# 第十九章 泛型

## 19.1引言

1.要点提示：泛型可以帮助我们在编译时而不是在运行时检测出错误。我们之前已经使用过泛型类ArrayList，泛型接口Comparable。

2.主要优点：使用泛型的主要优点是能够在编译时而不是在运行时检测出错误。

3.泛型的本质是参数化类型，也就是说所操作的数据类型被指定为一个参数。

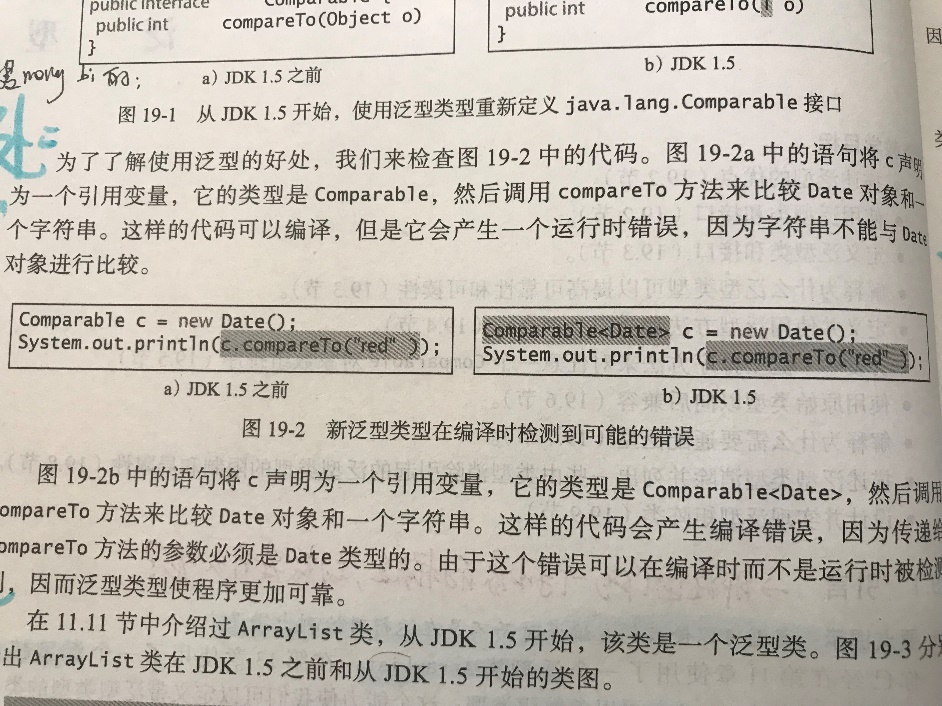
## 19.2动机和优点

1.要点提示：使用Java泛型的动机是在编译时检测出错误。

2.从JDK1.5开始，Java允许定义泛型类、泛型接口和泛型方法。

3.这里的<T>表示“形式泛型类型”，随后可以用一个十几具体类型来替换它。替换泛型类型称为“泛型实例化”。按照惯例，像E或T这样的单个大写字母表示形式泛型类型。

4.为了了解使用泛型的好处，我们来对照图19-2中的代码。如图中的a语句将c声明为一个引用变量，它的类型时Comparable，然后调用compareTo方法来比较Date对象和一个字符串，这样的代码可以“编译”，但是它会产生一个运行时错误，因为字符串不能与Date对象进行比较，如图：



如上图b中的语句将c声明为一个引用变量，它的类型是Comparable<Date>，然后调用compareTo方法来比较Date对象和一个字符串。这样的代码会产生“编译错误”，因为传递给compareTo方法的参数必须是Date类型的。由于这个错误可以在编译时而不是运行时被检测到，因而泛型类型使程序更加可靠。（为啥？）

5.泛型类型必须是引用类型。不能使用int、double或char这样的基本类型来替换泛型类型。例如，下面的语句是错误的：

ArrayList<int> intList=new ArrayList<>();

为了给int值创建了一个ArrayList对象，必须使用

ArrayList<Integer>intList=new ArrayList<>();

可以在intList中加入一个int值。例如，intList.add(5)

Java会自动的将5包装为new Integer（5）。这个过程称为“自动打包”。无需类型转换就可以从一个元素类型已指定的线性表中获取一个值，因为编译器已经知道了这个元素类型。例如，下面的语句创建了一个包含字符串的线性表，然后将字符串加入这个线性表，最后从这个线性表中获取该字符串。

ArrayList<String>list=new ArrayList<>();

List.add(“Red”);

List.add(“White”);

String s=list.get(0);

在JDK 1.5之前，由于没有使用泛型，所以必须把返回值的类型转换为String，如下所示：

String s=(String)(list.get(0));

6.如果元素是包装类型，例如，Integer、Double或Character，那么可以直接将这个元素赋值给一个基本类型的变量。这个过程称为“自动拆箱”，例如一下代码：

1 ArrayList<Double>list=new ArrayList<>();

2 list.add(5.5);

3 list.add(3.0);//将5.5和3.0自动转换为double对象并添加到list中

4 Double doubleObject=list.get(0);

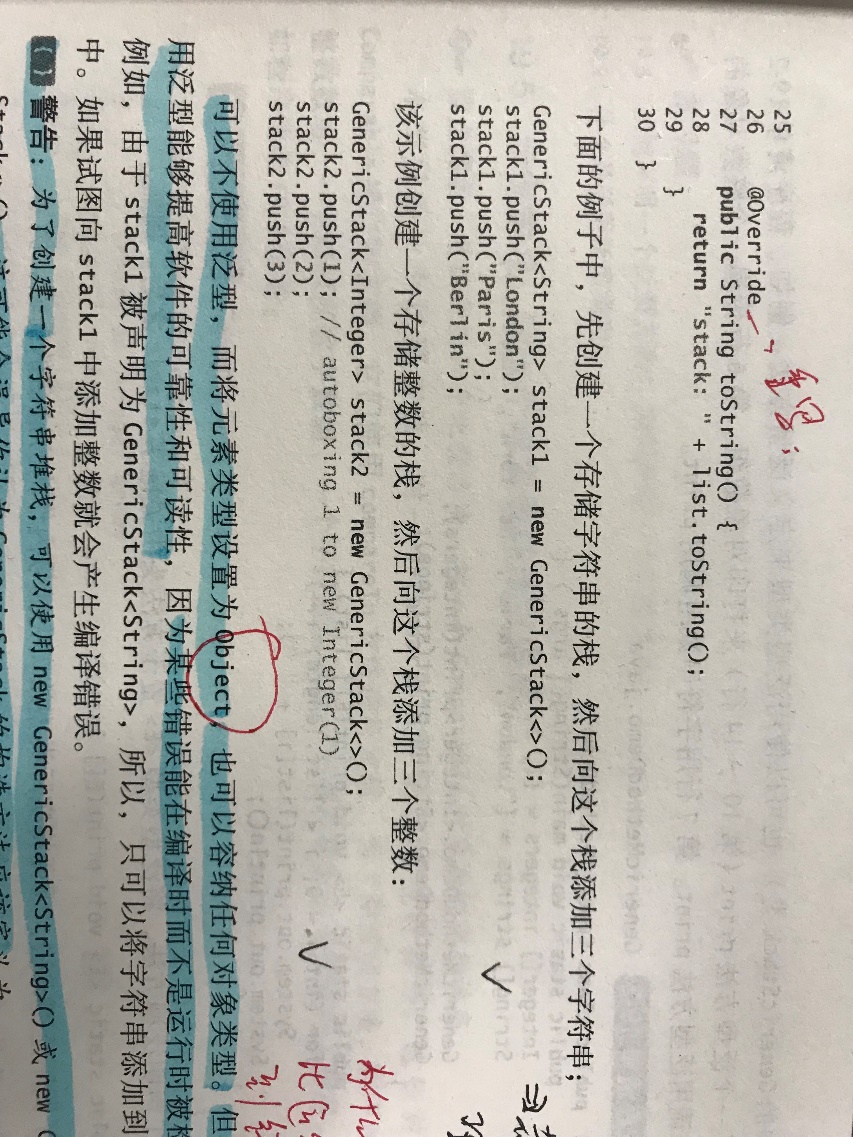
5 double d=list.get(1);//list中的第二个元素被赋给一个double变量。

## 19.3定义泛型类和接口

1.要点提示：可以为类或者接口定义泛型。当使用类来创建对象，或者使用类或者接口来声明变量时，必须指明具体的类型。

2.修改栈类，将元素类型通用化为泛型。实例如下所示：





可以不使用泛型，而将元素类型设为Object，也可以容纳任何对象类型。但是，使用泛型能提高软件的可靠性和可读性，因为某些错误能在编译时而不是运行时被检测到。

3.警告：为了创建一个字符串堆栈，可以使用new GenericStack<String>()或new Generic Stack<>()。这可能会导致你认为GenericStack的构造方法定义为

Public GenericStack<E>()

这是错误的，它应该被定义为 public GenericStack( )

4.注意：有时候，泛型类可以有多个参数。在这种情况下，应该讲所有的参数放在尖括号中，并且用逗号隔开，比如<E1,E2,E3>.

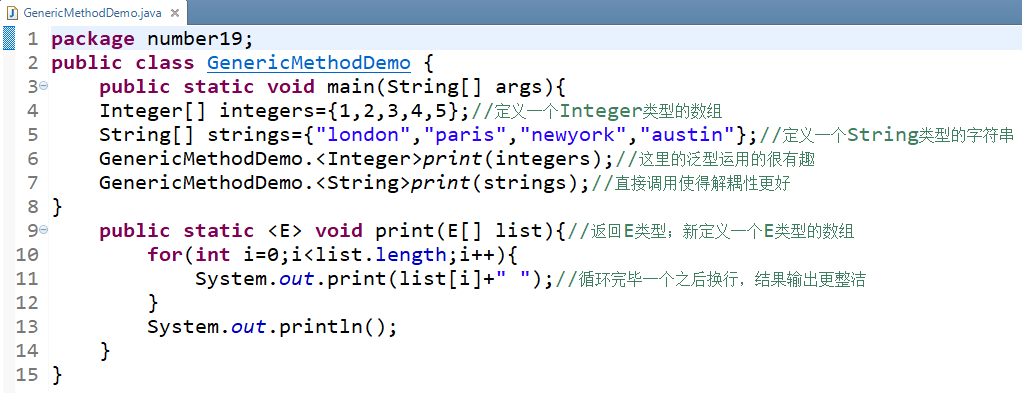
5.注意：可以定义一个类或者接口作为泛型类或者泛型接口的子类型。例如，在Java API中，Java.lang.String类被定义为Comparable接口，如下所示：

Public class String implements Comparable<String>

## 19.4泛型方法

1.要点提示：可以为静态方法定义泛型类型。

2.例如，以下程序定义了一个泛型方法print来打印多种类型的对象：

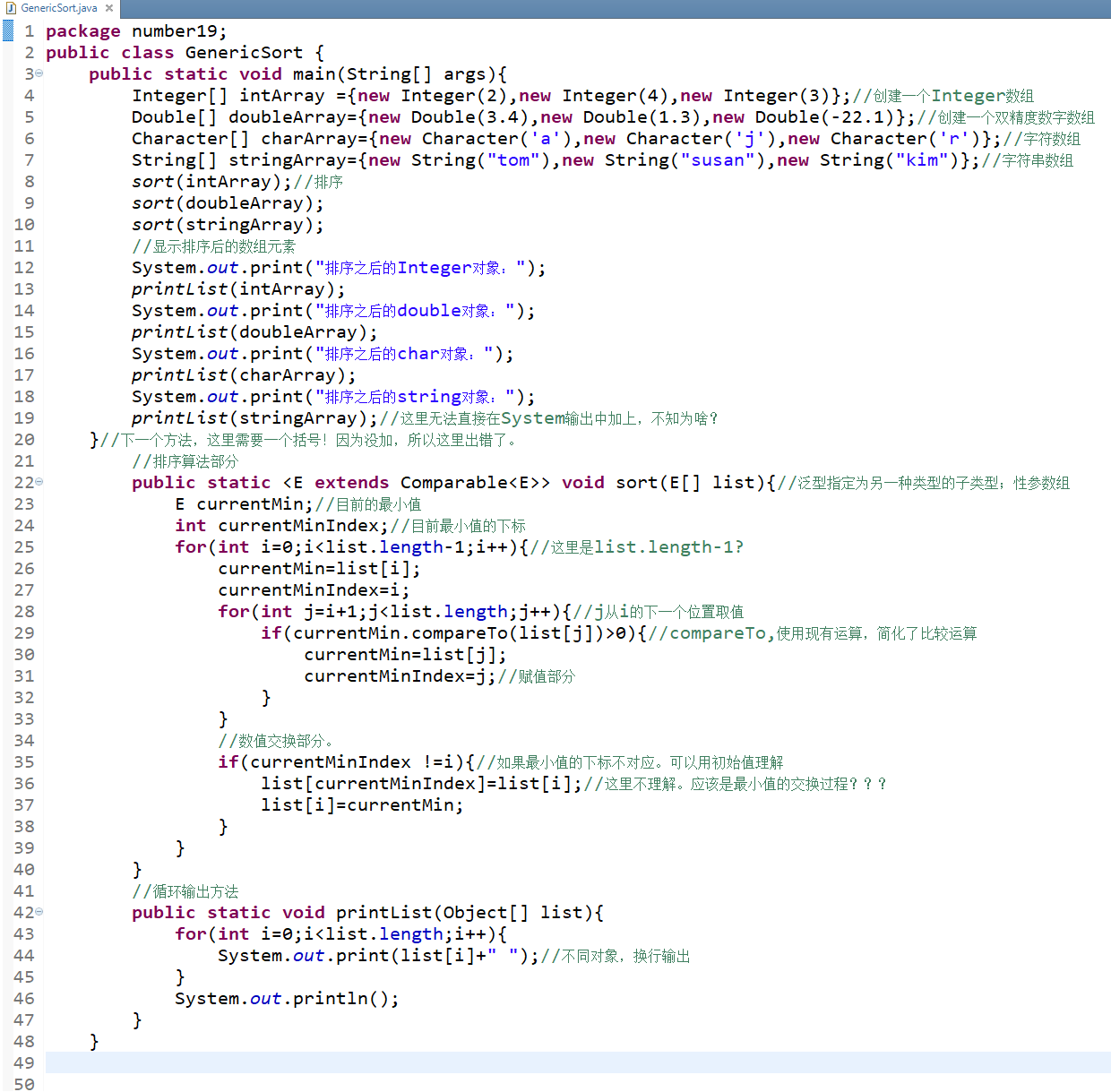


3.可以将泛型指定为另外一种类型的子类型，这样的泛型称为“受限的”。非受限泛型类型<E>等同于<E extends Object>

## 19.5示例学习：对一个对象数组进行排序

1.要点提示：可以开发一个泛型方法，对一个Comparable对象数组进行排序。

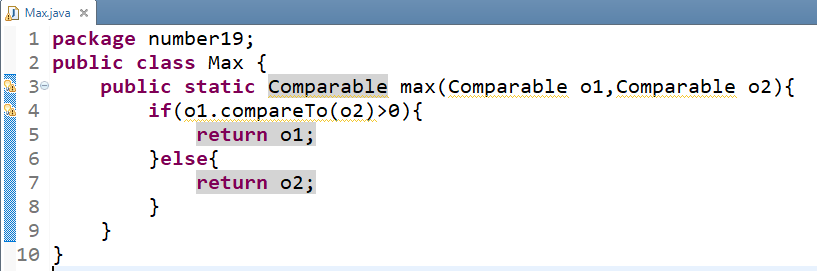
2.本节提供一个泛型方法，对一个Comparable对象数组进行排序。这些接口是Comparable接口的实例，它们使用compareTo方法进行比较。为了测试该方法，程序对一个整数数组、一个双精度数字数组、一个字符数组以及一个字符串数组分别进行了排序。程序如下：



泛型类型定义为<E extends Comparable<E>>这具有两个含义：首先，它指定E是Comparable的子类型；其次，它还指定进行比较的元素是E类型的。

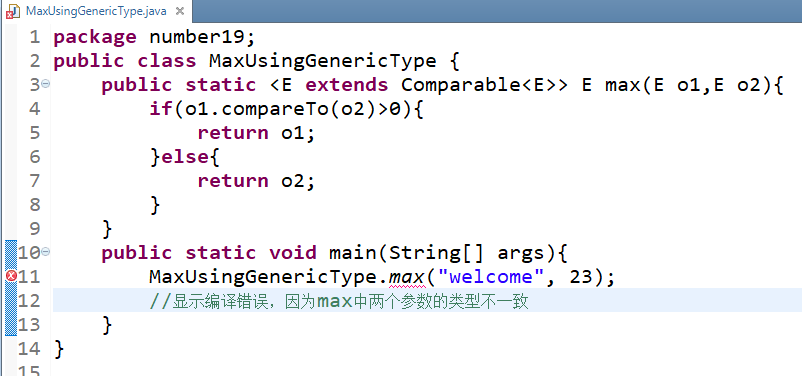
## 19.6原始类型和向后兼容

1.要点提示：没有指定具体类型的泛型类和泛型接口被称为原始类型，用于和早期的Java版本向后兼容。例如，从jdk1.5开始，在Java.lang.Comparable中使用了泛型类型，但是许多代码仍使用原始类型Comparable，例如以下程序所示：



Comparable o1和Comparable o2都是原始类型声明。但是，原始类型是不安全的。

2.编写max方法的更好方式是使用泛型类型，例如以下程序：

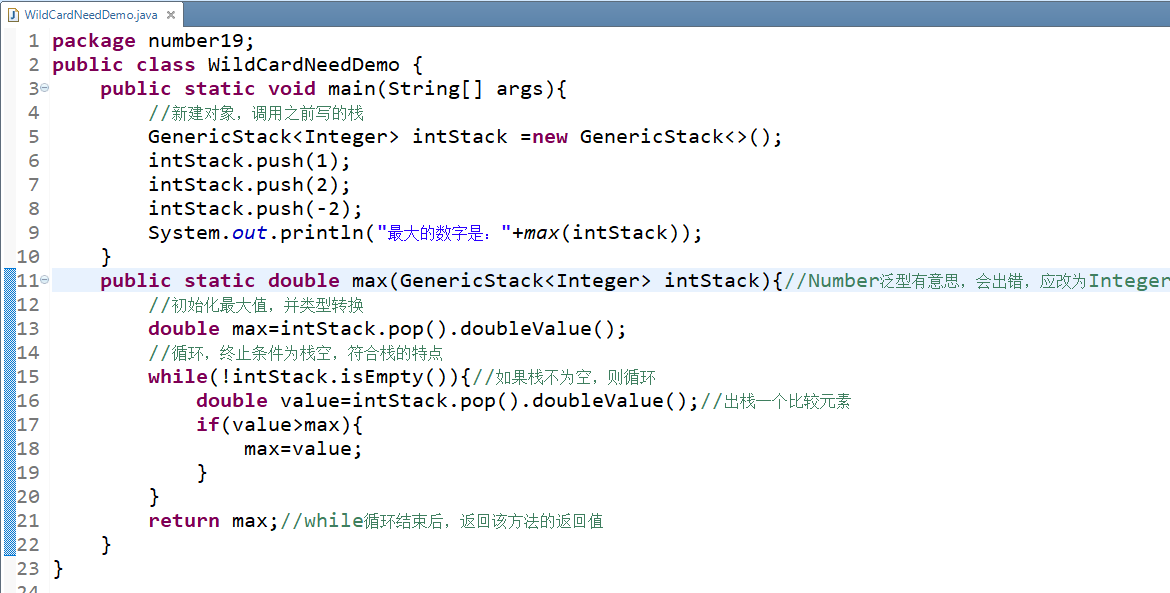


由于原始类型是不安全的，所以本书不再使用原始类型。为啥不安全？

## 19.7通配泛型

1.要点提示：可以使用非受限通配、受限通配或者下线通配来对一个泛型类型指定范围。

2.通配泛型是什么？为什么需要通配泛型？以下程序给出一个示例，以展示为什么需要通配泛型。该例子定义了一个max方法，该方法可以找出数字栈中的最大数。Main方法创建了一个整数对象栈，然后向该栈添加三个整数，最后调用max方法找出该栈中的最大数字。



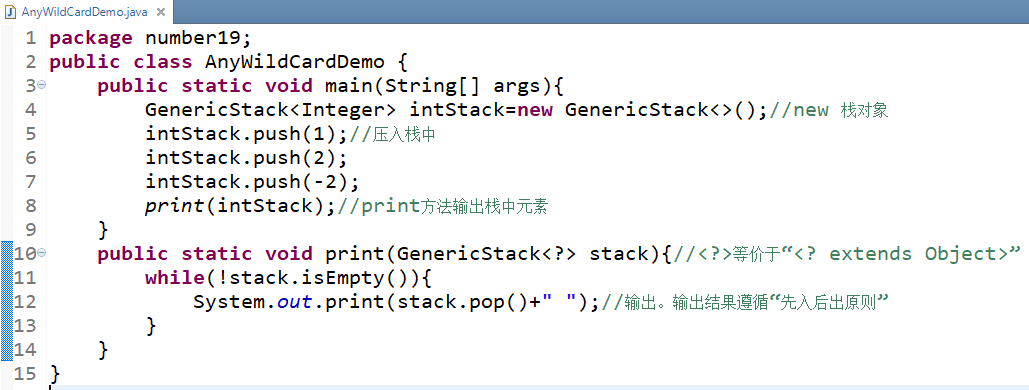
3.通配泛型类型有三种形式——？、extends T 或者 ？ super T，其中T是泛型类型。

第一种形式？称为“非受限通配”，它和？extends Object 是一样的。

第二种形式？extends T 称为“受限通配”，表示T或T的一个子类型。

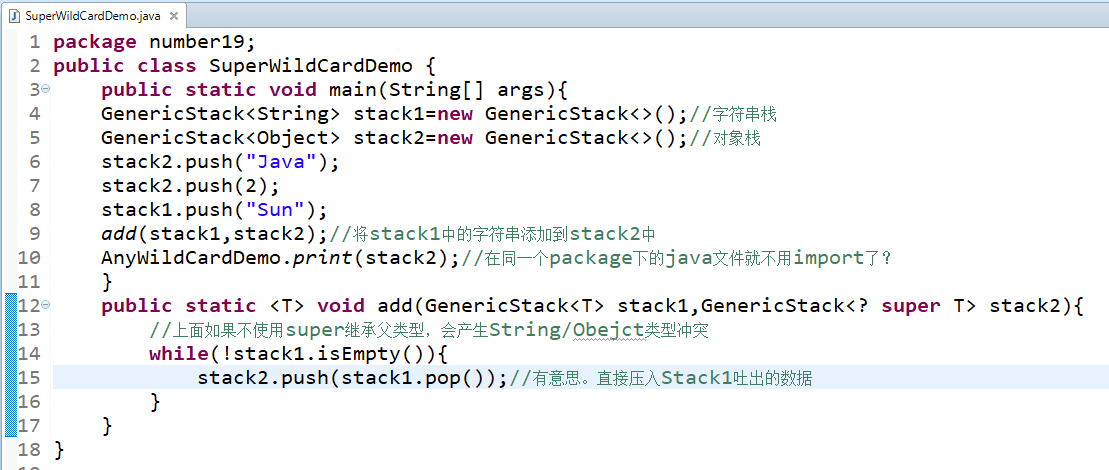
第三种形式？super T 称为“下限通配”表示T或T的一个父类型。

4.以下程序给出一个例子，它在print方法中使用？通配符，打印栈中的对象以及清空栈。<?>是一个通配符，表示任何一种对象类型，它等价于<? extends Object>。



5.以下程序创建一个字符串栈Stack1和一个对象栈Stack2，然后调用add(stack1,stack2)将stack1中的字符串添加到stack2中。

如果用<T>代替<? Super T>，那么在add(stack1,stack2)上就会产生一个编译错误，因为一个是String类型，一个是Object类型。<? Super T>表示类型T或T的父类型，Object是String的父类型。

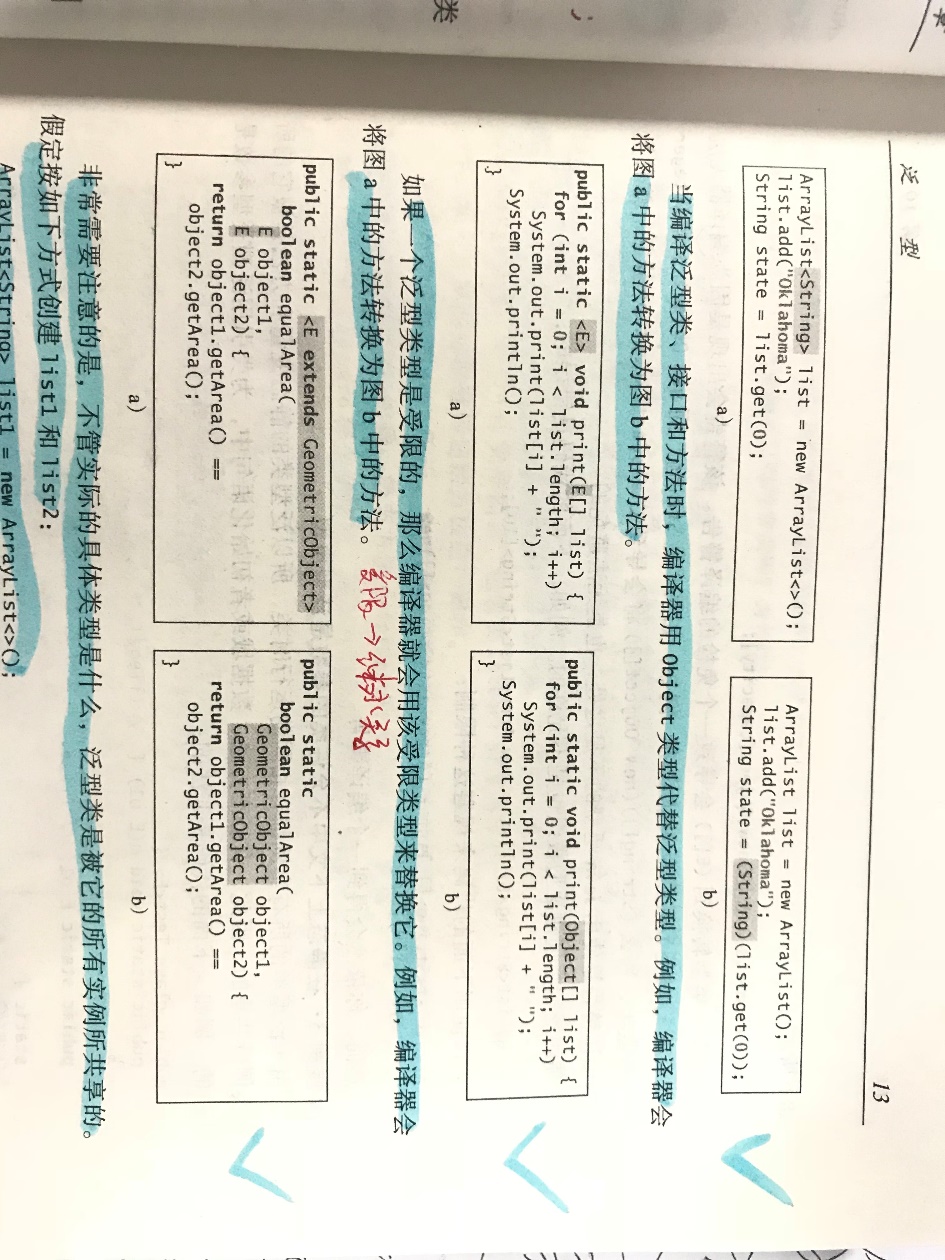


## 19.8消除泛型和对泛型的限制

1.要点提示：编译器可使用泛型信息，但这些信息在运行时是不可用的。这被称为“类型消除”。泛型是使用一种称为“类型消除”的方法来实现的。编译器使用泛型类型信息来编译代码，但是随后会消除它。因此，泛型信息在运行时是不可用的。这种方法可以使泛型代码向后兼容使用原始类型的遗留代码。

2.泛型存在于编译时。一旦编译器确认泛型类型是安全使用的，就会将它转换为原始类型。

（1）例如编译器会检查a图中的代码里泛型是否被正确使用，然后将它翻译成b图中所示的在运行的等价代码。图b中的代码使用的是原始类型：



（2）当编译泛型类、接口和方法时，编译器用Object类型代替泛型类型。例如，编译器会将图a中的方法转换为b中的方法。

（3）如果一个泛型类型是受限的，那么编译器就会用该受限类型来替换它。例如，编译器会将图a中的方法转换为图b中的方法。

3.非常需要注意的是，不管实际具体的类型是什么，泛型类是被它所有的实例所共享的，假定按如下方式创建list1和list2：

ArrayList<String>list1=new ArrayList<>();

ArrayList<Integer>list2=new ArrayList<>();

尽管在编译时ArrayList<String>和Arraylist<Integer>是两种类型，但是，在运行时只有一个ArrayList类会被加载到JVM中。List1和list2都是ArrayList的实例，因此，下面两条语句的执行结果都是true：

System.out.println(list1 instanceof ArrayList);

System.out.println(list2 instanceof ArrayList);

然而，表达式list1 instanceof ArrayList<String>是错误的。由于ArrayList<String>并没有在JVM中存储为单独的一个类，所以，在运行时使用它是毫无意义的。

4.由于泛型类型在运行时被消除，因此，对于如何使用泛型类型有一些限制。下面是其中的一些限制：

（1）限制1：不能只用 new E（）

不能使用泛型类型参数创建实例。

（2）限制2：不能使用new E[ ]

不能使用泛型类型参数创建数组。

（3）限制3：在静态上下文中不允许类的参数是泛型类

（4）限制4：异常类不能是泛型的。

## 19.9示例学习：泛型矩阵

1.要点提示：本节使用泛型类型来设计用于矩阵运算的类。

2.对于所有的矩阵，除了元素类型不同以外，它们的加法和乘法操作都是相似的。因此，可以设计一个父类，不管它们的元素类型是什么，该父类描述所有类型的矩阵共享的通用操作，还可以创建若干个适用于指定矩阵类型的子类。方法addMatrix和multiplyMatrix将泛型类型E[ ][ ]的两个矩阵进行相加和相乘。

