自定义泛型结构

泛型的声明：

interface List<T>和class GenTest<K,V>

其中T，K，V不代表值，而是表示类型。这里使用任意字母都可以。常用T表示，是Type的缩写。

泛型的实例化：

```
List<String> list = new ArrayList<String>();
Iterator<Customer> iterator = customer.iterator();
》T只能是类，不能用基本数据类型填充，但可以使用包装类填充
》把一个集合中的内容限制为一个特定的数据类型，这就是generic背后的核心思想
》体会：使用泛型的主要优点是能在编译时而不是在运行时检测错误。
```

1、自定义泛型类

1.泛型类可能有多个参数，此时应该将多个参数一起放在尖括号里面。比如：<E1,E2,E3>；

2.泛型类的构造器如下：public GenericClass(){}；

而下面的是错误的：public GenericClass<E> (){};

3.实例化后，操作原来泛型位置的结构必须与指定的泛型类型一致

4.泛型的不同应用不能相互赋值

>尽管在编译时ArrayList<String>和ArrayList<Integer>是两种类型,但是运行时只有一个ArrayList被加载到JVM中

5.泛型如果不指定,将被擦除,泛型对应的类型均按照Object处理,但不等于Object。经验:泛型要使用就一路都使用。要不用,一路都不要用

6.如果泛型是一个接口或抽象类,则不可创建泛型类的对象。

7.jdk 1.7，泛型的简化操作：ArrayList<Fruit> first = new ArrayList<>();

8.泛型的使用中不能使用基本数据类型,可以使用包装类替换

9.在类或接口上声明的泛型，在本类或本接口中即代表某种类型，可以作为非静态属性的类型、非静态方法的参数类型、非静态方法的返回值类型。但在静态方法中不能使用类的泛型。

10.异常类不能是泛型的

11.不能使用 new E[] 。但是可以 E [] elements = (E[])new Object[capacity);

参考：ArrayList 源码中声明： Object[] elementData 而非泛型参数类型数组。

12.父类有泛型,子类可以选择保留泛型也可以选择指定泛型类型:

> 子类不保留父类的泛型:按需实现

> 没有类型,擦除

> 具体类型

> 子类保留父类的泛型

> 全部保留

> 部分保留

结论:子类必须是“富二代”，子类除了指定或保留父类的泛型，还可以增加自己的泛型

```
class Father <T1 , T2>{
}
// 子类不保留父类的泛型
// 1.没有类型:擦除
class Son1 extends Father{// 等价于 class Son1 extends Father<Object, Object>
}
// 2.具体类型
class Son1 extends Father<String, Integer>{
}

 // 子类保留父类的泛型
 // 1.全部保留
```

```
class Son3 <T1 , T2> extends Father<T1 ,T2>{
}
// 部分保留
class Son3 <T2> extends Father<Integer,T2>{
}
```

泛型类举例:

```
class Person<T>{
// 使用T类型定义变量
private T info;
// 使用T类型定义一般方法
public T getInfo(){
return info;
}
public void setInfo(T info){
this.info = info;
 }
 // 使用T类型定义构造器
 public Person(){
 }
 public Person(T info){
 this.info = info;
 }
 }
```

2、自定义泛型方法

1.方法，也可以被泛型化，不管此时定义在其中的类是不是 泛型类。 在泛型方法中可以定义泛型参数，此时，参数的类型就是传入数据的类型 。

2.泛型方法的格式:

[访问权限] <泛型> 返回类型 方法名([泛型标识 参数名称]) 抛出的异常

泛型方法声明泛型时也可以指定上限