一、泛型的出现：

1、泛型是在 JDK1.5 以后出现的新特性。泛型是用于解决安全问题的，是一个安全机制。

2、JDK1.5 的集合类希望在定义集合时，明确表明你要向集合中装入那种类型的数据，无法加入指定类型以外的数据。

3、泛型是提供给 javac 编译器使用的可以限定集合中的输入类型说明的集合时，会去掉“类型”信息，使程序运行效率不受影响，对参数化的泛型类型，getClass()方法的返回值和原始类型完全一样。

4、由于编译生成的字节码会去掉泛型的类型信息，只要能跳过编译器，就可以往某个泛型集合中加入其它类型的数据，如用反射得到集合，再调用 add 方法即可。

二、好处：

1、使用泛型集合，可将一个集合中的元素限定为一个特定类型，集合中只能存储同一个类型的对象；这样就将运行时期出现的问题 ClassCastException 转移到了编译时期，方便与程序员解决问题，让运行时期问题减少，提高安全性。

2、当从集合中获取一个对象时，编译器也可知道这个对象的类型，不需要对对象进行强制转化，避免了强制转换的麻烦，这样更方便。

三、泛型格式：

通过<>来定义要操作的引用数据类型

如：TreeSet<String> -----> 来定义要存入集合中的元素指定为 String 类型

四、泛型定义中的术语：

如：ArrayList<E>类和 ArrayList<Integer>

1、ArrayList<E>整个称为泛型类型

2、ArrayList<E>中的 E 称为类型变量或类型参数

3、整个 ArrayList<Integer>称为参数化类型

4、ArrayList<Integer>中的 Integer 称为类型参数的实例或实际类型参数

5、ArrayList<Integer>中的<>称为 typeof

6、ArrayList 称为原始类型

参数化：parametered，已经将参数变为实际类型的状态。

五、在使用 java 提供的对象时，何时写泛型？

通常在集合框架中很常见，只要见到<>就要定义泛型，其实<>就是用来接收类型的，当使用集合时，将集合中要存储的数据类型作为参数传递到<>中即可。

六、关于参数化类型的几点说明：

1、参数化类型与原始类型的兼容性

第一、参数化类型可引用一个原始类型的对象，编译只是报警告，能不能通过编译，是编译器说了算。

如：Collection<String> coll = new Date();

第二、原始类型可引用一个参数化类型的对象，编译报告警告

如：Collection coll = new Vector<String>();

原来的方法接受一个集合参数，新类型也要能传进去。

2、参数的类型不考虑类型参数的继承关系：

Vector<String> v = new Vector<Objec>();//错误的

不写 Object 没错，写了就是明知故犯

Vector<Objec> v = new Vector<String>();//错误的

3、在创建数组实例时，数组的元素不能使用参数化的类型

如：Vector<Integer> v[] = newVector<Integer>[10];//错误的

当传入的类型不确定时，可以使用通配符?

1、使用?通配符可引用其他各种类型化的类型，通配符的变量主要用作引用，也可调用与参数化无关的方法，但不能调用与参数化有关的方法。

2、可对通配符变量赋任意值：

如：Collection<?> coll ---> coll = newHashSet<Date>();

对于一个范围内的一类事物，可以通过泛型限定的方式定义，有两种方式：

1、? extends E：可接收 E 类型或 E 类型的子类型；称之为上限。

如：Vector<? extends Number> x = newvector<Integer>();

2、? super E：可接收 E 类型或 E 类型的父类型；称之为下限。

如：Vector<? super Integer>x = newvector<Number>();