**泛型**

**概念**

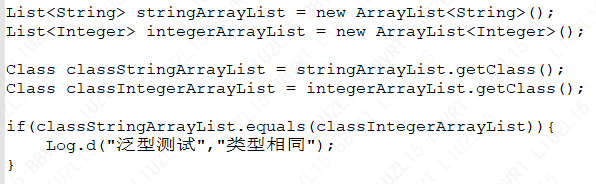
泛型，即“参数化类型”。

**目的**

主要目的之一就是用来指定容器要持有什么类型的对象，而且由编译器来保证类型的正确性。

**作用阶段**

泛型只在编译阶段有效。在编译之后会采取去泛型化的措施。



输出结果

**泛型测试：类型相同**

**注意事项**

泛型的类型参数智能是类类型（包括自定义类），不能是简单类型(int 等)

基本形式

<T>

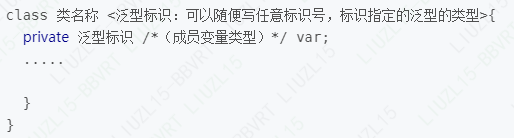
<T,K…..>

T，E，K，V等形式的参数常用于表示泛型

**使用范围**

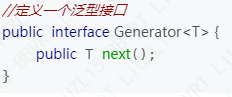
**泛型类**

基本写法



**在实例化泛型类时，必须指定T的具体类型**。

**泛型接口**



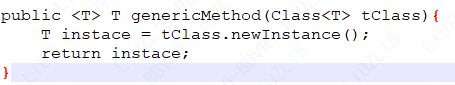
使用：

在声明类的时候，需将泛型的声明也一起加到类中。

class XXXGenerator<T> implements Generator<T>{

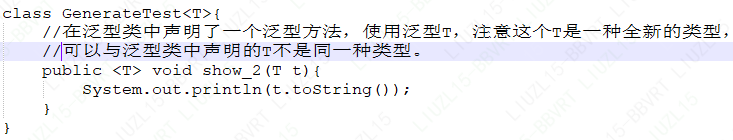
}

**泛型方法**



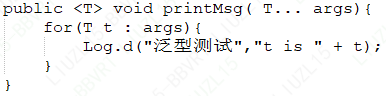
说明

1. 只有声明了<T>的方法才是泛型方法，泛型类中的使用了泛型的成员方法并不是泛型方法。
2. 同时声明了一个泛型 T
3. 及时在泛型类中声明了同样标识的泛型方法，泛型方法中的标识T也是一个全新的类型，可以与泛型类中的标识不一样的类型

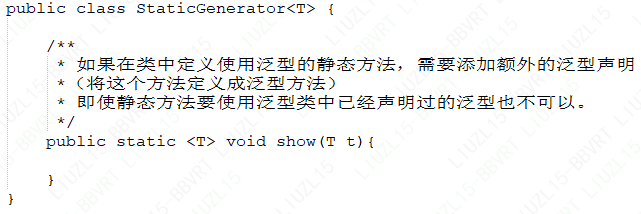


1. 泛型方法能使方法独立于类而产生变化。

泛型方法与可变参数



静态方法与泛型



**如果静态方法要使用泛型的话，必须将静态方法也定义成泛型方法**。

**泛型通配符**

类型通配符一般使用 ? 代替具体的类型实参，此处 ? 是类型实参，而不是类型形参。可以把 ? 看成所有类型的父类，是一种真实的类型。

public void showKeyValue(Generator<?> obj){ }

**泛型上下边界**

使用泛型的时候，我们还可以为传入的泛型类型实参进行上下边界的限制，如：类型实参只准传入某种类型的父类或某种类型的子类。

为泛型添加上边界，即传入的类型实参必须是指定类型的子类型，如：

Generic<? extends Number>

类

public class Generic<? extends Number>{}

泛型方法

public <T extends Number> T showKeyName(Generic<T> obj)

泛型形参

public void showKeyValue(Generic<? extends Number>)

**泛型数组(容器)**

不能创建一个确切的泛型类型的数组。

这个例子是不可以的。

List<String>[] ls = new ArrayList<String>[10];

使用通配符创建泛型数组是可以的，如：

List<?>[] ls = new ArrayList<?>[10];

这样也是可以的，如：

List<String>[] ls = new ArrayList[10];

示例

List<?>[] lsa = new List<?>[10];

// OK, array of unbounded wildcard type.

Object o = lsa;

Object[] oa = (Object[]) o;

List<Integer> li = new ArrayList<Integer>();

li.add(new Integer(3));

oa[1] = li; // Correct.

Integer i = (Integer) lsa[1].get(0); // OK