CS 211, ALL SECTIONS
LAB EXERCISE-12
DUE MONDAY, APRIL 15TH AT MIDNIGHT

The objective of this lab is to exercise using built in Java interfaces related to lists and collections.

OVERVIEW:

1. Write the class `ListUtils`. The class will contain three static methods, each used for converting between types of collections and related structures.
2. Download and use a tester module to ensure that your program is correct.
3. Prepare the assignment for submission and submit it.

This is a lab exercise on which collaboration (including discussing the solution on Piazza or searching online sources) is explicitly permitted. It is still in your interest to understand and prepare your final submission yourself.

BACKGROUND:

Java contains several different types of interfaces used to organize collections of items. You may already be familiar with the `List` interface, which is implemented by classes such as the `ArrayList`. A `List` represents a collection of elements from which elements can be stored or retreived, in which elements are ordered and can be retrieved by index. A `List` extends from a `Collection`, which represents a collection of elements but which may or may not have an order or element index. A `Set` or `Queue` is an example of a `Collection` which is not necessarily a `List`. Going further, a `Collection` is an `Iterable`, a structure which represents a sequence of elements. An `Iterable` may have persistent elements (such as with a `Collection`) or it may not (such as a data stream).

In this lab, we will write some utility methods within a class called `ListUtils`. These three methods will all be `static` methods, and their purpose will be to perform conversions between objects implementing one type of interface and objects implementing another type. For example, we will write a method for converting from an `Iterable` object to a `Collection`, and so on. We will discuss more about how this will work below.

Tip: you will definitely want to import `java.util` for this lab, since `List` and all of its relatives use it.

ITERABLE TO COLLECTION TASK:

```
public static <E> Collection<E> iterToCollection(Iterable<? extends E> iterable)
```

(*3pts*) Every `Collection` is an `Iterable` but not every `Iterable` is a `Collection`. However, if we have an `Iterable`, it implies a sequence of elements, so we can use that to built a `Collection`. That is what we will do in this method. To implement the method, we will need to create some kind of `Collection` object, and add elements from the `Iterable`'s sequence to it one by one.

Hint: we don't want to store the result in a `Collection` itself, because it is just an interface, but there may be a more familiar choice of structure which implements `Collection`.

COLLECTION TO LIST TASK:

```
public static <E> List<E> collToList(Collection<? extends E> coll)
```

(*3pts*) As above, every `List` is a `Collection` but not every `Collection` is a `List`. However, if we have a `Collection`, it implies a fixed number of elements which can be retrieved in sequence, so we can use that to built a `List` by numbering the element indices in the order they are retrieved. That is what we will do in this method. To implement the method, we will need to create some kind of `List` object, and add elements from the `Collection`.

LIST TO MAP TASK:

```
public static <E> Map<Integer, E> listToMap(List<? extends E> list)
```

(*3pts*) A `Map`, sometimes known as an associative array or dictionary, is in some ways similar to a `List`, except that instead of accessing elements by the indices 0, 1, 2, etc., the elements are accessed by a key, which may or may not be in order, and which may or may not be a number. Every element in a `Map` is stored as a key-value pair (the value of the element plus the key which is used to access it). Thus, when we call the `get()` method, instead of passing in an `int` index, we pass in the key corresponding to the element we're looking for. An example of a class which implements the `Map` interface in Java is the `HashMap` class.

A `List` is not a `Map` and a `Map` is not a `List`. However a `Map` stores collections of data indexed by some kind of mapping. A `Map` is essentially a dictionary which allows us to look up elements in a collection of data using some reference key to find each element. The reference key for a `Map` can be any data type, but if we make that data type an `Integer`, then we can use the key to represent a `List` index. Thus, in this method, we will take a `List` and use it to build a `Map` by using the list index of each element as the element's key within the `Map`. Thus, if our

`List` contains the elements `2`, `4`, and `6`, then this method will produce a `Map` with the mappings `0 ⇒ 2`, `1 ⇒ 4`, and `2 ⇒ 6`. As above, this would involve creating an appropriate type of `Map` and adding the elements from the `List`.

Testing:

Download the following:

- https://mason.gmu.edu/~iavramo2/classes/cs211/junit-cs211.jar (if necessary)
- https://mason.gmu.edu/~iavramo2/classes/cs211/s19/E12tester.java

This is a unit tester which is what we will use to test to see if your classes are working correctly.

When you think your classes are ready (you can try them even without the tester), do the following from the command prompt to run the tests.
On Windows:

```
javac -cp .;junit-cs211.jar *.java
java -cp .;junit-cs211.jar E12tester
```

On Mac or Linux:

```
javac -cp .:junit-cs211.jar *.java
java -cp .:junit-cs211.jar E12tester
```

In Dr Java:

- open the files, including `E12tester.java`, and click the *Test* button.

Submission:

Submission instructions are as follows.

1. Let *xxx* be your lab section number and let *yyyyyyyy* be your GMU userid. Create the directory `xxx_yyyyyyyy_E12/`
2. Place the `.java` file that you've created into the directory.
3. Create the file `ID.txt` in the format shown below, containing your name, userid, G#, lecture section and lab section, and add it to the directory.

   Full Name: Donald Knuth
   userID: dknuth
   G#: 00123456
   Lecture section: 004
   Lab section: 213

4. compress the folder and its contents into a .zip file, and upload the file to Blackboard.