《面向对象的特征》之封装

笔记本:	JAVA		
创建时间:	2018/7/19 16:40	**更新时间:**	2018/8/5 10:21
作者:	Debao Xu		

面向对象的三个特征：

1、封装 2、继承 3、多态

封装：
定义：即隐藏实现的具体细节，对外提供公共的访问方式（接口）
封装的Java代码体现：将属性通过关键字“private”进行私有化，对外提供对应的setXXX，getXXX方法来访问。
封装的好处：

1. 提高了安全性，不允许直接访问细节，并通过公共的方式来访问，可以实现可控。
2. 提高了复用性
3. 隔离了变化，防止程序出错。

下面举例说明封装的好处，将前面的代码复制下来，

```
class Student
{
	int num;
	String name;
	void eat()
	{
		System.out.println(num+" "+name+" "+"Eating!");
	}
}
class StudentDemo
{
	public static void main(String[] args)
	{
		Student s = new Student();// 创建对象

		s.num = 23;  //对象中的成员变量赋值
		s.name = "Rex"; // 同上
		s.eat();// 调用成员方法
	}
}
```

该程序的“bug”在于：在主方法中对成员变量num（年龄）进行赋值时，可以赋任何值（只要是int类型的，包括负值），这样在实际生活中就产生了问题（年龄不能为负）。
那么造成这一问题的原因是什么？原因就是在主方法中可以直接访问类中成员的属性，如果通过设置使不能直接对类中的属性进行更改，就可以解决这一问题。具体代码如下：

```
class Student
{
	private int num;//权限修饰符private使变量num只能在本类中使用，而不能再在主方法中直接赋值
	String name;
	public void setNum(int a )  //setNum函数间接对num赋值，该函数是public的，即在“类”外依然可以访问
	{
		if(a<0||a>130)
			throw new RuntimeException(a+", 数值是错误的！");//抛出异常
		else
			num = a;
	}
	public int getNum()
	{
		return num;
	}
	void eat()
	{
```

```
                System.out.println(num+" "+name+" "+"Eating!");
        }
}
class StudentDemo
{
        public static void main(String[] args)
        {
                Student s = new Student();// 创建对象
                s.setNum(23);  //通过set函数来间接对num赋值
                s.name = "Rex"; // 同上
                s.eat();// 调用成员方法
        }
}
```

该程序通过将变量num进行私有化（使用权限修饰符private），这样在主方法中就不能直接对变量num赋值，从而消除了因对num赋值而导致的错误。而要想对num进行操作就要使用函数setNum（该函数是public的，即在“类”的外部依然可以访问），通过setNum函数来间接对num操作。
这里是对类中的变量进行封装（私有化），另外，如果在类中的函数不需要对外提供访问，也可以将函数进行私有化。

构造函数

由来：在前面的程序中，类Student中的num和name在创建对象进行初始化时，其值num = 0，name = null。而这不符合实际情况，有时我们需要某些变量在初始化的过程中就有了相应的值，那么实现这一过程的方法就是通过“构造函数”来完成的。编写含有构造函数的程序如下，

```
class StuDemo
{
        private int num;
        private String name;
        StuDemo(String n,int m) //构造函数
        {
                name = n;
                num = m;
        }
        public void setNum(int a)
        {
                num = a;
        }
        public int getNum()
        {
                return num;
        }
        public void setName(String b)
        {
                name = b;
        }
        public String getName()
        {
                return name;
        }
        void Eat()
        {
                System.out.println(num+" "+name+" Eating!");
        }
}
public class Stu
{
        public static void main(String[] args)
        {
                StuDemo s = new StuDemo("DeBao",26);
                //s.setName("Rex");
                //s.setNum(23);
                s.Eat();
        }
}
```

```
运行结果：
26 DeBao Eating!
```

在上面的程序中并没有给name和num这两个变量赋值，但是在创建对象一开始，通过“构造函数”就已经给name和num赋了初值，name = “Debao”，num = 26，所以在调用了Eat()函数之后就能够输出结果。
“构造函数”的书写格式：

1. 函数名和类名相同
2. 没有返回值类型
3. 没有具体的返回值

“构造函数”和一般函数相比，有什么区别？
除了形式上的区别之外，在“运行上”，构造函数在创建对象时就执行了，而且只执行一次；而一般函数在创建对象之后如果有需要才被调用，而且可以被调用多次。
问题：在没有介绍“构造函数”之前，我们也创建了对象，如下：

```
Student s = new Student();// 创建对象
```

那么创建这个对象时，类中并没有类似于“Student()”这样的“构造函数”啊，那么创建时是如何初始化的呢？原因在于：1、一旦创建了一个类，那么他里面就会自动生成一个“默认的空参数的构造函数”，这个空参数的构造函数就被用于创建对象时的初始化。 2、一旦编写了自定义的“构造函数”，那么自动生成的空参数的构造函数就没有了。但是，如果想继续创建这样的对象 Student s = new Student(); 那么，我们可以自己编写一个带空参数的构造函数，然后通过这个函数创建对象。
关于构造函数的一些小细节：

1. 构造函数是有return语句的，return语句被用于结束该构造函数
2. 构造函数可以调用一般函数
3. 构造函数可以进行用关键字private进行“私有化”，让该构造函数只在本类中被使用，而且构造函数一旦被私有化，其他程序就无法创建该构造函数对应的对象，原因是在其他程序中创建出来的对象无法通过该构造函数进行初始化。

另外，构造函数与构造函数之间也是可以相互调用的，具体调用的方法是通过this关键字来解决，通过this带上参数列表的形式就可以调用其他构造函数，并且用于调用其他构造函数的this语句必须放在构造函数的第一行，因为初始化的动作要先执行。此外，构造函数是不能够被一般方法调用的，原因是：一般方法的的调用首先要创建对象，而创建对象又要初始化，初始化又要用到构造函数，这样就产生了矛盾。

```
StuDemo(String n,int m) //第一个构造函数
{
    this(28); //this关键字调用其他构造函数
    name = n;
}
StuDemo(int c) //第二个构造函数
{
    num = c;
}
StuDemo(String d) //第三个构造函数
{
    name = d;
}
```

上面有三个构造函数，其中在第一个构造函数中，通过this关键字来调用其他构造函数，根据this后面的参数列表（28），可以知道调用的是带有一个整型参数的构造函数（即第二个构造函数）。

关于this关键字（this代表的是对象，但仅仅是“代表”，而不是对象本身）：

1. 只要直接被对象调用的方法都持有this引用，即凡是访问了对象中的数据的方法都持有this引用。
2. 当成员变量和局部变量同名时，可以通过this关键字来区分。

举例说明this关键字在程序中的应用：

```
class Student
{
        private int age;
```

```
        private String name;
        public void setAge(int age)
        {
              this.age = age;
        }
        public int getAge()
        {
              return age;
        }
        public void setName(String name)
        {
              this.name = name;
        }
        public String getName()
        {
              return name;
        }
        public boolean equalsAge(Student a)
        {
              return a.age == this.age;
        }
}
public class StudentDemo
{
        public static void main(String[] args)
        {
              String name1;String name2;

              Student s1 = new Student();
              Student s2 = new Student();
              s1.setName("Rex");
              name1 = s1.getName();

              s2.setName("DeBao");
              name2 = s2.getName();

              s1.setAge(26);
              s2.setAge(26);

              boolean result = s1.equalsAge(s2);
              System.out.println(name1+"的年龄和"+name2+"一样吗?"+" "+result);
        }
}
```

运行结果:

```
Rex的年龄和DeBao一样吗? true
```