2、具体内容

this可以算是Java里面比较复杂的关键字，因为this的使用形式上决定了它的灵活性，在程序里面，使用this可以实现以下三类结构的描述：

- 当前类中的属性：this.属性
- 当前类中的方法(构造方法、普通方法)：this.方法名()、this()
- 描述当前对象；

■使用this调用当前类中属性

通过现在的分析可以发现，利用构造方法或者是setter方法都可以进行类中的属性的赋值，但是在进行赋值的时候，之前采用的是如下的定义形式：

```
class Person    //定义一个类
{
        private String name;        //人员的姓名
        private int age;            //人的年龄
        public Person(String n,int a){
                name = n;
                age = a;
        }
        public void tell(){
                System.out.println("姓名："+name+"年龄："+age);
        }
        //getter、setter略
}
public class JavaDemo{//主类
        public static void main(String[] args) {
        Person per = new Person("王五",20);
        per.tell();
        }
}
```

但是这个时候在构造方法定义的过程之中会发现有一点点问题：

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

这个问题出现在参数名称上，可以发现，此时构造方法中两个参数的目的是为了类中的name或age属性初始化，但是现在发现此时的代码n和a参数名称不好。

如果说现在将构造方法中的参数名称修改为name、age，则发现无法进行属性的正确设置：

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

在Java程序之中“{}”是作为一个结构体的边界符，那么在程序里面当进行变量（参数、属性都称为变量）使用的时候都会以“{}”作为一个查找边界，所以按照就近取用的原则，此时的构造方法并没有能够访问类中的属性，所以此时为了明确的标记处类中属性与参数的区别，往往会在属性前追加一个"this"，表示本类属性。

```
class Person    //定义一个类
{
    private String name;      //人员的姓名
    private int age;          //人的年龄
    public Person(String name,int age){
        this.name = name;
        this.age = age;
    }
    public void tell(){
        System.out.println("姓名："+this.name+"年龄："+this.age);
    }
    //getter、setter略
}
```

在你以后锁编写的程序代码之中，只要是访问本类中属性的时候，请一定要加上“this”实现访问。

■使用this调用方法

除了调用属性之外，this也可以实现方法的调用，但是对于方法的调用就必须考虑构造与普通方法：

- 构造方法调用（this()）：使用关键字new实例化对象的时候才会调用构造方法；
- 普通方法调用（this.方法名称()）：实例化对象产生之后就可以调用普通方法；

范例：调用类中的普通方法

```
class Person    //定义一个类
{
    private String name;      //人员的姓名
    private int age;          //人的年龄
```

```
    public Person(String name,int age){
        this.setName(name);
        this.setAge(age);
        //setAget(age);//加与不加都表示本类方法
    }
    public void tell(){
        System.out.println("姓名："+this.name+"年龄："+this.age);
    }
    public void setName(String name){
        this.name = name;
    }
    public void setAge(int age){
        this.age = age;
    }
    public String getName(){
        return this.name;
    }
    public String getAge(){
        return this.age;
    }
}
public class JavaDemo{//主类
    public static void main(String[] args) {
     Person per = new Person("王五"，20);
    per.tell();
    }
}
```

除了普通的方法调用之外，还需要进行构造方法的调用，对于构造方法的调用，肯定是要放在构造方法中执行。现在假设说类中一共定义有三个构造方法，但是要求不管调用哪个构造方法，都执行一行输出语句“一个新的person类对象实例化“

传统做法实现：	利用this()构造调用优化：

class Person　//定义一个类
{
　　private String name;　　//人员的姓名
　　private int age;　　//人的年龄
　　public Person(){
　　System.out.println("一个新的person类对象实例化");
　　}
　　public Person(String name){
　　System.out.println("一个新的person类对象实例化");
　　this.name = name;
　　}
　　public Person(String name,int age){
　　System.out.println("一个新的person类对象实例化");
　　this.name = name;
　　this.age = age;
	class Person　//定义一个类
	{
	　　private String name;　　//人员的姓名
	　　private int age;　　//人的年龄
	　　public Person(){
	　　System.out.println("一个新的person类对象实例化");
	　　}
	　　public Person(String name){
	　　this();　　//调用本类无参构造
	　　this.name = name;
	　　}
	　　public Person(String name,int age){
	　　this(name);　//调用本类无参构造
	　　this.age = age;
	　　//setAget(age);//加与不加都表示本类方法
	　　}

```
        //setAget(age);//加与不加都表示本类方法
    }
    public void tell(){
        System.out.println("姓名："+this.name+"年龄："+this.age);
    }
    //setter getter 略
}
public class JavaDemo{//主类
    public static void main(String[] args) {
     Person per = new Person("王五");
    // Person per = new Person("王五",20);
    per.tell();
    }
}
```

```
    public void tell(){
        System.out.println("姓名："+this.name+"年龄："+this.age);
    }
    //setter getter 略
}
public class JavaDemo{//主类
    public static void main(String[] args) {
     Person per = new Person("王五");
    // Person per = new Person("王五",20);
    per.tell();
    }
}
```

如果要想评价一各代码的好坏：

- 代码结构可以重用，提供的是一个中间独立的支持；
- 我们的目标是：没有重复；

对于本类构造方法的互相调用需要注意以下几点重要问题：

- 构造方法必须在实例化新对象的时候调用，所以“this()”的语句只允许放在构造方法的首行。
- 构造方法相互调用时请保留有程序的出口，别形成死循环。

注意：构造方法能够调用普通方法，但是在普通方法不能够调用构造方法。

```
    public Person(){
        this("HAHA",11);
        System.out.println("一个新的person类对象实例化");
    }
    public Person(String name){
        this();      //调用本类无参构造
        this.name = name;
    }
    public Person(String name,int age){
        this(name);    //调用本类无参构造
        this.age = age;
        //setAget(age);//加与不加都表示本类方法
    }
```

此时的程序在进行编译的时候将会直接出现错误提示：告诉用户，你出现了构造方法的递归调用。

构造方法互调用案例：

现在要求定义一个描述有员工信息的程序类，该类中提供有：编号、姓名、部门、工资，在这个类之中提供有四个构造方法：

- 【无参构造】编号定义为1000，姓名定义为无名氏；

- 【单参构造】传递编号，姓名定义为“新员工”，部门定义为“未定”，工资为0；
- 【三参构造】传递编号、姓名、部门，工资为2500.0；
- 【四参构造】所有的属性全部进行传递。

范例：进行代码的初期实现

```
class Emp
{
	private long empno;	//员工编号
	private String ename;//员工姓名
	private String dept;//部门名称
	private double salary;//基本工资
	public Emp(){
		this.empno = 1000;
		this.ename = "无名氏";
	}
	public Emp(long empno){
		this.empno = empno;
		this.ename = "新员工";
		this.dept = "未定";
	}
	public Emp(long empno,String ename,String dept){
		this.empno = empno;
		this.ename = ename;
		this.dept = dept;
	}
	public Emp(long empno,String ename,String dept,double salary){
		this.empno = empno;
		this.ename = ename;
		this.dept = dept;
		this.salary = salary;
	}
	//setter、getter略
	public String getInfo(){
		return "雇员编号：“+this.empno+
				"雇员姓名：“+this.ename+
				"部门名称：“+this.dept+
				"基本工资：“+this.salay;
	}
}
public class JavaDemo{//主类
	public static void main(String[] args) {
		Emp emp = new Emp(7369L,"SMITH","CAIWUBU",6500.0);
		System.out.println(emp.getInfo());
	}
}
```

此时可以发现代码有重复，所以就可以对Emp类进行简化定义。

```
class Emp
{
```

```
	private long empno;	//员工编号
	private String ename;//员工姓名
	private String dept;//部门名称
	private double salary;//基本工资
	public Emp(){
		this.(1000,"无名氏",null,0.0);
	}
	public Emp(long empno){
		this.(empno,"新员工","未定",0.0);
	}
	public Emp(long empno,String ename,String dept){
		this.(empno,ename,dept,2500.0);
	}
	public Emp(long empno,String ename,String dept,double salary){
		this.empno = empno;
		this.ename = ename;
		this.dept = dept;
		this.salary = salary;
	}
	//setter、getter略
	public String getInfo(){
		return "雇员编号："+this.empno+
				"雇员姓名："+this.ename+
				"部门名称："+this.dept+
				"基本工资："+this.salay;
	}
}
public class JavaDemo{//主类
	public static void main(String[] args) {
		Emp emp = new Emp(7369L,"SMITH","CAIWUBU",6500.0);
		System.out.println(emp.getInfo());
	}
}
```

代码的任何位置上都可能有重复，所以消除重复的代码是先期学习之中最需要考虑的部分。

1、综合案例：简单Java类

2、具体内容

在以后进行项目的开发与设计的过程之中，简单Java类都将作为一个重要的组成部分存在，慢慢接触到正规的项目设计之后，简单Java类无处不在，并且有可能会产生一系列的变化。

所谓的简单java类指的是可以描述某一类信息的程序类，例如：描述一个人、描述一本书、描述一个雇员，并且在这个类之中并没有特别复杂的逻辑操作，只作为一种信息存储的媒介存在。

对于简单Java类而言，其核心的开发结构如下：

- 类名称一定要有意义，可以明确的描述某一类事物；
- 类之中的所有属性都必须使用private，同时封装后的属性必须要提供有setter、getter方法；注意：是简单java类才必须使用private
- 类之中可以提供有无数多个构造方法，但是必须要保留有无参构造方法；
- 类之中不允许出现任何的输出语句，所有内容的获取必须返回；
- 【非必须】可以提供有一个获取对象详细信息的方法，暂时将此方法名称定义为getInfo()；

范例：定义一个简单java类

```
class Dept      //类名称可以明确描述出某类事物
{
    private long deptno;
    private String dname;
    private String loc;
    public Dept(){}//必须提供无参构造方法
    public Dept(long deptno,String dname,String loc){
        this.deptno = deptno;
        this.dname = dname;
        this.loc = loc;
```

```
	}
	public void setDeptno(long deptno){
		this.deptno = deptno;
	}
	public void setDname(String dname){
		this.dname = dname;
	}
	public void setLoc(String loc){
		this.loc = loc;
	}
	public void getDname(){
		return this.Dname;
	}
	public void getDeptno(){
		return this.deptno;
	}
	public void getLoc(){
		return this.loc;
	}
	public void String getInfo(){
		return "[部门信息]部门编号："+this.deptno+
				" 部门名称："+this.Dname+
				" 部门位置："+this.loc;

	}

}
public class JavaDemo{//主类
	public static void main(String[] args) {
		Dept dept = new Dept(10,"technology","beijing");
		System.out.println(dept.getInfo());
	}
}
```

这种简单java类基本上就融合了所有的现在接触到的概念，例如：数据类型划分、类的定义、private封装、构造方法、方法定义、对象实例化。