Github repository: <https://github.com/Dospinescu-Rares/FLCD-Project>

FLCD week 3 assignment

Dospinescu Rares

Group 932/1

Informatics in English Specialization

***public class SymbolTable***

The SymbolTable class provides a simple interface to manage symbols (identifiers and constants) using a custom hash table implementation. It allows inserting, looking up, checking existence, and removing symbols based on their names. Symbols are stored as key-value pairs, where the key is the symbol name (a String) and the value can be any Object.

**public void insert(String name, Object value)**

Inserts a symbol into the symbol table with the given name and value

* **name**: the name of the symbol (identifier or constant)
* **value**: the value associated with the symbol

**public Object lookup(String name)**

Looks up and retrieves the value associated with the given symbol name

* **name**: the name of the symbol to be looked up
* **returns** the value associated with the symbol, or null if not found

**public boolean contains(String name)**

Checks if the symbol table contains a symbol with the given name

* **name**: the name of the symbol to be checked
* **returns** true if the symbol is in the table, false otherwise

**public void remove(String name)**

Removes the symbol with the given name from the symbol table

* **name**: the name of the symbol to be removed

**public void display()**

Displays the contents of the symbol table. Symbols are displayed as key-value pairs.

**public class MyHashTable<K, V>**

Custom implementation of a hash table data structure. This hash table allows storing key-value pairs and provides basic operations like put, get, containsKey, remove, and keySet.

* **<K>:** the type of keys stored in the hash table
* **<V>:** the type of values associated with the keys

**private static class Entry<K, V>**

Inner class representing key-value pairs stored in the hash table.

* **<K>:** the type of keys
* **<V>:** the type of values

**private int getHash(K key)**

Private helper method to calculate the hash value for a given key.

* **key:** the key for which the hash value is calculated
* **returns** the hash value

**public void put(K key, V value)**

Inserts a key-value pair into the hash table. If the key already exists, the associated value is updated.

* **key:** the key to be inserted or updated
* **value:** the value associated with the key

**public V get(K key)**

Retrieves the value associated with the given key.

* **key:** the key for which the value is retrieved
* **returns** the value associated with the key, or null if key not found

**public boolean containsKey(K key)**

Checks if the hash table contains the given key.

* **key:** the key to be checked for existence
* **returns** true if the key is found, false otherwise

**public void remove(K key)**

Removes the key-value pair with the given key from the hash table.

* **key:** the key of the pair to be removed

**public Set<K> keySet()**

Returns a set of all keys stored in the hash table.

* **returns** a set containing all keys in the hash table

**public int size()**

Returns the number of key-value pairs in the hash table.

* **returns** the number of key-value pairs in the hash table