ASSIGNMENT 1 is about the Adapter Pattern.

(1) First you have to complete the implementation of the Adaptee.

The following describes the public part of the **ADAPTEE**. This code is provided as SortedList.java

You have to write the binary search methods that are used in the *add*, *contains*, and *remove* methods that are already provided.

Package assignment01
Class SortedList<E extends Comparable<? super E>>

---

public class SortedList<E extends Comparable<? super E>> extends Object
A class that maintains a list of elements that are sorted according to the natural ordering of the generic parameter E. Many classes are Comparable, see Comparable. Elements are stored in an ArrayList. Operations use binarySearch rather than linear search for efficiency.

Constructor
SortedList()

Public Method Details
public void add(E e)
Inserts the specified element at the correct position in the sorted order. If there are multiple elements equal to e, the new copy of e enters as the last copy of e in the list.
Parameters: e - element to be inserted.

public boolean contains(E e)
Returns true if this list contains the specified element. More formally, returns true if and only if this list contains at least one element o such that Objects.equals(o, e).
Parameters: e - element whose presence in this list is to be tested
Returns: true when this list contains the specified element.

public boolean remove(E e)
Removes the first occurrence of the specified element from this list, if it is present. If the list does not contain the element, it is unchanged. More formally, removes the element with the lowest index i such that Objects.equals(e, get(i)) (if such an element exists). Returns true if this list contained the specified element (or equivalently, if this list changed as a result of the call).
Parameters: e - element to be removed from this list, if present
Returns: true when this list contains the specified element.

Implementation of the methods *binarySearchHigh* and *binarySearchLow*.
You will find the binary search algorithm in many places, an LLM will write it for you. BUT there are changes needed.

(i) You cannot use == for equality. In Java, you test equality of non-primitive variables using val1.equals(val2) which returns true or false provided val1 is not null. You only use val1 == val2 when the types of the variables are primitive: byte, short, int, long, char, boolean, float, double. Also == should be used if val1 and val2 refer to a Singleton such as an enum type, although equals also works. Using the equals method is an error for the *primitive* types.

(ii) When you have a *comparable* type as in this exercise, you test for less-than using val1.compareTo(val2) < 0, which is true when val1 comes before val2 in the ordering of E
You test for greater-than using val1.compareTo(val2) > 0, which is true when val1 comes after val2 in the ordering of E.

(iii) When the method *binarySearchHigh(list, e)* locates an element *e* in *list*, it has to move the index **up** until the last position where *e* occurs in *list*. Then it returns 1 more than that. The purpose is to find the location where we would add a new copy of the same element *e*.
If the method *binarySearchHigh(list, e)* does not locate the element *e* in *list*, then it returns -*left*-1, where we assume you are using the standard local counters *left* and *right* in your binary search.
The purpose of such a return value, call it *index*, is that *-index-1* is the location where the element *e* that was not found should be inserted into *list* to maintain the correct order.

(iv) When the method *binarySearchLow(list, e)* locates an element *e* in *list*, it has to move the index **down** until the first position where *e* occurs in *list*. Then it returns that index. The purpose is to find the location where we would remove the first copy of the element *e* in *list* when we call *remove*.
If the method *binarySearchLow(list, e)* does not locate the element *e* in *list*, then it returns -*left*-1, as explained at the end of (iii).

<u>You must add the following code to SortedList.java:</u>

```
public String toString() {
        return internalList.toString();
}
public List<E> asList() {
        return internalList;
}
public int size() {
        return internalList.size();
}
```

<u>You must add the following code to SampleComparable.java</u>

```
public static void resetNEXT_ID () {
        NEXT_ID = 0;
}
```

Part (2) is to implement the following SortedSet interface by writing an object adapter and a class adapter for SortedList.

```
public interface SortedSet<E extends Comparable<? super E>> {
        void add(E e);
        boolean contains(E e);
        boolean remove(E e);
        List<E> asList();
        int size();
}
```

You need to complete ObjectAdapter.java and ClassAdapter.java. Then run the Driver for testing.