1、通配符的基本使用

1.使用类型通配符：？

例如：List<？>，Map<?>

List<？>是List<String>、List<Object>等各种泛型类的父类

2.读取List<？>的对象list中的元素时，永远是安全的，因为不管list的真实类型 是什么，它包含的都是Object。

3.写入list中的元素时，不行。因为我们不知道c的元素类型，所以我们不能向其中添加对象

```
// 将任意元素加入到其中不是类型安全的
Collection<?> c = new ArrayList<String>();
c.add(new Object()); // 编译错误
```

》唯一的例外是null，他是所有类型的成员。

》另外我们使用get()方法并使用其返回值。返回值是一个未知的类型，但是我们知道，它总是一个Object。

```
public static void static(String[] args){
 List<?> list = null;
 list = new ArrayList<String>();
 list = new ArrayList<Integer>();
 // 编译不通过
 list.add(null);

 List<String> l1 = new ArrayList<String>();
 List<String> l1 = new ArrayList<String>();
```

```
 l1.add("huahua");
 l2.add(12);
 l2.add(34);
}

public static void read(List<?> list){
 for(Object o:list){
 System.out.println(o);
 }
}
```

注意点：

1.不能用在泛型方法声明上，返回值类型前面的<>里不能使用?

```
public static<?> void test(List<?> list) {}
```

2.不能使用在泛型类的声明上

```
public GenericTypeClass<?>{}
```

3.不能使用在创建对象上，右边属于创建集合对象

```
Collection<?> c = new ArrayList<?>();
```

2、有限制的通配符的使用

1、通配符指定上限

上限extends：使用时指定的类型必须是继承某个类，或者实现某个接口，即<=

? extends A:

G<? extends A> 可以作为G<A>和G<B>的父类，其中B是A的子类

2、通配符指定下限

下限super：使用时指定的类不能小于操作的类

? super A:

G<? super A> 可以作为G<A>和G<B>的父类，其中B是A的父类

3、举例：

》<? extends Number> （无穷小）

只允许Number和Number子类的引用调用

》<? super Number> （无穷大）

只允许Number和Number父类的引用调用

》<? extends Comparable>

只允许泛型为实现Comparable接口的实现类的引用调用

```
@Test
public void test4(){
List<? extends Person> list1 = null;
List<? super Person> list2 = null;

List<Student> list3 = new ArrayList<Student>();
List<Person> list4 = new ArrayList<Person>();
List<Object> list5 = new ArrayList<Object>();

 list1 = list3;
 list1 = list4;
// list1 = list5;

// list2 = list3;
 list2 = list4;
 list2 = list5;

 //读取数据:
 list1 = list3;
 Person p = list1.get(0);
 //编译不通过
 //Student s = list1.get(0);

 list2 = list4;
 Object obj = list2.get(0);
 ////编译不通过
// Person obj = list2.get(0);

 //写入数据:
 //编译不通过
```

```
// list1.add(new Student());

 //编译通过
 list2.add(new Person());
 list2.add(new Student());
 }
```

```
 public static void printCollection3(Collection<? extends Person> coll){
 // Iterator只能用Iterator<?> 或者 Iterator<? extends Person>. Why?
 Iterator<?> iterator = coll.iterator();
 while(iterator.hasNext()){
 System.out.println(iterator.next());
 }
 }

 public static void printCollection4(Collection<? super Person> coll){
 // Iterator只能用Iterator<?> 或者 Iterator<? super Person>. Why?
 Iterator<?> iterator = coll.iterator();
 while(iterator.hasNext()){
 System.out.println(iterator.next());
 }
 }
```