

## HW 12: Hash Tables

You must create a `HashTable` object, that is able to store many arbitrary values, indexed by an arbitrary key type. Like the `UnorderedTable`, this will use two generics: `K` and `V`.

Recall that a hash table is just an array of sequential-search symbol tables. The easiest way to do this is to create a nested class called `Node` or `UnorderedTable`, that recycles your code from this week's lab. Then, your `HashTable` will have one field, which would ideally be an array of `Nodes`. However, remember that Java has severe problems with making an array of objects that take a generic type. This assignment will be much easier if you just use an `ArrayList`. Make sure that your `ArrayList` is only allocated once, and that it never has to resize itself.

Your `HashTable` class must have the following public methods (which should run in at most  $O(\alpha)$  time, where  $\alpha$  is the load factor):

- `HashTable(int tableSize)`, which is the constructor. It makes a brand new, empty `HashTable`, with an internal array or array list of `tableSize` `Nodes`.
- `void clear()`, which deletes all key-value pairs from the `HashTable`.
- `boolean delete(K key)`, which removes a key and its value. If the key isn't in the `HashTable`, do nothing and return `false`. If it is, delete it and return `true`.
- `V get(K key)`, which returns the value to which the key maps. If the key is not contained in the `HashTable`, return `null`.
- `int getSize()`, which returns the total number of key-value pairs there are in the `HashTable`. This should be constant in time.
- `void put(K key, V value)`, which puts the given key-value pair into the `HashTable`. If key already mapped to something else, it should replace the old value.
- A unit-testing `main()` method to test all the above methods, in a variety of situations.

You may of course have other private methods, if you wish. For example, it may be useful to have a private `calcIndex()` function that takes a key, and returns the proper hash code (modded to the table size and made positive).

In addition to the `HashTable` class, you must make a program called "*Animalpedia*". The job of *Animalpedia* is to load a tab-separated (`.tsv`) file that contains three columns:

1. The name of an animal.
2. The type of the animal (e.g. mammal, arthropod, etc.)
3. The number of feet the animal has.

It will then create two different `HashTables`: one that indexes the type of the animal by its name, and one that indexes the number of feet by the name. Each may have a table size of 101 (which is a prime number). Once the two tables are loaded, it will answer queries from the user. It will ask for an animal. If the user enters an animal it recognizes, the program

