Computer Laboratory 9
CSci 1913: Introduction to Algorithms, Data Structures, and Program Development
March 27/28, 2017

0. Introduction.

This assignment involves re-implementing the `Map` class from laboratory #7, but using a linear, singly-linked list instead of an array. Such a list is sometimes called an *association list*. Association lists are used in the AI programming language *Lisp,* and in some other languages that are based on Lisp.

1. Theory.

An *association list* is a linear, singly-linked list of nodes. Each node in an association list has three slots, called `key`, `value`, and `next`. The `key` and `value` slots point to objects of specific types. The `next` slot points to the next node in the list. Each node associates its `key` object with its `value` object. Association lists have the following operations; they involve traversing nodes, searching for one with a given `key` object.

- You can get the `value` object from a node. It's the object that is associated with the node's `key` object.

- You can change the `value` object in a node. After the change, the `key` object is associated with a different `value` object.

- If you cannot find a node, then you can add a new one that contains a `key` object and a `value` object. This establishes a new association between a `key` object and a `value` object.

As a result of these operations, you can use an association list something like a dictionary. For example, the `key` objects might be `String`'s that are English words for numbers. The `value` objects might be `Integer`'s that correspond to those words. If you give the association list an English word, then you can get back its corresponding number.

Association lists work by performing a kind of linear search. As a result, if an association list has n keys, then each of the operations described above will require $O(n)$ comparisons. Later in this course, we'll discuss more efficient alternatives to association lists. These will need only $O(\log n)$ or even $O(1)$ comparisons.

2. Implementation.

You must write a Java class called `AssociationList` that implements an association list. To simplify grading, your class must use the same names for things that are used here. It must have two class parameters, called `Key` and `Value`, so it looks like this.

```
class AssociationList<Key, Value>
{
   ⋮
}
```

Here `Key` is the type of the association list's `key` objects, and `Value` is the type of the association list's `value` objects.

Within the class `AssociationList`, you must have a `private` class called `Node`. The class `Node` must have three `private` slots: a slot called `key` whose type is `Key`, a slot called `value` whose type is `Value`, and a slot called `next` whose type is `Node`. It must have a constructor that initializes these three slots from its arguments. Don't try to use the node classes from the lectures, or from other assignments. They have the wrong number of slots, and the wrong types of slots. *You are not allowed to use arrays in any way.* If you implement `AssociationList` using one or more arrays, then you will receive zero points for this lab.

Your class must also have a `private` variable called `first`. It must point to the first `Node` in a linear singly-linked list of Node's. Along with Node and first, your class must have these methods. All of them work with first somehow.

```
public AssociationList()
```

Constructor. Initialize the AssociationList so it is empty.

```
public Value get(Key key)
```

Search the association list for a Node whose key slot equals the key parameter. Return the value slot of that Node. If no Node has a key slot that equals the key parameter, then throw an IllegalArgumentException.

```
private boolean isEqual(Key leftKey, Key rightKey)
```

Test if leftKey is equal to rightKey. Either or both may be null. This method is necessary because you must use == when leftKey or rightKey are null, but you must use the equals method when both are not null. (Recall that null has no methods.)

```
public boolean isIn(Key key)
```

Search the association list for a Node whose key slot equals the key parameter. (This can be done using isEqual.) Return true if you find such a Node, and return false otherwise.

```
public void put(Key key, Value value)
```

Search the association list for a Node whose key slot equals the key parameter. (This can be done using isEqual.) Change the value slot of that Node to be the value parameter. If there is no such Node, then add a new Node to the front of the association list. The new node's key slot is the key parameter, and its value slot is the value parameter.

The file **tests.java** on Moodle contains Java code that performs a series of tests. Each test calls a method from your class AssociationList, and prints what the method returns. Each test is also followed by a comment that tells how many points it is worth, and what must be printed if it works correctly.

3. Deliverables.

Run the tests, then turn in the Java source code for the class AssociationList. Your lab TA will tell you how and where to turn it in. If your lab is on **March 28, 2017**, then your work must be turned in by **11:55 PM** on **Tuesday, April 4, 2017.** If your lab is on **March 29, 2017**, then your work must be turned in by **11:55 PM** on **Tuesday, April 5, 2017.** To avoid late penalties, do not confuse these two dates.