

1、

```
public class ClassAndObjectTest {
    public static void main(String[] args) {
        //创建类的实例，定义一个对象变量引用这一实例
        MyClass obj = new MyClass();
        //通过对象变量调用类的公有方法
        obj.myMethod("Hello");
        //给属性赋值
        obj.setValue(100);
        //输出属性的当前值
        System.out.println(obj.getValue());
        //直接访问对象公有字段
        obj.Information = "Information";
        //输出对象公有字段的当前值
        System.out.println(obj.Information);
    }
}

/**
 * 自定义Java类的示例
 */
class MyClass {
    // 公有字段
    public String Information = "";

    // 自定义公有Java实例方法
    public void myMethod(String argu) {
        System.out.println(argu);
    }

    // 定义属性：私有字段+get方法+set方法
    private int value;
    public int getValue() {
        return value;
    }
    public void setValue(int value) {
        this.value = value;
    }
}
```

结果：

```
Console
<terminated> ClassAndObjectTest
Hello
100
Information
```

从上述示例中，我们可以总结出以下知识点：

- (1) 我们需要定义一个对象变量;
- (2) 然后 “创建 (new)” 一个对象，赋值给对象变量;
- (3) 现在就可以通过对象变量使用对象，主要方式有：
 - (a) 直接调用类的方法;
 - (b) 存取类的字段。

2、早期我们经常这样定义变量

```
int value=100;
```

前面的示例中这样定义变量

```
MyClass obj = new MyClass();
```

这两种方式定义的变量是一样的吗？

答：不一样。如int，float之类的变量称为“原始数据类型”的变量)。

定义一个原始类型的变量时,会马上给其分配内存；“引用”一个对象的变量称为“引用类型”的变量，有时又简称为“对象变量”。当声明一个对象类型的变量时,实际上并没有创建一个对象，此变量=null。

MyClass obj=null

obj



null

int value=100

value



100

3、对于原始数据类型的变量（比如int），可以直接使用“==”判断两变量值是否相等，对象变量也可以使用“==”判断两变量值是否相等吗？

```
public class Test {  
  
    public static void main(String[] args) {  
        int value1=100;  
        int value2=100;  
        System.out.println(value1==value2);//true  
    }  
}
```

请输入并运行以下代码，得到什么结果？

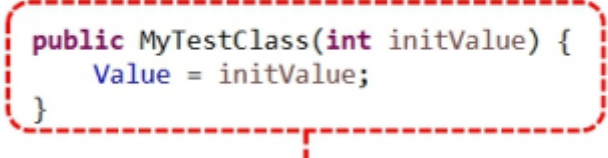
```
public class Test {  
  
    public static void main(String[] args) {  
        Foo obj1=new Foo();  
        Foo obj2=new Foo();  
        System.out.println(obj1==obj2);//?  
    }  
}  
  
class Foo{  
    int value=100;  
}
```

答案是**false**.当“==”施加于原始数据类型变量时，是比较变量所保存的数据是否相等，当“==”施加于引用类型变量时，是比较这两个变量是否引用同一对象。引用代表地址，所以“==”实际上相当于比较两个引用类型变量中保存的对象地址是否相同。

4、ObjectEquals.java

```
public class ObjectEquals {
    public static void main(String[] args) {
        MyTestClass obj1=new MyTestClass(100);
        MyTestClass obj2=new MyTestClass(100);
        System.out.println(obj1==obj2);//判断对象是否相同
        System.out.println(obj1.equals(obj2));//判断属性是否相同
    }
}
class MyTestClass
{
    public int Value;
    //注意：只有参数类型为Object的，才是重写了Object的equals方法
    //参数类型为MyTestClass的，仅仅是Overload了equals方法。
    //    @Override
    public boolean equals(Object obj)
    {
        return ((MyTestClass)obj).Value==this.Value;
    }
    public boolean equals(MyTestClass obj)
    {
        return obj.Value==this.Value;
    }
    public MyTestClass(int initValue)//构造方法可以进行方法重载 初始化，不能进行其他操作
    {
        Value=initValue;
    }
}
```

```
class MyTestClass {  
    public int Value;  
  
    public boolean equals(MyTestClass obj) {  
        return obj.Value == this.Value;  
    }  
  
    public MyTestClass(int initValue) {  
        Value = initValue;  
    }  
}
```



请总结一下，这个方法有哪些“与众不同之处”，你能列出几条？

上述所标出的方法，称为类的“构造方法”，有时也习惯称为“构造函数”。当创建一个对象时，它的构造方法会被自动调用。构造方法与类名相同，没有返回值。如果类没有定义构造函数，Java编译器在编译时会自动给它提供一个没有参数的“默认构造方法”。

5、以下代码为何无法通过编译？哪儿出错了？

```
public class Test {  
  
    public static void main(String[] args) {  
        Foo obj1=new Foo();  
    }  
}  
  
class Foo{  
    int value;  
    public Foo(int initValue) {  
        value=initValue;  
    }  
}
```

结果显示Foo是未定义的。如果类提供了一个自定义的构造方法，将导致系统不再提供默认构造方法。

6、InitializeBlockDemo.java

```
public class InitializeBlockDemo {

    /**
     * @param args
     */
    public static void main(String[] args) {

        InitializeBlockClass obj=new InitializeBlockClass();
        System.out.println(obj.field);

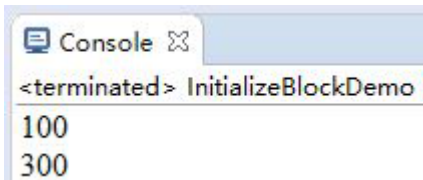
        obj=new InitializeBlockClass(300);
        System.out.println(obj.field);
    }

}

class InitializeBlockClass{
    //下面这句在初始化块之前与之后，会影响到field字段的初始值
    //public int field=100;

    {
        field=200;
    }
    public int field=100;
    public InitializeBlockClass(int value){
        this.field=value;
    }
    public InitializeBlockClass(){
    }
}
```

结果:



```
<terminated> InitializeBlockDemo |
100
300
```

总结：类字段的初始化顺序：

（1）执行类成员定义时指定的默认值或类的初始化块，到底执行哪一个要看哪一个“排在前面”。

（2）执行类的构造函数。

类的初始化块不接收任何的参数，而且只要一创建类的对象，它们就会被执行。因此，适合于封装那些“对象创建时必须执行的代码”。

7、请运行TestStaticInitializeBlock.java示例，观察输出结果，总结出“静态初始化块的执行顺序”。

```
class Root
{
    static{
        System.out.println("Root的静态初始化块");
    }
    {
        System.out.println("Root的普通初始化块");
    }
    public Root()
    {
        System.out.println("Root的无参数的构造器");
    }
}
class Mid extends Root
{
    static{
        System.out.println("Mid的静态初始化块");
    }
    {
        System.out.println("Mid的普通初始化块");
    }
    public Mid()
    {
        System.out.println("Mid的无参数的构造器");
    }
    public Mid(String msg)
```

```

    {
        //通过this调用同一类中重载的构造器
        this();
        System.out.println("Mid的带参数构造器，其参数值：" + msg);
    }
}
class Leaf extends Mid
{
    static{
        System.out.println("Leaf的静态初始化块");
    }
    {
        System.out.println("Leaf的普通初始化块");
    }
    public Leaf()
    {
        //通过super调用父类中有一个字符串参数的构造器
        super("Java初始化顺序演示");
        System.out.println("执行Leaf的构造器");
    }
}

}

public class TestStaticInitializeBlock
{
    public static void main(String[] args)
    {
        new Leaf();

    }
}

```

结果:


```
Console [Java Application]
<terminated> TestStaticInitializeBlock [Java Application]
Root的静态初始化块
Mid的静态初始化块
Leaf的静态初始化块
Root的普通初始化块
Root的无参数的构造器
Mid的普通初始化块
Mid的无参数的构造器
Mid的带参数构造器，其参数值：Java初始化顺序演示
Leaf的普通初始化块
执行Leaf的构造器
```

总结：静态初始化块的执行顺序

- (1) 静态初始化块只执行一次。
- (2) 创建子类型的对象时，也会导致父类型的静态初始化块的执行。

8、静态方法中只允许访问静态数据，那么，如何在静态方法中访问类的实例成员（即没有附加static关键字的字段或方法）？

请编写代码验证你的想法。

答：可采用类的对象实例化进行访问。

```
//类的对象实例化
//王荣荣 2016/10/16
public class SquareIntTest {
    public static void main(String[] args) {
        for (int x=1; x <= 10; x++) {
            SquareIntTest obj;    //创建类的示例obj
            obj=new SquareIntTest();
            int result = obj.square(x);
            // Math库中也提供了求平方数的方法
            // result=(int)Math.pow(x,2);
            System.out.println("The square of " + x + " is " + result + "\n");
        }
    }
    // 自定义求平方数的静态方法
    public int square(int y) {
        return y * y;
    }
}
```

```
}  
}
```

结果：

```
Console  X  
<terminated> SquareIntTest [Java Application]  
The square of 1 is 1  
  
The square of 2 is 4  
  
The square of 3 is 9  
  
The square of 4 is 16  
  
The square of 5 is 25  
  
The square of 6 is 36  
  
The square of 7 is 49  
  
The square of 8 is 64  
  
The square of 9 is 81  
  
The square of 10 is 100
```

9、使用类的静态字段和构造函数，我们可以跟踪某个类所创建对象的个数。请写一个类，在任何时候都可以向它查询“你已经创建了多少个对象？”。

源程序：

```
//王荣荣 2016/10/21  
public class Newclass {  
    public static void main(String[] args) {  
        a1 b1 = new a1();  
        a1 b2 = new a1();  
        a1 b3 = new a1();  
        //我已经创建了3个对象，接下来调用a1类的静态变量查询创建了多少对象
```

```
        System.out.println("我已经创建的对象个数为:  "+a1.value);
    }
}
class a1 {
//将用来计次的变量value赋予初始值为0
    public static int value=0;
    a1() {
        //构造函数，每创建一个实例value就增加1
        value=value+1;
    }
}
```

结果:



The screenshot shows a console window titled "Console" with a close button. Below the title bar, the text "<terminated> Newclass [Java Application]" is displayed. The output of the program is "我已经创建的对象个数为: 3".