Tutorial 7

Objectives

Practice with Linked Lists and the Java Collections Framework

---

Attendance Quiz

Please log on to cuLearn using one of the computer in the tutorial room and complete the attendance quiz. You can only access the quiz if you log in using one of the computers in the lab. You cannot use a laptop for this. This is a time limited quiz. Be sure to do this as soon as you arrive.

At the end of the tutorial a TA will assign you a grade, call it G, which is 0, 1 or 2, depending on the progress you make during the tutorial. If you spend your time reading mail and chatting with your friends you will receive 0. If you have completed the attendance quiz on time then G will be your tutorial grade. If you have not completed the attendance quiz on time, then your tutorial grade will be max(0, G - 1/2). Each tutorial grade will therefore be one of 0, 0.5, 1.5 or 2.

Project Teams

Starting with this tutorial, you will work on the tutorial material as a group. You will each still write the tutorial quiz at the start of the tutorial, but the ending tutorial grade will be the same for each member in the team.

It is expected that your team will make significant progress on the Linked List problem of this tutorial. Do not worry if you do not reach the Generics/JCF portion.

Linked Lists

Recall that a **list** is a collection of ordered data x_0, x_1, x_2, ..., x_n-1. Here, the length of the list is n.

Download the **Node**, **LList** and **TestLL** classes. Read through the classes and discuss within your group. Be sure you understand all of the **Node** class. Look at what is provided and what is missing in the **LList** class. Your task for this tutorial is to complete the **LList** class and build up the **TestLL** class.

You should be doing this as a group. Discuss what you need to do, break up and do some individual work, combine the individual work together, and then repeat. Don't do everything at once. Take small steps. What methods seem easiest to solve first? Which methods will be easier when others are already done? Think about these things before you start. Try working in pairs instead of individuals if you find getting started is hard.

1. implement the **set(int k, String s)** method: sets x_k to be s.
2. implement the **swap(int p1, int p2)** method: swaps the data in x_p1 and x_p2
3.

implement the **removeFront()** method: removes x_0, the list adjusts itself to be length n-1.
4. implement the **remove(int k)** method: removes x_k, the list adjusts itself to be length n-1
5. implement the **find(String s)** method: returns first k such that s_k = s, return -1 if s is not in the list
6. implement the **compareTo** method: lists are compared by their lengths. A longer list is *greater* than a shorter list. Two lists are *equal* if their lengths are the same.
7. implement a static **same(LList, a, LList b)** method that returns **true** if both lists are the same (and **false** otherwise). The notion of *same* is the intuitive notion of two lists being the same (not the == sense).

Add a few lines to the **TestLL** program to test the **find** method.

Generics

All of the classes in the Java Collections Framework (JCF) use generics. As we saw with the **Comparable** interface, one purpose of this is that it lets us specify which type of objects we are dealing with (we can think of the class as having a parameter that specifies a type inside it). Note that when using generics we can only use reference data types. Java does not allow us to use primitive data types with generics (and this is one very good reason we have the primitive wrapper classes). We will cover generics in more detail in class later.

ArrayList

The **ArrayList** is a commonly used class in the the JCF. It implements the list abstract data type (ADT).

https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

Open up a web browser and see what methods the ArrayList class provides. In particular, look at **add(E e)**, **add(int index, E e)**, **get(int index)**, **remove(int index)**, **set(int index, E e)**, and **size()**. Note, **E** is the type you specify when creating the ArrayList (as follows):

```
import java.util.ArrayList;

...

/* empty string list (E is String here) */
ArrayList<String> slist = new ArrayList<String>();

/* empty integer list (E is Integer here) */
ArrayList<Integer> ilist = new ArrayList<Integer>();
```

1. Modify your **TestLL** program so that for each linked list in the program you have a mirror list that is an ArrayList. Use the same operations (when applicable) to create the ArrayLists.
2. Overload the **same** method to take an **LList** and an **ArrayList** that checks if two lists are the same.

HashSet

The **HashSet** class in the JCF implements the Set ADT. A set is an unordered collection of unique items. Important methods include **add**, **contains**, **remove**, **isEmpty** and **size**.

https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html

```
import jave.util.HashSet;
...
HashSet<String> set = new HashSet<String>();
```

Run the **SetORList** program. Rerun the program several times with different values of *size* (see the main method). Try 1000, 10000, 20000, 50000, etc.

What can we say about using an ArrayList and using a HashSet?

Modify the **SetORList** to generate a collection of 10 items (Integers for example). Sort the ArrayList using Collections.sort(). This is similar to Arrays.sort(), but it works for Collections rather than Arrays.

How do you sort the data in the set?

Map

The **Map** class implements the Dictionary ADT. It stores <key,value> pairs and so you will see it has two generic parameters. The first parameter is for the type of the keys and the second is for the type of the values.

https://docs.oracle.com/javase/8/docs/api/java/util/Map.html

```
import java.util.Map;
...
Map<String, String> dictionary = new Map<String, String>();

dictionary.put("cat", "small cute mammal");
dictionary.put("dog", "small mammal slightly less cute than cat");
```

Write a Java method **randomNumbers(int size, int range)** that returns an ArrayList of exactly size unique numbers in between 0 and range (inclusive). The numbers should be generated randomly.

Your method should create a Map using using Integer:Integer key:value pairs. Each time you generate a random integer (use the **Random** class), you should check if that number is a key in the Map. If it is, increment its value in the Map. Do not add this number to your arraylist to return. If the number is not in the Map, add it to the Map with value 1 and add the number to the arraylist to return.

Once your arraylist has size numbers in it, you can return it. When you return the arraylist, what information does the Map store?

Extra

1. Write a static method that takes two **LList** objects that are each already sorted in alphabetical order, and returns a new **LList** that contains all elements of the two input lists in alphabetical order. For example, if **l1 = [A]->[D]->[G]** and **l2=[B]->[C]->[H]->[W]**, then the output will be **[A]->[B]->[C]->[D]->[G]->[H]->[W]**.
2. Use a Map to find the frequency distribution of words in a body of text (a large string). You can use a Scanner object to break the big string up into individual words (tokens). Find the 10 most frequent words

in the text.