Lancaster University

## Week 3: Insights into Interfaces and Collections

Something a little different this week! Today we'll be looking at how to write a class that is compliant with a given interface, and also have some fun discovering how to use and abuse dynamic data structures in Java.

### Task 1: Writing a test program for the Java Collections API

Your task is to create a class that is compliant with the **CollectionTest** interface, that can be used to measure the runtime performance of some standard dynamic data structures that are included in the Java API – **LinkedList, ArrayList and HashMap**. In particular you are to create code that can test **add**, **index lookup** and **search** operations on each of these collections.

As part of the fun, I've also provided a visualization tool for you, so you will also be able to develop a real-life awareness of the relative costs of these operations and data structures! 😊

Download the resources zip file supplied alongside this document and extract the java files contained within. You will find three files in the zip file:

- **Person.java**. Our favourite class. A simple class that we've seen in the lectures. You may notice I have added a few more additional methods – don't worry about these for now. We'll talk about them in the lectures next week 😊

- **CollectionTest.java**. Take a look at the source code in this file. Note this is not a class, but an **interface**. Read the code and Javadoc to understand the purpose of the methods. Note this interface also makes use of enumerations – we covered these in the Week 1 task 1 review. If you don't understand enumerations, you might want to go back and review this.

- **PerformanceVisualizer.java**. A Java class that performs automated tests on classes that implement the **CollectionTest** interface. It also produces a HTML web page ("results.html") that provides graphs demonstrating how long it took to undertake ADD, SEARCH and INDEX operations on the **ArrayList**, **LinkedList** and **HashMap** data structures.

- **You MAY NOT (AND DO NOT NEED TO) EDIT ANY OF THESE FILES TO COMPLETE THE TASK.**

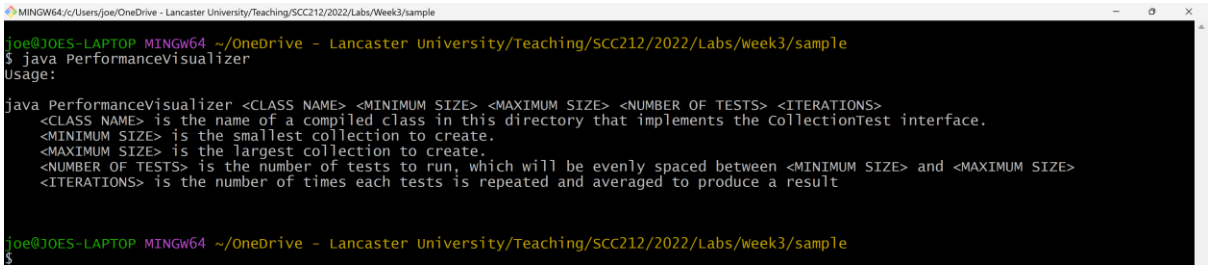
## Getting Started

- Start by creating your own class to perform the Java Collections tests. You can choose the name.
- Following the discussions in this week's lectures, write your class such that it is compliant with the CollectionTest interface.
- Ensure your class has a default constructor for your class (one that takes no parameters).
- You are only required to create collections that hold instances of the Person class. I would recommend storing these collections as instance variables in your class. Note that these collections should only be filled when instructed to do so.
- Implement the required methods such that they meet the descriptions provided in the API documentation of the CollectionTest interface. Pay particular attention to the required behaviour of the ADD, INDEX and SEARCH operations.
- When implementing this interface, you should pay particular attention to ensuring that you always use the **most efficient** way to achieve the task for each data structure. Give particular care when using the HashMap collection. However, ensure tests are fair. Do not create artificially fast or slow tests based on any assumptions.
- Note the iterations parameter and its intended behaviour.

## Task 2: Using the Performance Visualizer

Once you have written a solution that is at least partially complete (e.g. one data structure), you can run the PerformanceVisualizer to generate some graphs showing how well your collections are performing.

To run the PerformanceVisualizer, simply compile it using javac. Then run it on the command line to discover how to use it. Note that it requires a number of parameters on the command line to control its behaviour:




Each time PerformanceVisualizer is run, it will produce a HTML file called results.html showing graphs of how long it took to complete the ADD, INDEX and SEARCH operations using your class. Experiment with different sizes of collection to get a deeper understanding of how these really perform in the real world and compare to the theory you would expect from SCC120.

Lancaster University

## Portfolio Contribution

As discussed in the introductory lecture, all practical work this term will contribute to your portfolio assessment.

This piece of work will carry marks for core functionality related to the accurate implementation of the given interface, and the effective use of the Collections API. Marks will be allocated via an automated test harness that will test whether or not your implementation creates an accurate, fair and optimal test as per the interface specification.