Laboratory 5: Tree Sort

Java API
FileIO Documentation
Lab 05 Documentation

In general, if you want to sort an array of objects, you must write a sort method that can operate on that specific type of array, which can result in many versions of the sort method. Writing a single sort method that can handle arrays containing a large number of different types of objects is an excellent way to achieve code reuse for sorting. Such a sort method leverages the powerful object-oriented concept of **polymorphism, or is-a relationships**.

In this lab, you will write a **treeSort** method that accepts an array of KeyedItems (see later) and uses a specific traversal of a binary search tree to sort the array. As 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 interface is very easy to use. 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)

The idea behind the 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.

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. Note that 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

Download the following files:

- 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
- 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 getSong(int index)
- public void addSong(String title, String length) //create the song and then add it

Part III: Tree Sort

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

- 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)

As a convenience to the user, the array passed in (unsorted) is not directly sorted. Instead, 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 just in case the unsorted array is still needed by the user for some reason.

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.

Instead, 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

Download the following files:

- 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.

Only one submission per team is necessary, but please make sure to include both names at the top of your source code, as well as in the comments section when submitting the lab, so both people can get credit.