
		Person per = new Person("张三") ;
		Person chd = new Person("张四") ;
		Car car = new Car("BENZ") ;
```

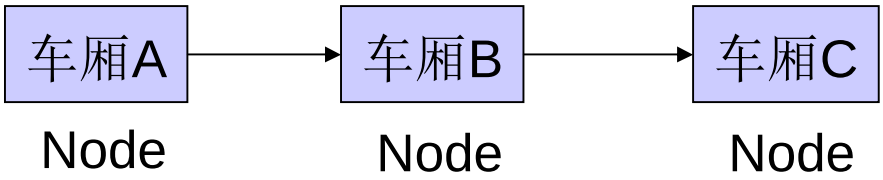
```
        Car ppc = new Car("碰碰车") ;
        per.setCar(car) ;
        car.setPerson(per) ;
        per.setChild(chd) ;
        chd.setCar(ppc) ;
        System.out.println(per.getCar().getBrand()) ;
        System.out.println(car.getPerson().getName()) ;
        System.out.println(per.getChild().getCar().getBrand()) ;
    }
};
```

例如：在 emp 表中的字段（empno、ename、mgr、job、sal），能否通过类的关系表示出来呢？

```
class Emp {
    private int empno ;
    private String ename ;
    private String job ;
    private float sal ;
    private Emp mgr ;
};
public class RefDemo06 {
    public static void main(String args[]){
    }
};
```

3.2.3、链表初步认识（了解）

下面要求通过程序表示出如下的一种关系：

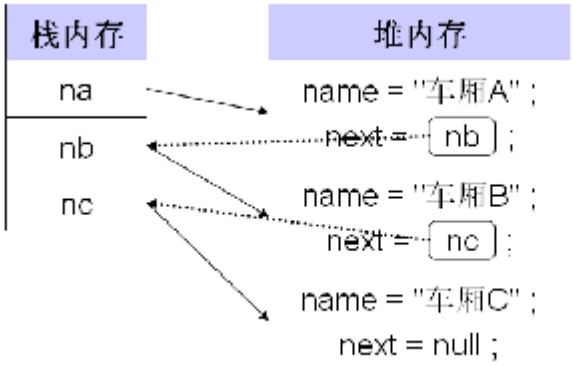


要想表示出这种节点的关系，则肯定首先要定义出一个节点的操作类。

```
class Node {
    private String name ;  // 保留节点名称
    private Node next ;          // 保存下一个节点的引用
    public Node(String name){
        this.name = name ;
    }
    public String getName(){
        return this.name ;
    }
```

```
    public void setNext(Node next){
        this.next = next ;
    }
    public Node getNext(){
        return this.next ;
    }
};
public class RefDemo07 {
    public static void main(String args[]){
        Node na = new Node("车厢 A") ;
        Node nb = new Node("车厢 B") ;
        Node nc = new Node("车厢 C") ;
        na.setNext(nb) ;
        nb.setNext(nc) ;
    }
};
```

下面通过内存关系图来表示出以上的关系。



但是，以上只是设置好了关系，那么该如何把内容全部输出呢？只能通过递归的方式完成。

```
public class RefDemo08 {
    public static void main(String args[]){
        Node na = new Node("车厢 A") ;
        Node nb = new Node("车厢 B") ;
        Node nc = new Node("车厢 C") ;
        na.setNext(nb) ;
        nb.setNext(nc) ;
        print(na) ;
    }
    public static void print(Node node){
        if(node != null){ // 避免空指向异常
            System.out.println(node.getName()) ;
            if(node.getNext() != null){
                print(node.getNext()) ;
            }
        }
```

```
    }
};
```

本程序就是在数据结构上所讲解的单向链表的基本实现原理，但是本程序的代码完全是靠手工的方式处理关系的，这样肯定不方便，能否编写一个代码，可以自动的完成增加和输出的功能呢？

要想完成此功能，肯定有以下几点必须考虑：

1、 必须有一个可以保存节点关系的类：Node

2、 有一个可以操作 Node 的类：Link

3、 但是在操作类之中需要注意一个问题，必须保留好根节点，第一个节点就是根节点

```
class Node {
    private String name ;  // 保留节点名称
    private Node next ;          // 保存下一个节点的引用
    public Node(String name){
        this.name = name ;
    }
    public String getName(){
        return this.name ;
    }
    public void setNext(Node next){
        this.next = next ;
    }
    public Node getNext(){
        return this.next ;
    }
    public void addNode(Node newNode){
        if(this.next == null){
            this.next = newNode ;
        } else {
            this.next.addNode(newNode) ;    // 当前节点的下一个继续往下判断
        }
    }
    public void printNode(){
        System.out.println(this.name) ;
        if(this.next != null){
            this.next.printNode() ;
        }
    }
};
class Link {      // 表示的是一个节点的操作类
    private Node root ;          // 表示根节点
    public void add(String name){     // 增加新节点
        Node newNode = new Node(name) ;    // 新节点
        if(this.root == null){   // 现在没有根节点
            this.root = newNode ; // 第一个节点是根节点
        } else {     // 应该排在最后
```

```
                this.root.addNode(newNode) ;     // 向后面排队
            }
        }
        public void print(){
            this.root.printNode() ;
        }
};
public class RefDemo09 {
        public static void main(String args[]){
            Link link = new Link() ;
            link.add("车厢 A") ;
            link.add("车厢 B") ;
            link.add("车厢 C") ;
            link.add("车厢 D") ;
            link.add("车厢 E") ;
            link.print() ;
        }
};
```

这就是一个链表的基本数据模型。

3.3、对象比较（重点）

对象比较的概念本身并不难理解，就是判断两个对象的内容是否相等。

对象比较的核心就是判断两个对象中的属性是否完全相等。

范例：对象比较的第一种情况

```
class Person {
        private String name ;
        private int age ;
        public Person(String name,int age){
            this.name = name ;
            this.age = age ;
        }
        public String getName(){
            return this.name ;
        }
        public int getAge(){
            return this.age ;
        }
};
public class CompareDemo01 {
        public static void main(String args[]){
            Person per1 = new Person("张三",30) ;
            Person per2 = new Person("张三",30) ;
```

```
        if(per1.getName().equals(per2.getName()) && per1.getAge() == per2.getAge()){
            System.out.println("是同一个人！") ;
        } else {
            System.out.println("不是同一个人！") ;
        }
    }
};
```

现在的功能实现了，但是存在问题，对于对象的比较操作而言，如果按照以上的做法，由主方法进行比较，实际上就相当于增加了一个第三方，由第三方帮你进行比较，肯定不合适，所以最好的做法是在 Person 类中本身就已经具备了一个这样的功能。

```
class Person {
    private String name ;
    private int age ;
    public Person(String name,int age){
        this.name = name ;
        this.age = age ;
    }
    public boolean compare(Person person){
        if(this == person){     // 地址相等了
            return true ;
        }
        if(this.name.equals(person.name) && this.age==person.age){
            return true ;
        } else {
            return false ;
        }
    }
    public String getName(){
        return this.name ;
    }
    public int getAge(){
        return this.age ;
    }
};
public class CompareDemo02 {
    public static void main(String args[]){
        Person per1 = new Person("张三",30) ;
        Person per2 = new Person("张三",30) ;
        if(per1.compare(per2)){
            System.out.println("是同一个人！") ;
        } else {
            System.out.println("不是同一个人！") ;
        }
    }
```

```
};
```

对象比较的操作代码，在以后的开发中将非常的重要，一定要熟练编写。

3.4、static 关键字（重点）

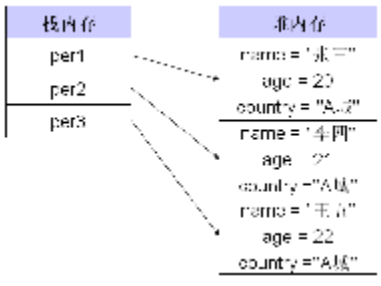
static 关键字在类中可以声明属性或方法。声明的属性将称为全局属性，声明的方法将成为类方法。

3.4.1、声明属性

在讲解之前先来观察以下的一道程序。

```
class Person {
    private String name ;
    private int age ;
    String country = "A 城" ;
    public Person(String name,int age){
        this.name = name ;
        this.age = age ;
    }
    public String getInfo(){
        return "姓名：" + this.name + "，年龄：" + this.age + "，城市：" + country ;
    }
};
public class StaticDemo01{
    public static void main(String args[]){
        Person per1 = new Person("张三",20) ;
        Person per2 = new Person("李四",21) ;
        Person per3 = new Person("王五",22) ;
        System.out.println(per1.getInfo()) ;
        System.out.println(per2.getInfo()) ;
        System.out.println(per3.getInfo()) ;
    }
};
```

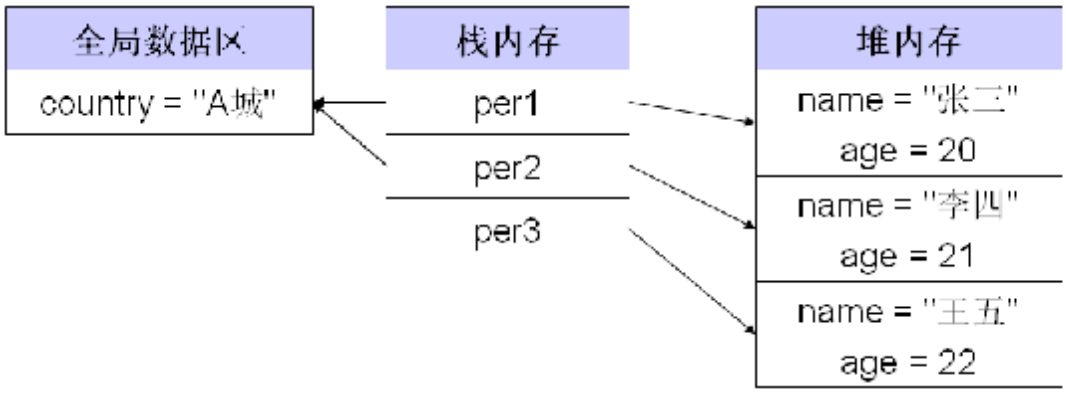
但是以上的程序存在了一个问题，现在的所有人的城市都是在 A 城，但是后来这个城市的名字修改成了 X 城，那么如果现在已经产生了 500000 个 Person 对象，肯定现在要修改 50000 次对象中的 country 属性，因为按照内存的分配来讲，每一个对象都单独占着各自的 country 属性。



现在以上的设计根本就无法满足，因为对于 country 属性而言，肯定是所有对象都是一样的，而且如果按照以上的设计，每个对象占用每个对象自己的 country 属性的话，则肯定会造成内存空间的浪费，那么就可以将 country 属性设置成一个公共属性，那么公共属性就可以使用 static 进行操作。

```
class Person {
    private String name ;
    private int age ;
    static String country = "A 城" ;
    public Person(String name,int age){
        this.name = name ;
        this.age = age ;
    }
    public String getInfo(){
        return "姓名：" + this.name + "，年龄：" + this.age + "，城市：" + country ;
    }
};
public class StaticDemo02{
    public static void main(String args[]){
        Person per1 = new Person("张三",20) ;
        Person per2 = new Person("李四",21) ;
        Person per3 = new Person("王五",22) ;
        System.out.println(per1.getInfo()) ;
        System.out.println(per2.getInfo()) ;
        System.out.println(per3.getInfo()) ;
        System.out.println("----------------------") ;
        per1.country = "B 城" ;
        System.out.println(per1.getInfo()) ;
        System.out.println(per2.getInfo()) ;
        System.out.println(per3.getInfo()) ;
    }
};
```

现在发现修改了一个对象中的 country 属性之后，所有对象的 country 属性都发生了变化，那么 country 属性肯定就成为了公共属性，此时的内存图如下：



这个时候的 country 属性就成为了一个公共属性，所有的对象可以同时拥有，但是一般而言这些公共属性虽然可以使用普通的对象调用，可一般不这样操作，而是通过所有对象的最大级 —— 类，完成调用。

```
Person.country = "B 城" ;
```

由于全局属性拥有可以通过类名称直接访问的特点，所以这种属性又称为**类属性**。

3.4.2、声明方法

对于属性而言，不管是何种属性都要考虑封装，对于全局属性也不例外，那么既然全局属性可以通过类名称访问，那么也可以使用 static 定义一个“类方法”，很明显此方法也可以通过类名称直接访问。

```
class Person {
    private String name ;
    private int age ;
    private static String country = "A 城" ;
    public Person(String name,int age){
        this.name = name ;
        this.age = age ;
    }
    public static void setCountry(String c){
        country = c ;
    }
    public String getInfo(){
        return "姓名：" + this.name + "，年龄：" + this.age + "，城市：" + country ;
    }
};
public class StaticDemo03 {
    public static void main(String args[]){
        Person per1 = new Person("张三",20) ;
        Person per2 = new Person("李四",21) ;
        Person per3 = new Person("王五",22) ;
        System.out.println(per1.getInfo()) ;
        System.out.println(per2.getInfo()) ;
        System.out.println(per3.getInfo()) ;
        System.out.println("----------------------") ;
        Person.setCountry("B 城") ;
        System.out.println(per1.getInfo()) ;
        System.out.println(per2.getInfo()) ;
        System.out.println(per3.getInfo()) ;
    }
};
```

3.4.3、static 的使用注意

static 本身声明方法或声明属性，但是在方法的访问上本身也存在着问题。

1、 使用 static 声明的方法，不能访问非 static 的操作（属性或方法）

2、 非 static 声明的方法，可以访问 static 声明的属性或方法

```
class Person {
	static String country = "A 城" ;
	private String name ;
	public static void setCountry(String c){
		print() ;
		System.out.println(name) ;
	}
	public void print(){	// 非 static 方法
	}
};
```

```
class Person {
	static String country = "A 城" ;
	private String name ;
	public static void setCountry(String c){
	}
	public void print(){	// 非 static 方法
		setCountry("") ;
		System.out.println(country) ;
	}
};
```

那么，为什么呢？

1、 如果说现在一个类中的属性或方法都是非 static 类型的，肯定是要有实例化对象才可以调用。

2、 static 声明的属性或方法可以通过类名称访问，可以在没有实例化对象的情况下调用。

当一个实例化对象产生之后，可以调用所有的非 static 的操作，那么肯定也就可以调用所有的 static 操作。

回顾：在之前讲解方法的时候给出一个如果一个方法由主方法调用的格式：

```
public static 返回值类型 方法名称(参数列表){
	[return 返回值 ;]
}
```

下面通过代码来观察：

```
public class StaticDemo05{
	public static void main(String args[]){
		new StaticDemo05().printInfo() ;
	}
	public void printInfo(){
		System.out.println("Hello World!!!") ;
	}
};
```

3.4.4、static 的应用（理解）

既然 static 的属性是所有对象所共同拥有的，那么就可以利用此特点完成一个自动命名的功能。

```
class Book {
```

```
    private String name ;
    private static int count = 0 ;
    public Book(){
        this.name = "无名字 - " + ++count ;
    }
    public Book(String name){
        this.name = name ;
    }
    public String getName(){
        return this.name ;
    }
};
public class StaticDemo06 {
    public static void main(String args[]){
        System.out.println(new Book("JAVA 基础").getName()) ;
        System.out.println(new Book().getName()) ;
        System.out.println(new Book().getName()) ;
        System.out.println(new Book().getName()) ;
        System.out.println(new Book("Oracle").getName()) ;
    }
};
```

以后在学习 java 其他内容的时候将使用以上的概念。

3.4.5、主方法的组成

主方法：是所有调用的起点，定义如下：

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

实际上每一个修饰符或参数都是有意义的：

- public：表示外部都可以访问
- static：方法可以由类直接调用
- void：程序的起点，一旦开始了就没回头路
- main：系统规定好的默认的方法名称
- String args[]：是一个字符串数组，用于接收参数

在主方法中，可以在一个程序运行时输入多个参数，每个参数之间使用空格分隔，所有的参数都靠 String args[]进行接收。

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

但是，如果现在要想输入的参数中本身就包含了空格呢？通过双引号完成。

```
java Hello "hello world" "hello mldn"
```

3.5、习题（重点）

现在要求完成一个用户的登陆程序，通过初始化参数，输入登陆的用户名和密码

例如：java LoginDemo 用户名 密码

如果用户名是 hello，密码是 mldn 的话，则表示登陆成功，显示登陆欢迎的信息，如果登陆失败，则显示登陆失败的信息。

自己思考如何做，先完成基本功能，之后再进行结构的修改。

以后对于程序的开发，按照以下的步骤思考：

1、 先完成功能；

2、 再优化结构或者加入验证。

核心功能就是输入内容之后验证。

```
public class LoginDemo {
    public static void main(String args[]){
        String name = args[0] ;
        String password = args[1] ;
        if("hello".equals(name) && "mldn".equals(password)){
            System.out.println("用户登陆成功！") ;
        } else {
            System.out.println("用户登陆失败！") ;
        }
    }
};
```

以上就属于基本功能，但是这个时候肯定要分析以上代码的问题了。

1、 数组越界

```
public class LoginDemo {
    public static void main(String args[]){
        if(args.length != 2){    // 输入参数不足两个
            System.out.println("ERROR...") ;
            System.exit(1) ; // 直接退出系统
        }
        String name = args[0] ;
        String password = args[1] ;
        if("hello".equals(name) && "mldn".equals(password)){
            System.out.println("用户登陆成功！") ;
        } else {
            System.out.println("用户登陆失败！") ;
        }
    }
};
```

主方法属于一切操作的起点。所以主方法将作为一个程序的客户端存在，那么客户端的操作代码越少越好。

下面按照这个思路，肯定要对程序的功能结构进行细分。

```
class Login {
    private String name ;
```

```
    private String password ;
    public Login(String name,String password){
        this.name = name ;
        this.password = password ;
    }
    public boolean isLogin(){
        if("hello".equals(this.name) && "mldn".equals(this.password)){
            return true ;
        } else {
            return false ;
        }
    }
};
class LoginOperate {   // 登陆的具体操作
    private String args [] ; // 接收所有的登陆信息
    public LoginOperate(String args[]){
        this.args = args ; // 接收数组
        if(this.isExit()){ // 符合退出条件
            System.out.println("ERROR...") ;
            System.exit(1) ; // 系统退出
        }
    }
    public boolean isExit(){
        if(this.args.length != 2){
            return true ;       // 长度不够，退出
        } else {
            return false ;
        }
    }
    public String login(){
        Login login = new Login(args[0],args[1]) ;
        if(login.isLogin()){
            return "用户登陆成功！" ;
        } else {
            return "用户登陆失败！" ;
        }
    }
};
public class LoginDemo {
    public static void main(String args[]){
        System.out.println(new LoginOperate(args).login()) ;
    }
};
```

在进行程序类的结构划分上，就一个原则：一个类只完成一个类的功能。

3.6、代码块（理解）

在程序中使用“{}”括起来的一段代码就称为代码块，根据代码块出现的位置或声明的关键字的不同，分为四种：

- 普通代码块：在一个方法中定义的代码块，称为普通代码块
- 构造块：在一个类中定义的代码块
- 静态块：使用 static 关键字声明的代码块
- 同步代码块：留到线程再讲

从实际的开发来看，代码块的使用并不多。

3.6.1、普通代码块

```
public class CodeDemo01 {
    public static void main(String args[]) {
        {    // 普通代码块
            int x = 10 ;
            System.out.println("x = " + x) ;
        }
        int x = 100 ;
        System.out.println("x = " + x) ;
    }
};
```

3.6.2、构造块

构造块是定义在一个类中的代码块。

```
class Demo {
    {
        System.out.println("构造块。。。") ;
    }
    public Demo(){
        System.out.println("构造方法。。。") ;
    }
};
public class CodeDemo02 {
    public static void main(String args[]) {
        new Demo() ;
        new Demo() ;
        new Demo() ;
    }
};
```

构造块会优先于构造方法执行，而且每当一个新的实例化对象产生时，都会调用构造块，会调用多次。

3.6.3、静态块

使用 static 关键字定义的代码块，而且静态块是定义在类中的。

```
class Demo {
	static{
		System.out.println("静态块。。。") ;
	}
	{
		System.out.println("构造块。。。") ;
	}
	public Demo(){
		System.out.println("构造方法。。。") ;
	}
};
public class CodeDemo03 {
	static{
		System.out.println("主方法中的静态块") ;
	}
	public static void main(String args[]) {
		System.out.println("----------------------------") ;
		new Demo() ;
		new Demo() ;
		new Demo() ;
	}
};
```

在主类中定义的静态块将优先于主方法执行，而且可以发现静态块优先于构造块执行，而且只执行一次。

但是，静态块本身也非常的有意思，可以使用静态块“替代”掉主方法。

```
public class CodeDemo04 {
	static {
		System.out.println("Hello World!!!") ;
		System.exit(1) ;
	}
};
```

此代码没有任何的实际作用，只能纯粹的单单的作为娱乐使用。

3.7、内部类（重点）

内部类随着 JAVA 技术的发展，慢慢的已经形成了自己的独特应用。

3.7.1、内部类的基本语法

所谓的内部类就是指一个类的内部还包含了另外的一个操作类，被包含的类称为内部类，包含的类称为外部类。

```
class Outer {      // 定义外部类
    private String info = "Hello" ;
    class Inner {      // 定义内部类
        public void print(){
            System.out.println(info) ;  // 输出 info 属性
        }
    };
    public void fun(){
        new Inner().print() ;
    }
};
public class InnerDemo01 {
    public static void main(String args[]){
        new Outer().fun() ;
    }
};
```

从以上的代码中可以观察到内部类的特点：

1、 缺点：破坏了一个程序的标准结构；

2、 优点：可以方便的访问外部类中的私有成员。

如果要想观察出内部类优点的话，则需要将以上的 Outer 和 Inner 类拆分成两个独立的类。

```
class Outer {      // 定义外部类
    private String info = "Hello" ;
    public void fun(){
        new Inner(this).print() ;
    }
    public String getInfo(){
        return this.info ;
    }
};
class Inner {      // 定义内部类
    private Outer out ;
    public Inner(Outer out){
        this.out = out ;
    }
    public void print(){
        System.out.println(out.getInfo()) ;// 输出 info 属性
    }
};
public class InnerDemo02 {
    public static void main(String args[]){
        new Outer().fun() ;
    }
};
```

此时，现在的内部类是通过外部类的 fun()方法进行对象实例化的，有没有可能在一个类的外部来实例化内部类的对

象呢？

观察内部类的*.class 文件的格式：Outer$Inner.class，其中程序中的“$”符号要替换成“.”，也就是说现在内部类的类名称：外部类.内部类。

```
class Outer {      // 定义外部类
    private String info = "Hello" ;
    class Inner {      // 定义内部类
        public void print(){
            System.out.println(info) ;     // 输出 info 属性
        }
    };
};
public class InnerDemo03 {
    public static void main(String args[]){
        Outer.Inner in = null ;                 // 声明内部类的对象
        in = new Outer().new Inner() ;
        in.print() ;
    }
};
```

内部类如果要被外部所调用的话，则一定要先产生外部类的实例化对象，之后再产生内部类的实例化对象。

3.7.2、使用 static 声明内部类

在内部类的定义中，也可以使用 static 关键字完成操作，一旦使用 static 声明了一个内部类的话，则此类将成为外部类，且只能访问外部类中的 static 成员。

```
class Outer {      // 定义外部类
    private static String info = "Hello" ;
    static class Inner {      // 定义内部类，是 static 变为外部类
        public void print(){
            System.out.println(info) ;     // 输出 info 属性
        }
    };
};
public class InnerDemo04 {
    public static void main(String args[]){
        // Outer.Inner in = new Outer().new Inner() ;
        Outer.Inner in = new Outer.Inner() ;
        in.print() ;
    }
};
```

3.7.3、在方法中声明内部类（重点）

理论上而言，一个内部类可以在任意的位置上声明，例如：一个循环语句中，或者在一个方法之中，从开发来看，

在方法中声明内部类的操作出现的是最多的。

```
class Outer {
    private String info = "hello" ;
    public void fun(){
        class Inner {      // 方法中声明内部类
            public void print(){
                System.out.println(info) ;
            }
        };
        new Inner().print() ;
    }
};
public class InnerDemo05 {
    public static void main(String args[]){
        new Outer().fun() ;
    }
};
```

此时，可以发现，一个在方法中定义的内部类，依然可以访问外部类中的属性，但是对于方法的参数，这个内部类是无法直接访问的，如果要访问，则在参数前面必须使用 final 关键字进行声明。

```
class Outer {
    private String info = "hello" ;
    public void fun(final int x){
        final int y = 100 ;
        class Inner {      // 方法中声明内部类
            public void print(){
                System.out.println(info) ;
                System.out.println("x = " + x) ;
                System.out.println("y = " + y) ;
            }
        };
        new Inner().print() ;
    }
};
public class InnerDemo06 {
    public static void main(String args[]){
        new Outer().fun(30) ;
    }
};
```

4、总结

1、 this 关键字

- this 关键字可以调用本类中的属性

- 可以调用本类中的方法，但是如果要调用构造方法的话，则一定要放在构造方法的首行，且至少保留一个构造方法中没有使用 this 调用。
- this 表示当前对象：当前操作类中方法的对象就是当前对象。

2、 引用传递的问题，并且可以使用引用表示出一些关系。

3、 链表是一项比较综合的运动，基本上融合了类的设计、引用传递、构造方法、封装型、this 关键字。

4、 对象比较的时候实际上是将两个对象的每个属性进行比较，是一个类的内部自己所提供的功能。

5、 static 关键字

- static 关键字声明的属性称为全局属性，为所有对象所共同拥有，可以直接通过类名称进行访问。
- static 声明的方法可以直接由类名称调用，但是 static 的方法不能调用非 static 的操作，而非 static 的方法可以调用所有的 static 操作。
- 主方法的组成。

6、 代码块可以不会，只需要了解形式即可。

7、 内部类的结构及各个操作必须掌握

- 使用 static 声明的内部类就是外部类
- 内部类可以在方法中声明，但是此时，要想访问方法中的参数，参数前必须加上 final 关键字

5、预习任务

继承、super、this、覆写、final 关键字、接口和抽象类，**对象多态性**

6、作业

将链表的作业尽可能完成，不完成也至少把之前的增加和输出搞明白，如果可以的话增加以下的功能：

1、 删除节点

2、 查找节点是否存在

如果可以，适当的使用内部类完成看。