GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Department of Computing & Mathematics

CP2SD – Data Structures & Algorithms
Lab 2: Lists and Collections

Lists and collections are indispensible data structures and can be found in all but the most trivial of applications. Although the Java Collections framework (*java.util* library) contains excellent implementations of the data structures (or components) that we will be studying on this course, knowledge of the inner workings of these structures is essential for their proper usage. Thus, we will be implementing our own versions of many of these data structures, even though far better implementations are already available. It is important however, to be completely *au fait* will all the main structure in the *java.util* library for your future career as a software developer!

- Download the Zip archive ***lab2Lists.zip*** from Moodle and extract the files into a folder on your hard disk. As the classes involved are all packaged, ensure that you create the correct directory structure and study the source code before you compile.

The classes contained in the Zip archive provide a basic implementation of a single-linked list and have all been parameterized with a generic type. Recall from the lecture, that adding and removing from a linked list is much faster than using an *ArrayList* for similar purposes.

- ***Node.java:*** An abstraction of a list element. Contains an instance variable of the parameterized type and an instance variable that points at the next node in the linked list.
- ***SingleLinkedList.java:*** Represents a single-linked list. Contains the necessary methods to add, remove and iterate over the list.
- ***Iterator.java:*** Our own implementation of an Iterator. Note: do not import the

Tasks
- Create a class called *Test.java* and add a main method that instantiates the linked list and adds a few elements:
```
SingleLinkedList<String> sll = new SingleLinkedList<String>();
sll.add("John");
sll.add("Paul");
sll.add("Mary");
```
- Use the methods *add(), addFirst(), addAfter(), removeFirst(), removeAfter()* to exercise the essential methods of the linked list. Make sure you understand why adding, removing and insertion is faster than an indexed implementation of a list.
- Use the iterator to transverse the linked list:
```
Iterator<String> i = sll.iterator();
while(i.hasNext()){
   System.out.println(i.next());
}
```

Exercises

1. Using the *Dictionary* class from last week's lab, read the *String[]* into both a single-linked list and an *ArrayList* and compare the two for add/removal running time by inserting new words at the beginning, end and middle of each collection.
2. Implement a double-linked list by extending the functionality of *SingleLinkedList* and *Node* to accommodate a pointer to the previous node as well as the next node.
3. Create a circular-linked list using *SingleLinkedList* by add the head of the list as the next node to the list tail.
4. Finally, examine the JavaDocs API for both the ArrayList and LinkedList classes in the java.util library. Read in the String[] from the Dictionary class into both collections and examine the running time of the add and remove methods.