COM6516 Object oriented programming and software design: Practical session 5

The aim of this Lab class is to give you experience working with the classes provided by the Java Collections Framework. You should work through the following tasks, completing the coding in your own time if needed.

A useful tutorial introduction to the Java Collections Framework – http://docs.oracle.com/javase/tutorial/collections/index.html

Task 1 – Lists

Write a java program `AnimalNames.java` that creates a list of names and converts them to upper case. A good starting point for this program is to create a `String` array and convert it to an implemented List:

```
List<String> fixedList = Arrays.asList("elephant", "lion", "leopard", "tiger");
System.out.println(fixedList);
List<String> myList = new LinkedList<String>(fixedList);
```

Take a look at the API documentation (http://docs.oracle.com/javase/8/docs/api/) to see the methods that are available. Note that `Arrays.asList` returns a fixed-size list, so you cannot use `remove` and `add` methods on it or its `iterator`.

We can use a `ListIterator` to navigate through the `LinkedList`, and remove and add elements at a particular location. A reference to an `iterator` is returned by the `iterator` method, so in your code you could write either of

```
Iterator<String> iter = myList.iterator();
```

or

```
ListIterator<String> iter = myList.listIterator();
```

The `Iterator` has three principal methods (see the API documentation); `iter.next` will return the next element of the `LinkedList`, as well as move the `Iterator` to point to the next element in the list. A `ListIterator` also has methods for moving backwards and adding an element to the list. Use the `Iterator` to iterate through the list of `Strings` making use of the `toUpperCase` method of the `String` class to convert all lower case characters in each `String` to upper case characters. These can either be displayed on the screen, stored in a new List, or used to replace the contents of the `List` (using the `Iterator`'s remove and add methods). Hint: look up `Iterator.hasNext` here – https://docs.oracle.com/javase/8/docs/api/

As an example, the list

```
elephant, lion, leopard, tiger
```

should be converted to:

```
ELEPHANT, LION, LEOPARD, TIGER
```

Task 2 – Hash sets

Use the program `HashSetTest.java` and the `Person` class as a starting point for this part of the exercise. `HashSetTest` reads the first name, surname and age of a list of people from `person.txt`, stores the data in a `HashSet` of `Person` objects and uses an `Iterator` to display the data.

a) Modify `HashSetTest` to use an `Iterator` over the `Set` so that only the details of people with surname "`James`" or "`Cole`" are displayed.

b) Use an `Iterator` over the `Set` to remove all people with surname "`Wright-Phillips`".

c) Starting with the basic `HashSetTest` program again, change the `HashSet` to a `TreeSet` and run the program again. Why is the output different?

d) Change the declaration of the people variable in `HashSetTest` to the following:

```
Set<Person> people = new TreeSet<Person>(new AgeComparator());
```

Run the program again. Why is the output different?

e) Change the program so that the list of people is displayed in reverse order of age.

d) comparator is set to use the age value to compare different instances, if the age is the same, then the two objects are regarded as the same one thus not added into the set.

Task 3 – More Lists

The program `Shakespeare.java` reads the text file `shakespeare.txt`, storing each word in the file into a `List`.

a) Use an `Iterator` to display all the words in the `List` beginning with the letter '`a`'.

b) Produce a sorted list of the words in the file. Consider implementing the `Comparator` interface – https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html to make a case insensitive comparator.

c) Produce a count of how many times each word (ignoring the case of each letter) appears in the file. Try to complete it by sorting.

In order to complete this task, you will be using some of the methods of the `Collections` class; sorting requires comparators which are classes implementing the `Comparator` interface.

Task 4 – Lists again

Write and test a code `MoreList` with a method `eitherNotBoth` that takes two lists of words and returns a new list that contains all elements that occur in either input list, but not in both. Here you would want to use collections that implement the `Set` interface but convert the result back to a `List` for output.

Task 5 – Even more Lists

Write a program `EvenMoreList.java` that creates two lists (**a** and **b**) each with at least three elements, and performs the following operations on them:

a) List **b** is merged into **a** in an interleaved fashion (so if **a** = [*x,y,z*] and **b** = [*p,q,r,s*] then **a** becomes [*x,p,y,q,z,r,s*] while **b** does not change).

b) Every second element is then removed from **b** (so in the example **b** would become [*p,r*] while **a** does not change).

c) Every element that is in **b** is removed from **a** (so **a** would become [*x,y,q,z,s*] and **b** does not change).