CSCI 2110- Data Structures and Algorithms
Laboratory No. 4
Week of October 9th, 2017

This lab will get you familiarized with the Unordered List Data Structure.

<u>Marking Scheme</u>
Each exercise carries 10 points. Your final score will be scaled down to a value out of 10.
Working code, Outputs included, Efficient, Good basic comments included: 10/10
No comments: subtract one point
Unnecessarily inefficient: subtract one point
No outputs: subtract two points
Code not working: subtract up to six points depending upon how many methods are incorrect.

***<u>Error checking</u>**: Unless otherwise specified, you may assume that the user enters the correct data types and the correct number of input entries, that is, you need not check for errors on input.*

***<u>Submission</u>**: All submissions are through Brightspace. Log on dal.ca/brightspace using your Dal NetId. Submissions are pretty straightforward. Instructions will be also be given in the first lab.*
Deadline for submission: Sunday, October 15th, 2017 at 11.55 p.m. (five minutes before midnight).

***<u>What to submit</u>**:*
A zip file containing all source codes and a text document containing sample outputs.

Exercise 0 (no marks): Download the following files (given next to the Lab 4 link):
Node.java (Generic Node Class)
LinkedList.java (Generic Linked List Class)
List.java (Generic Unordered List Class)
Expense.java
ExpenseList.java (sample application that uses the Unordered List Class)
ExpenseListDemo.java
sepexpenses..txt (Sample text file for input to ExpenseListDemo.java)

Study these codes, run the ExpenseListDemo.java program and check the results. You will find these useful for solving the remaining exercises.

Exercise 1: Develop an application (similar to the ExpenseList class) that can store and manage information about students taking courses at a university.
First create a Student class that can hold the following pieces of information:
Student ID First Name Last Name Email Major Faculty
Add appropriate methods to this class.

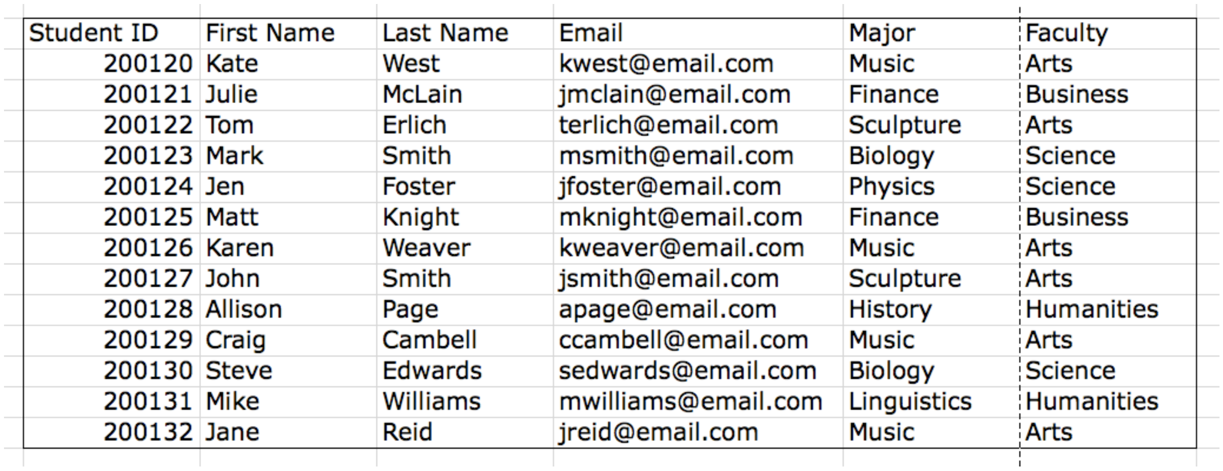
Next create a StudentList class that has as its attribute an unordered list of student records, and the following methods:

- Add a student record to the list
- Delete a student record from the list
- Display records of all the students taking a specific major
- Display records of all students belonging to a particular faculty
- Search for a student's record given an ID number

- Search for a student's record given the last and first names
- Search for a student's record given the email address

You may add other methods as necessary.

Next, create a client demo program that reads a file such as the one given below, creates an unordered list, and demonstrates all the methods that you have developed.



Student ID	First Name	Last Name	Email	Major	Faculty
200120	Kate	West	kwest@email.com	Music	Arts
200121	Julie	McLain	jmclain@email.com	Finance	Business
200122	Tom	Erlich	terlich@email.com	Sculpture	Arts
200123	Mark	Smith	msmith@email.com	Biology	Science
200124	Jen	Foster	jfoster@email.com	Physics	Science
200125	Matt	Knight	mknight@email.com	Finance	Business
200126	Karen	Weaver	kweaver@email.com	Music	Arts
200127	John	Smith	jsmith@email.com	Sculpture	Arts
200128	Allison	Page	apage@email.com	History	Humanities
200129	Craig	Cambell	ccambell@email.com	Music	Arts
200130	Steve	Edwards	sedwards@email.com	Biology	Science
200131	Mike	Williams	mwilliams@email.com	Linguistics	Humanities
200132	Jane	Reid	jreid@email.com	Music	Arts

Source for the above sample fake database:
http://wiki.ucalgary.ca/page/Courses/Computer_Science/CPSC_203/CPSC_203_Template/Labs_Template/TA_Examples_for_Access

NOTE: Just one sample output which shows the results for all the methods that you have developed is sufficient.

Exercise 2: Design a class called ListUtility.java that contains a list of static methods that manipulate unordered lists.

```
public class ListUtility{

      public static Lis<T> findUnion(List<T> list1, List<T> list2){
            //Create and return a third list that contains
            //the items that are either in list1 or in list2 or both.
            //Do not add duplicates.
      }
      public static List<T> findIntersection(List<T> list1, List<T> list2){
            //Create and return a third list that contains the items
            //that are common to both list1 and list2.
            //Do not add duplicates.
      }
      public static List<T> interleave(List<T> list1, List<T> list2){
            //Create and return a third list that contains the items
            //in list1 interleaved with the items in list2.
            //For example, list1={A, C}, list2={B, P, M, N, Z}
            //list3 = {A, B, C, P, M, N, Z}
      }
      public static List<T> chopSkip(List<T> list1){
            //Create and return a list that has the items in list1
            //with every second item removed.
            //For example, if list1={A, B, C, D, E}
            //the list returned is {A, C, E}
      }

}
```

Write a test program that creates two unordered lists from input data supplied by the user (you may assume that each list contains just Strings such as “A”, “C”, etc.) Make sure you test all the methods for at least three cases.

Exercise 3: In this exercise, you will be developing a geography quiz that tests kids on their knowledge of countries and their capitals. Download the CountriesCapitals.txt file given next to the lab link. This file has the country name on one line followed by its capital on the next:

```
Afghanistan
Kabul
Albania
Tirana
Etc.
```

Store the Country-Capital pairs as items in an unordered list. Your application must randomly select a country or a city and ask a question that matches a country to a capital or a capital to a country. It should then search the unordered list and report whether that answer is correct or incorrect. Here’s a sample screen dialog:

```
Welcome to the Country-Capital Quiz
Play? Yes
What is the capital of Canada?
Ottawa
Correct. Play? Yes
What country has Vienna as its capital?
Austria
Correct. Play? Yes
What is the capital of Albania?
Algiers
Incorrect. The correct answer is Tirana. Play? No
Game over.
```

NOTE: Only one output that shows a dialogue similar to the above is sufficient.