(1)形式类型参数可以有多个

Myclass<A,B,C>

形式类型的参数不能：

（1）创建对象数组

（2）不能作为夫类型

（3）不能使用instance表达式

（4）不能出现在异常处理中

（5）不能是static 形式

泛型的实现方式：

编译器级别的实现（泛型擦除）

（1）用在类名后 public class<T extends String> 用作方法的参数的类型

（2）public <T> void f(T x) 方法泛型

上界，下界：

（1）public class<T extends String>

（2）public class<S extends T, T>

（3）public class<S extends Cloneable & Serializable>

（4）泛型的申明在前 返回值在后

Public static <T> T (顺序) get(){}

（5）在进行方法调用的时候：xxxx.<String>method(“xxxx”);（调用的是泛型方法）

泛型与数组（只有可具体化类型可以创建数组）：

数组是协变类型 String数组是Object数组的字类型 但是对于list而言，则不是

什么是方法签名：

方法名称和参数类型两个部分

泛型的运用就是为了及早的发现运行时才会发现的错误：



使用泛型后的：



不会再运行时出错的：

渲染类的抽象：

**abstract** **class** ComponentRenderer <C **extends** Component> {

**abstract** **void** render(C component);

}

实现逻辑：

**class** ButtonRenderer **extends** ComponentRenderer<Button> {

**void** render(Button button) {

//界面渲染

}

}

会出错的逻辑：

抽象表示：

**abstract** **class** ComponentRenderer {

**abstract** **void** render(Component window);

}

实现逻辑：

**class** ButtonRenderer **extends** ComponentRenderer {

**void** render(Component window) {

**if** (window **instanceof** Button) {

//界面渲染

}

**else** {

**throw** **new** IllegalArgumentException();

}

}

}

测试类：

**public** **class** NormalComponent {

**public** **void** render() {

ButtonRenderer renderer = **new** ButtonRenderer();

renderer.render(**new** Button());

renderer.render(**new** Label());

}

}