# 请大家阅读文档时，在视图里勾选导航窗格，在左边显示章节目录方便浏览。

# 一、编程题1

下列代码定义了迭代器接口，数组迭代器和容器类：

*/\*\*  
 \* 迭代器接口，用于遍历组件树里的每一个组件. 注意这不是java.util.Iterator接口  
 \*/*interface Iterator {  
 */\*\*  
 \* 是否还有元素  
 \** ***@return*** *如果元素还没有迭代完，返回true;否则返回false  
 \*/* boolean hasNext();  
  
 */\*\*  
 \* 获取下一个元素  
 \** ***@return*** *下一个元素  
 \*/* Object next();  
}  
  
*/\*\*  
 \* 数组迭代器  
 \*/*class ArrayIterator implements Iterator{  
 private int pos = 0;  
 private Object[] a = null;  
  
 public ArrayIterator(Object[] array){  
 a = array;  
 }  
  
 @Override  
 public boolean hasNext() {  
 return !(pos >= a.length);  
 }  
  
 @Override  
 public Object next() {  
 if(hasNext()){  
 Object c = a[pos];  
 pos ++;  
 return c;  
 }  
 else  
 return null;  
 }  
}  
  
*/\*\*  
 \* 容器类，内部用Object[]保存元素  
 \*/*class Container {  
 private Object[] elements;  
 private int elementsCount = 0;  
 private int size = 0;  
  
 public Container(int size){  
 elements = new Object[size];  
 this.size = size;  
 }  
  
 public boolean add(Object e){  
 if(elementsCount < size){  
 elements[elementsCount ++] = e;  
 return true;  
 }  
 else{  
 return false;  
 }  
 }  
  
 */\*\*  
 \* 返回容器的迭代器  
 \** ***@return*** *\*/* public Iterator iterator(){  
 return new ArrayIterator(elements);  
 }  
}  
public class Test{  
 public static void main(String[] args){  
 Container container = new Container(6);  
 container.add("12");  
 container.add("34");  
 container.add("56");  
 container.add("78");  
 container.add("9");  
 container.add(10);  
 Iterator it = container.iterator();  
 while (it.hasNext()){  
 String s = (String)it.next();  
 if( s != null)  
 System.*out*.println(s);  
 }  
 }  
}

1. 上述代码存在什么问题？请分析存在问题的原因。

**container.add(10);一行存在问题，由于container中存在int类型数据10，不属于String类型数据，会在遍历时抛出类型转换异常。**

2. 请将迭代器接口Iterator、数组迭代器ArrayIterator、容器Container分别改成泛型迭代器接口Iterator<T>、泛型数组迭代器ArrayIterator<T>、泛型容器Container<T>，并写出和上面一样的测试代码。要求泛型代码实现和非泛型接口/类同样的方法，另外泛型容器Container<T>也必须实现iterator方法。

*/\*\**

*\* 迭代器接口，用于遍历组件树里的每一个组件. 注意这不是java.util.Iterator接口  
 \*/*interface Iterator<T> {  
 */\*\*  
 \* 是否还有元素  
 \*  
 \** ***@return*** *如果元素还没有迭代完，返回true;否则返回false  
 \*/* boolean hasNext();  
  
 */\*\*  
 \* 获取下一个元素  
 \*  
 \** ***@return*** *下一个元素  
 \*/* T next();  
}  
  
*/\*\*  
 \* 数组迭代器  
 \*/*class ArrayIterator<T> implements Iterator<T> {  
 private int pos = 0;  
 private T[] a = null;  
  
 public ArrayIterator(T[] array) {  
 a = array;  
 }  
  
 @Override  
 public boolean hasNext() {  
 return !(pos >= a.length);  
 }  
  
 @Override  
 public T next() {  
 if (hasNext()) {  
 T c = a[pos];  
 pos++;  
 return c;  
 } else  
 return null;  
 }  
}

import java.lang.reflect.Array;  
  
*/\*\*  
 \* 容器类，内部用T[]保存元素  
 \*/*class Container<T> {  
 private Object[] elements;  
 private int elementsCount = 0;  
 private int size = 0;  
  
 public Container(int size){  
 elements = new Object[size];  
 this.size = size;  
 }  
  
 public boolean add(T e){  
 if(elementsCount < size){  
 elements[elementsCount ++] = e;  
 return true;  
 }  
 else{  
 return false;  
 }  
 }  
  
 */\*\*  
 \* 返回容器的迭代器  
 \** ***@return*** *new ArrayIterator<T>((T[])elements)  
 \*/* public Iterator<T> iterator(){  
 return new ArrayIterator<T>((T[])elements);  
 }  
}

public class Test {  
  
 public static void main(String[] args) {  
 *testInteger*();  
 *testString*();  
 }  
  
 public static void testInteger() {  
 System.*out*.println("-----Test Integer-----");  
 Container<Integer> container = new Container<>(6);  
 container.add(1);  
 container.add(1);  
 container.add(3);  
 container.add(4);  
 container.add(5);  
 container.add(10);  
 Iterator<Integer> it = container.iterator();  
 while (it.hasNext()) {  
 System.*out*.println(it.next());  
 }  
 }  
  
 public static void testString() {  
 System.*out*.println("-----Test String-----");  
 Container<Object> container = new Container<>(6);  
 container.add("12");  
 container.add("34");  
 container.add("56");  
 container.add("78");  
 // container.add(10);  
 Iterator<Object> it = container.iterator();  
 while (it.hasNext()) {  
 String s = (String) it.next();  
 if (s != null)  
 System.*out*.println(s);  
 }  
 }  
}

# 二、编程题2

实现一个泛型二元组类TwoTuple<T1, T2>，其中T1、T2分别是二元组第1个、第2个元素的类型参数，要求如下：

（1）T1、T2是实现了Comparable接口的类型；

（2）TwoTuple<T1, T2>必须实现Comparable接口；

（3）TwoTuple<T1, T2>必须覆盖equals方法和toString方法；

（4）TwoTuple<T1, T2>二个私有数据成员变量名分别为first, second；

（5）必须实现私有数据成员first, second的公有getter和setter方法;

（6）Comparable接口的compareTo方法实现的语义是：如果二个元组对象的first部分不相等，以二个对象first成员比较结果作为最终比较结果；如果二个元组对象的first部分相等，则以二个对象second成员比较结果作为最终比较结果；

（7）覆盖equals方法的实现语义是：二个元组对象的first和second分别都相等时，这个二个元组对象相等。

当实现好泛型类之后，通过下面的代码进行测试：

public static void main(String[] args){  
  
 TwoTuple<Integer,String> twoTuple1 =new TwoTuple<>(1, "ccc");  
 TwoTuple<Integer,String> twoTuple2 =new TwoTuple<>(1, "bbb");  
 TwoTuple<Integer,String> twoTuple3 =new TwoTuple<>(1, "aaa");  
 TwoTuple<Integer,String> twoTuple4 =new TwoTuple<>(2, "ccc");  
 TwoTuple<Integer,String> twoTuple5 =new TwoTuple<>(2, "bbb");  
 TwoTuple<Integer,String> twoTuple6 =new TwoTuple<>(2, "aaa");  
 List<TwoTuple<Integer,String>> list = new ArrayList<>();  
 list.add(twoTuple1);  
 list.add(twoTuple2);  
 list.add(twoTuple3);  
 list.add(twoTuple4);  
 list.add(twoTuple5);  
 list.add(twoTuple6);  
  
 //测试equals，contains方法是基于equals方法结果来判断  
 TwoTuple<Integer,String> twoTuple10 =new TwoTuple<>(1, "ccc"); //内容=twoTuple1  
 System.*out*.println(twoTuple1.equals(twoTuple10)); //应该为true  
 if(!list.contains(twoTuple10)){  
 list.add(twoTuple10); //这时不应该重复加入  
 }  
  
 //sort方法是根据元素的compareTo方法结果进行排序，课测试compareTo方法是否实现正确  
 Collections.*sort*(list);

for (TwoTuple<Integer, String> t: list) {  
 System.*out*.println(t);  
 }

TwoTuple<TwoTuple<Integer,String >,TwoTuple<Integer,String >> tt1 =  
 new TwoTuple<>(new TwoTuple<>(1,"aaa"),new TwoTuple<>(1,"bbb"));  
 TwoTuple<TwoTuple<Integer,String >,TwoTuple<Integer,String >> tt2 =  
 new TwoTuple<>(new TwoTuple<>(1,"aaa"),new TwoTuple<>(2,"bbb"));  
 System.*out*.println(tt1.compareTo(tt2)); //输出-1  
 System.*out*.println(tt1);

}

上述测试代码的输出结果应为：

true

(1, aaa)

(1, bbb)

(1, ccc)

(2, aaa)

(2, bbb)

(2, ccc)

-1

((1, aaa), (1, bbb))

package Exercise2;  
  
import java.util.Objects;  
  
public class TwoTuple<T1 extends Comparable, T2 extends Comparable> implements Comparable {  
 private T1 first;  
 private T2 second;  
  
 public TwoTuple(T1 first, T2 second) {  
 this.first = first;  
 this.second = second;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 TwoTuple<T1, T2> T = (TwoTuple<T1, T2>) o;  
 return Objects.*equals*(first, T.first) && Objects.*equals*(second, T.second);  
 }  
  
 public T2 getSecond() {  
 return second;  
 }  
  
 public void setSecond(T2 second) {  
 this.second = second;  
 }  
  
 public T1 getFirst() {  
 return first;  
 }  
  
 public void setFirst(T1 first) {  
 this.first = first;  
 }  
  
 @Override  
 public String toString() {  
 return "(" + first + "," + second + ")";  
 }  
  
 @Override  
 public int compareTo(Object o) {  
 TwoTuple<T1, T2> T = (TwoTuple<T1, T2>) o;  
 if (first.equals(T.first))  
 return second.compareTo(T.second);  
 else  
 return first.compareTo(T.first);  
 }  
}

package Exercise2;  
  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.List;  
  
public class TestTwoTuple {  
 public static void main(String[] args) {  
  
 TwoTuple<Integer, String> twoTuple1 = new TwoTuple<>(1, "ccc");  
 TwoTuple<Integer, String> twoTuple2 = new TwoTuple<>(1, "bbb");  
 TwoTuple<Integer, String> twoTuple3 = new TwoTuple<>(1, "aaa");  
 TwoTuple<Integer, String> twoTuple4 = new TwoTuple<>(2, "ccc");  
 TwoTuple<Integer, String> twoTuple5 = new TwoTuple<>(2, "bbb");  
 TwoTuple<Integer, String> twoTuple6 = new TwoTuple<>(2, "aaa");  
 List<TwoTuple<Integer, String>> list = new ArrayList<>();  
 list.add(twoTuple1);  
 list.add(twoTuple2);  
 list.add(twoTuple3);  
 list.add(twoTuple4);  
 list.add(twoTuple5);  
 list.add(twoTuple6);  
  
 //测试equals，contains方法是基于equals方法结果来判断  
 TwoTuple<Integer, String> twoTuple10 = new TwoTuple<>(1, "ccc"); //内容=twoTuple1  
 System.out.println(twoTuple1.equals(twoTuple10)); //应该为true  
 if (!list.contains(twoTuple10)) {  
 list.add(twoTuple10); //这时不应该重复加入  
 }  
  
 //sort方法是根据元素的compareTo方法结果进行排序，课测试compareTo方法是否实现正确  
 Collections.sort(list);  
  
  
 for (TwoTuple<Integer, String> t : list) {  
 System.out.println(t);  
 }  
  
 TwoTuple<TwoTuple<Integer, String>, TwoTuple<Integer, String>> tt1 =  
 new TwoTuple<>(new TwoTuple<>(1, "aaa"), new TwoTuple<>(1, "bbb"));  
 TwoTuple<TwoTuple<Integer, String>, TwoTuple<Integer, String>> tt2 =  
 new TwoTuple<>(new TwoTuple<>(1, "aaa"), new TwoTuple<>(2, "bbb"));  
 System.out.println(tt1.compareTo(tt2)); //输出-1  
 System.out.println(tt1);  
 }  
}