

Laboratory 5: Tree Sort

In this lab, you will write a **treeSort** method that accepts an array of KeyedItems and uses a binary search tree to sort the array.

KeyedItem is an **abstract class**, objects must extend KeyedItem if they are to be sorted with the **treeSort** method.

Iterator

The Iterator interface will allow you to visit all of the items in a collection. The **hasNext()** method tells you whether there are more items to process, and the next() method will retrieve the next item to process. Use a while loop to continue processing elements in a collection until hasNext() is false.

- public boolean hasNext() //are there any more items to process?
- public Object next() //get the next item in the collection to process

Example:

```
Iterator iter = new Iterator();

while (iter.hasNext()) {

    Object myObject = iter.next();

    //do something with myObject

}
```

Lab:

- Comparable ("natural ordering")
- KeyedItem (immutable)
- CD
 - Song
- Sorting
 - TreeSort

Part I: KeyedItem (immutability)

Write the abstract KeyedItem class. It has a single instance variable of type Comparable so that value-oriented data structures can compare keys and determine where to place or find the object with that key, and this instance variable is set only once by the constructor.

There is a **getKey** method that returns the key, but **no setKey** method. The `getKey` method exposes the key and can result in the key being changed unless the key itself is immutable.

The `String` class and the primitive wrapper classes are immutable and `Comparable`, so these types can be safely used as keys. It is possible to use other classes as keys (as long as they are `Comparable`), but you must be very careful in writing such classes to maintain immutability of the key.

Put `KeyedItem` in a **package** called **ki** by placing it in the appropriate **subdirectory** (also named `ki`) so that the binary search tree classes provided can import the `KeyedItem` class.

Part II: CD

Complete the `CD` class.

- [CD.java](#) //the year is the search key

Complete the `CD` class by extending `KeyedItem`. Set the year of the `CD` as the key field (use the primitive wrapper class for integers, the `Integer` class).

Make sure that there is no `setYear` method in your `CD` class to ensure that the key is immutable. A `getYear` method is not necessary as `KeyedItem` already has a `getKey` method. Write the following two methods:

- `public Song getSng(int index):` //Return the song with the given index as parameter. Make sure index is valid or not.
- `public void addSng(String title, String length)` //create a new song using the arguments and add the song list after checking if the number of elements in the song lists are less than `numTracks`(we do not want to exceed the limit)

Part III: Tree Sort

Complete the following two methods in a utility class (**`TreeSort.java`**):

- **Method Overloading** is a feature that allows a class to have more than one method having the same name, if their argument lists are different.
- `public static KeyedItem[] treeSort(KeyedItem[] unsorted)` //convenience method to sort all of the items in the array (calls the next method).

- `public static KeyedItem[] treeSort(KeyedItem[] unsorted, int n) //sort the first n items in the array (check n for validity)`

The array passed in (unsorted) is not directly sorted. Create a new array with size equal to the number of items to be sorted. Return this new array with the items in sorted order.

You can use the **Class** class and the **Array** class to figure out what the actual type of the objects in the array are, and create an array of that type. To get the type of the array, use the **getComponentType()** method in the **Class** class. Your driver will still need to downcast the array as a whole from **KeyedItem[]** to the type of the array, but you won't need to cast each individual object in the array. To accomplish this, refer to the Java API. You are interested in the **newInstance()** method in the **Array** class. (Done for you).

Refer to the **BinarySearchTree** documentation to create a BST that will **allow duplicate keys** and will **preserve FIFO ordering for duplicate keys**. Place all of the items from the unsorted array into the tree and perform the appropriate traversal to get the items back out in sorted order (**look at the documentation for the **Iterator** and **TreeIterator** interfaces**). Place the sorted items in the array that you instantiated and that will be returned to the user.

Part IV: Tree Sort Driver

- [TreeSortDriver.java](#)
- [Keyboard.java](#)
- [make.bat](#)
- [cds.txt](#)

Complete the provided driver to test your work. You may want to write a `make.bat` file that includes the classpath information that you need for the jar file. You will also need to refer to the **FileIO** documentation (see the top of the lab). Use try-catch to continue to prompt for a valid file until the end user enters one.

Some detailed info:

This lab is about the object-oriented concept of **polymorphism, or is-a relationships**.

Abstract **KeyedItem** class is that the object extending this class has a field/instance variable (the **key**) that should not be changed after it has been set in the constructor (**immutability**). This key is then the field

that **value-oriented data structures** will use to place or find the corresponding object in the collection. If this key is not protected from change, the value-oriented data structure containing the object can be corrupted.

- [FileIOException.java](#) //has the following public interface:
- [FileIO.java](#) //has the following public interface:
 - `FileIO(String fileName, int operation)` //either `FileIO.FOR_READING` or `FileIO.FOR_WRITING`
 - `boolean EOF()` //has the end of the file been reached?
 - `String readLine()`
 - `void writeLine(String line)`
 - `void close()` //close the file when you are done, especially when writing to a file
- [Song.java](#) //constructor takes the title and the length of the Song as Strings

There is a potential problem with creating a new array and returning it to the user, however. You don't know the actual type of the objects in the array, just their supertype (`KeyedItem`). You can create and return an array of type `KeyedItem[]`, but if the user needs a particular object in the sorted array, they will have to downcast from `KeyedItem` to the actual type of the object. If you need to process all of the items in the array, that is a lot of casting.