泛型在繼承上的體現：

/\*

1. 泛型在继承方面的体现

虽然类A是类B的父类，但是G<A> 和G<B>二者不具备子父类关系，二者是并列关系。

补充：类A是类B的父类，A<G> 是 B<G> 的父类

\*/

@Test

**public** **void** test1(){

Object obj = **null**;

String str = **null**;

obj = str;

Object[] arr1 = **null**;

String[] arr2 = **null**;

arr1 = arr2;

//编译不通过

// Date date = new Date();

// str = date;

List<Object> list1 = **null**;

List<String> list2 = **new** ArrayList<String>();

//此时的list1和list2的类型不具有子父类关系

//编译不通过

// list1 = list2;

/\*

反证法：

假设list1 = list2;

list1.add(123);导致混入非String的数据。出错。

\*/

show(list1); show1(list2);

}

**public** **void** show1(List<String> list){

}

**public** **void** show(List<Object> list){

}

@Test

**public** **void** test2(){

AbstractList<String> list1 = **null**;

List<String> list2 = **null**;

ArrayList<String> list3 = **null**;

list1 = list3;

list2 = list3;

List<String> list4 = **new** ArrayList<>();//具有子父类关系

}

1. 通配符的使用：

/\*

2. 通配符的使用

通配符：?

类A是类B的父类，G<A>和G<B>是没有关系的，二者共同的父类是：G<?>

\*/

@Test

**public** **void** test3(){

List<Object> list1 = **null**;

List<String> list2 = **null**;

List<?> list = **null**;

list = list1;

list = list2;

//编译通过

// print(list1);

// print(list2);

List<String> list3 = **new** ArrayList<>();

list3.add("AA");

list3.add("BB");

list3.add("CC");

list = list3;

//添加(写入)：对于List<?>就不能向其内部添加数据。

//除了添加null之外。

//编译不通过

// list.add("DD");

// list.add('?');

list.add(**null**);

//获取(读取)：允许读取数据，读取的数据类型为Object。

Object o = list.get(0);

System.out.println(o);

}

**public** **void** print(List<?> list){

Iterator<?> iterator = list.iterator();

**while**(iterator.hasNext()){

Object obj = iterator.next();

System.out.println(obj);

}

}

1. 涉及通配符的集合的數據的讀取和寫入：

見上。

1. 有限制條件的通配符的使用：

/\*

3.有限制条件的通配符的使用。

? 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

(可以當作list範圍為A至其子...子類 ( <=A ))

? 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

(可以當作list範圍為A至其最大父類(Object) ( >=A ))

\*/

@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;

// 编译不通过

// 　 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);

//因為繼承person,範圍最大為person,所以只能定義為其最大父類person

list2 = list4;

Object obj = list2.get(0);

////编译不通过

// Person obj = list2.get(0);//範圍為person至其父類,所以只能定義為其最大父類Object

//写入数据：

//编译不通过

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

//因為list1定義範圍的最小子類不確定,所以不能添加可能大過於它的對象

//编译通过

//因為list2定義範圍的最小子類為person,所以對象為person或其子..子類都可以添加

list2.add(**new** Person());

list2.add(**new** Student());

}