一．泛型

（一）概述

本质是参数化类型，将所操作的数据类型指定为一个参数。即将数据类型都抽象为一个参数T，在使用时传入具体的数据类型，如String,int等等...

这种参数类型可以用在类，方法和接口中，分别称为泛型类，泛型方法，泛型接口。

（二）定义格式

<类型>：指定一种类型的格式

<类型1，类型2>：指定多种类型的格式

（三）优点

将运行时期的异常提前到了编译时期。

二．泛型类

（一）格式

修饰符 class 类名<类型> { }

（二）示例

public class Generic<T> {

  private T t;

  public T getT() {

    return t;

 }

  public void setT(T t) {

    this.t = t;

 }

}

把T作为某种数据类型的参数，代表着某种数据类型。在成员方法中也是使用同样的参数T作为某种数据类型。

当我们调用该泛型类时，传入的数据类型就是T的实参，所有的成员中的T就自动换成了该数据类型。

三． 泛型方法

（一）格式

修饰符 <类型> 返回值类型 方法名(类型 变量名) { }

在普通的方法格式前面加上<类型>，表示传入调用该方法时传入的数据类型。

泛型方法可以用于普通类中，只在调用该泛型方法时可以传入各种数据类型，而创建类对象时不用声明类型。

（二）示例

带有泛型方法的普通类

public class Generic {

  public <T> void show(T t) {

    System.out.println(t);

 }

}

四．泛型接口

（一）格式

修饰符 interface 接口名<类型> { }

（二）示例

接口不像类那样，可以直接创建对象，而是需要实现类来实现接口，因此，定义泛型接口还需要定义泛型接口的实现类。

1. 泛型接口

public interface Generic<T> {

  void show(T t);

}

1. 泛型接口的实现类

public class GenericImpl<T> implements Generic<T> {

  @Override

  public void show(T t) {

    System.out.println(t);

 }

}

注意：实现类的参数也都是T。

五．类型通配符

（一）定义

类型通配符用于表示各种泛型List。

（二）分类

1. 类型通配符 <?>

List<?>：表示元素类型未知的List，它可以匹配任何数据类型

1. 类型通配符上限： <? extends 类型>

如：

List<? extends Number> :表示的类型是Number或者其子类型。

理解： extends关键字用于继承，则？的类型继承的是Number类型，则一定是Number类型或者子类型。

 List<? extends Number> list6 = new ArrayList<Integer>();

1. 类型通配符下限： <? super 类型>

如：

List<? super Number> :表示的类型是Number或者其父类型

理解：与extends相对记忆。

List<? super Number> list7 = new ArrayList<Object>();

六．可变参数

（一）定义

参数的个数是可变的，用作方法的参数个数可变，称为可变参数。

（二）格式

修饰符 返回值类型 方法名(数据类型… 变量名) { }

示例：  
public static int sum(int…a){}

基本的格式与普通方法一样，只是在参数位置处有三个点…，表示可变参数。

（三）注意事项

传入的可变参数保存到数组a中。

如果一个方法不止有可变参数，还有其他参数，则可变参数要放在最后面。

如下所示：

Public static int sum(int b, int …a){}