CS 250 - Fall 2021 - Lab 8: ADT, List, Linked List

Available: Oct 20, 2021
Due date: **Oct 25, 2021, 11:59 PM**

In this lab, we will practice different abstract data types which belong to the **Java.util.Collections** package in Java.

Objectives:

The objectives of this lab are:

- To understand different abstract data types in the Collections package: LinkedList, Stack, Set, HashSet, Map, and HashMap.
- To practice coding abstract data types.

Prerequisite and Resource:

- Read Java API documentation about LinkedList, Set, HashSet, Map, and HashMap, etc.

Part 1: Testing the List interface and LinkedList class

In the first part of this lab, we will develop a Java application that uses Java's LinkedList class. The LinkedList class is very similar to the ArrayList class in that they both implement the List and Collection interfaces. Like the ArrayList, you can do different operations such as adding an item, getting an item, and removing an item from the list. In addition, the LinkedList class introduces a few *convenient* methods for operating on the first and last elements in the collection.

The methods which implemented the List<E> interface:	**The additional *convenience* methods in the LinkedList class**:
- add(index, E): add an item at the index - add(E): add an item at the end - get(index): get an item at the index - remove(index): get an item at the index	- addFirst(E): add an item to the front - addLast(E): add an item to the end - getFirst(): get the first item in the list - getLast(): get the last item in the list - removeFirst(): remove the first item - removeLast(): remove the last item

1. Create a new project named Lab8 and create a class named LinkedListApp

2. Copy the following skeleton to replace the generated code in Eclipse

```
public class LinkedListApp {
	//Generate a LinkedList of Integer between 0-n
	private static LinkedList<Integer> genLLn(int n) {
		//TODO add code below
	}

	//Return the reverse of the input linked list
	private static LinkedList<Integer> reverse(LinkedList<Integer> list) {
		//TODO add code below
	}

	//Calculate the sum of the power of two of the elements in the input linked list
	private static double sumPower2LL(LinkedList<Integer> list) {
		//TODO add code below
	}

	public static void main(String[] args) {
		//TODO add code below
	}
}
```

3. **Complete the method genLLn(int n)**: This method receives the input positive number n and generates a LinkedList object which stores all integer numbers between 0 and n. For example if we call genLLn() with n=5, we get a LinkedList<Integer> object with the following values {0, 1, 2, 3, 4, 5}.

Hint: first, check if the input n is valid (positive). If not, throw a new IllegalArgumentException. If n is positive, create a LinkedList of Integer objects. Then, use a for loop to insert numbers from 0 to n into the LinkedList object.

4. **Complete the method reverse(LinkedList<Integer> list)**: This method receives an input LinkedList of Integer and returns a reverse order of elements of such a LinkedList. For example, if we call reverse() on the following LinkedList {0, 1, 2, 3, 4, 5}, we get the reverse LinkedList: {5, 4, 3, 2, 1, 0}.

Hint: Simply use a loop to add the number at the end of the list to the first to a new LinkedList variable.

5. **Complete the method sumPower2LL(LinkedList<Integer> list)**: This method computes and returns the sum of power of two of all elements in the input LinkedList<Integer>. For exame, if we call sumPower2LL() on the following LinkedList {0, 1, 2, 3, 4}, we get sum = 0*2^0 + 1*2^1 + 2*2^2 + 3*2^3 + 4*2^4 = 98.

Hint: Use a for loop on the elements of the input LinkedList and use Math.pow() to calculate the power of two.

6. Add code to the **main**() method to test the above three methods:

- Create a Scanner object
- Prompt a text "Enter a positive number (n): "
- Use the Scanner object to read the number n
- Get the LinkedList of Integer object by calling the method genLLn() with the number n
- Print the above LinkedList to the Console
- Now, get the reverse of the above LinkedList by calling the reverse() method
- Print the second LinkedList (the reversed of the first LinkedList)
- Call the method **sumPower2LL** with the first LinkedList and print the result.

Part 2: Testing the Set interface and HashSet class

A set is a group of items with no duplicates. The Set interface does not require ordering (Java's HashSet), but some set implementations can be ordered (Java's TreeSet). Sets support at least the following operations:

- **add**(): Adding an element to the set if it's not already there.
- **remove**(): Removing an element from the set.
- **contains**(): Checking whether a given item is in the set.

Java's HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements the Set interface.

Implement this program using the following steps:

1. Copy the following code to the LinkedListApp.java file before the main() method:

```
/**
 * Remove the duplicate elements in a linked list
 * @param list: the LinkedList of Integer
 * @return: the Set of Integer which containing no-duplicate elements from the input
list
 */
private static Set<Integer> removeDuplicate(LinkedList<Integer> list) {
        //TODO add code below
}
```

2. **Complete the removeDuplicate() method**. This method receives an input LinkedList of Integer. It needs to return a Set of Integers that contains no duplicate elements from the LinkedList object. Use the following hints:

- Create a Set object of Integer
- For each integer number in the input list
    - If the number is not in the Set object, add it to the Set object

Hint: There are two ways to insert an element to a Set safely:

- You can use the method **Set.add**() to try adding an element to a Set object. If this method returns false, then such an element already exists in the Set object.
- Use the method **Set.contains**() to check if an element exists in the Set object before inserting the element to the Set object.

3. Add code to the **main()** method to test the **removeDuplicate** method with the following modification:

- Prompt a text "Enter the list of integer numbers: "
- Create a LinkedList object of Integer
- Use the Scanner object to read an input line
- Split the input line into a String array using the space delimiter
- Parse each string element in the String array into an Integer and add it to the LinkedList object
- Remove the duplicate elements and get a Set object by calling the removeDuplicate() method
- Print the numbers in the Set object

Part 3: Testing the Map interface and HashMap class

A map is a collection of key-value mappings, like a dictionary from Python. As a set, the keys in a map are unique. Maps also do not need to maintain order (Java's HashMap), but some map implementations can be ordered (TreeMap, for example). Maps support at least the following operations:

- **put**(), and **putIfAbsent**(): Adding a key-value pair to the map or updating the value if the key exists.
- **remove**(): Removing a key-value pair from the map.
- **containsKey**() or **containsValue**(): Checking whether a given key or value is on the map.
- **get**(): Returning the value associated with a given key.

Unlike set and list, Map is not a direct sub-interface of the Java Collection interface. This is because Collection specifies collections of a single element type, but Map operates on key-value pairs. Instead, "The Map interface provides three *collection views*, which allow a map's contents to be viewed as a set of keys, collection of values, or set of key-value mappings."

- keySet(): returns a Set of all the keys. The keys are unique, so a set is an appropriate choice here.
- values(): returns a Collection of all the values. The values are not necessarily unique, so we prefer a more general Collection rather than a Set.
- entrySet(): returns a Set of key-value pairs with a wrapper type called Entry. The entry class's job is just to hold the key and the value together in a single class, so you can imagine that it might look something like this.

In this part, we will create a HashMap object to store the person's id and name. The program will accept the following commands:
- A <person_id> <person_name>: This command will add a new entry to the HashMap object with key = person id, value = person name
- R <person_id>: This command will remove an entry with key = person_id in the HashMap
- SK <person_id>: This command search and display for an entry with key = person_id in the HashMap
- SV <person_name>: This command search and display for an entry with value person_name in the HashMap
- V: print the HashMap to the Console

The following steps guide you through implementing the code for this part:

1. Create a new class named HashMapApp:

```
public class HashMapApp {
        public static void main(String[] args) {
                //TODO add code below
        }
}
```

2. **Complete the main() method:** To test the HashMap of person id and name:

- Create a HashMap object
- Create a Scanner object
- while(true) {
    - Use the Scanner object to read an input line
    - String[] arr = Split the line using a space delimiter
    - If arr[0] equals "A", then add an entry to the HashMap with key=arr[1], value=arr[2]
    - If arr[0] equals "R", then remove an entry to the HashMap with key=arr[1]
    - If arr[0] equals "SK", then search and display an entry in the HashMap with key=arr[1]

- If arr[0] equals “SV”, then search and display an entry in the HashMap with value=arr[1]
- If arr[0] equals “V”, then display all entries in the HashMap
- Otherwise, exit

- }

Lab Assignment

Complete the following assignment of this lab:

1. Implement the following method that merges two LinkedList<String> objects into one LinkedList but does not contain duplicate string elements. For example, merging of two LinkedList {“e”, “f”, “a”} and {“b”, “a”, “d”, “c”} will result as a LinkedList {“e”, “f”, “a”, “b”, “d”, “c”}

```
public static LinkedList<String> merge(LinkedList<String> list1, LinkedList<String> list2)
{
        //TODO Add code below
}
```

2. Implement the following method that receives two Set of String objects and return another Set of String that contain only common elements in the given two Sets. For example, if you call the common() method on the following two sets {“e”, “f”, “a”, “c”} and {“b”, “a”, “d”, “c”}, it should return a Set with values {“a”, “c”}.

```
public static Set<String> common(Set<String> set1, Set<String> set2) {
        //TODO Add code below
}
```

Lab 8 Result Submission:

You need to submit the results of the following sections to the D2L assignment item of Lab8:

- Complete the required Lab Assignment

- Compress the whole Eclipse source code of your Lab8 project in zip format using the Export function