DATA STRUCTURE EXERCISE DOCUMENTATION

The OpenAddrHashMap class is a hash map that uses open addressing and universal hashing to store key-value pairs. It implements the Dictionary interface, which defines methods for storing and retrieving key-value pairs from a data structure.

The OpenAddrHashMap class has the following fields:

- INITIAL_CAPACITY: a final static int that hold the default capacity of the array
- size: an int that holds the number of entries in the hash map.
- array: an array of Entry objects that stores the entries in the hash map.
- hashFunction: a UniversalHashingFunction object that is used to hash keys to find their positions in the array.

It has the following methods:

- A constructor that creates a new OpenAddrHashMap object with INITIAL_CAPACITY and a new UniversalHashingFunction object.
- A private constructor that creates a new OpenAddrHashMap object with the specified length and a new UniversalHashingFunction object.
- void put(K key, V value): stores a key-value pair in the hash map. This method first calls the rehashIfNeeded() method to ensure that the array has enough capacity to store the new key-value pair, and then calls the insert() method to store the key-value pair in the array.
- V remove(K key): removes a key-value pair from the hash map if it exists. This method first calls the rehashIfNeeded() method and then uses the hash function to find the position of the key in the array. It stores the value of the key-value pair in a variable, sets the corresponding entry in the array to null, and decrements the size field. It then checks the elements next to the deleted one and relocates them if needed.
- void rehashIfNeeded(): checks if the array needs to be resized and if it does, creates a new array with double or half the capacity of the old array and rehashes all the key-value pairs from the old array into the new array.
- void insert(K key, V value): inserts a key-value pair into the array. This method uses the hash function to find the position of the key in the array and stores the key-value pair there. If the position is already occupied, it moves to the next position in the array until it finds an empty position. It also checks if the key already exist and if it does it overwrites the entry with the new value without incrementing the size.
- V get(K key): searches for the given key and returns the value of the entry if it’s found else returns null.

- boolean contains(K key): searches for the given key and returns true if it's found else returns false.
- int size(): returns the size
- boolean isEmpty(): returns true if size = 0 else return false.
- void clear(): assigns a new array with the default capacity for the hashMap and a new hashFunction and thus deletets all current elements stored
- Iterator<Entry<K, V>> iterator(): return a new HashMapIterator Object made exclusively for this hashMap

The OpenAddrHashMap class also has the following inner classes:

- The EntryImpl class is a private inner class that implements the Entry interface. It has two fields for the key and value and a constructor that takes the key and value as arguments and stores them in the fields. It has getKey() and getValue() methods that return the key and value, respectively.
- The HashMapIterator class implenting the iterator interface on the hashMap. It includes a hasNext() method that returns a boolean indicating whether there are more elements in the collection, and a next() method that returns the next element in the collection.

The UniversalHashingFunction class is a Java class that provides a method for hashing objects in a hash map.

The UniversalHashingFunction class has the following fields:

- **`DEFAULT_SEED`**: a public constant that can be used as a default seed for the random number generator.
- **`DEFAULT_INPUT_BITS`**: a private constant that is used as the default width of the input array.
- **`EXCLUSIVE_UPPER_BOUND`**: a private constant that is used as the exclusive upper bound for the random number generator.
- **`arrayFunction`**: an array of integers that is used to store the array used for the actual hashing.
- **`random`**: an instance of the **`Random`** class, used to generate random numbers.
- **`b`**: a private field that stores the length of the **`arrayFunction`** array.
- **`u`**: a private field that stores the width of the **`arrayFunction`** array.

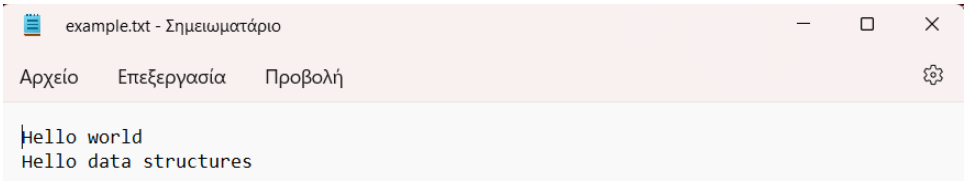
It has the following methods:

- **`UniversalHashingFunction(int b)`**: a constructor that creates an instance of the **`UniversalHashingFunction`** class with a randomly generated array for hashing.

- **`UniversalHashingFunction(int b, int seed)`**: a constructor that creates an instance of the **`UniversalHashingFunction`** class with a specific seed for the random number generator, allowing for the same sequence of numbers to be generated each time.
- **`hash(int hashcode)`**: a method that takes in a hashcode as an input and returns the index at which the object should be placed in the hash map. This method works by creating an input array based on the given hashcode, and then using this input array and the **`arrayFunction`** to calculate the index.

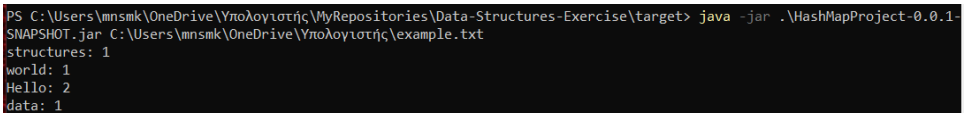
DATA STRUCTURE USAGE EXAMPLE

Given the following text file:



Correct usage:

An example of executing the .jar file which takes exactly one command line argument (path for the file to count frequency of words).



Wrong usage:

An example of executing the .jar file without any command line argument (path for the file to count frequency of words).