EEM 480 Homework 4

Yunus Nihat Özpolat

public class Hashmapint<K, V>{

Entry<K, V>{

Entry<K, V> is a nested class within the **Hashmapint<K, V>** class that represents a key-value pair in the hash map. It has the following fields:

- **key**: a key of type **K**
- **value**: a value of type **V**
- **next**: a reference to the next **Entry** in the linked list, or **null** if this is the last entry

It also has getter and setter methods for each of these fields. The **Entry** class is used to implement a linked list of key-value pairs in the hash map, with each entry in the list representing a single key-value pair. This is used to handle collisions, where multiple keys hash to the same index in the array.

}

Hashmapint() {

This is the default constructor for the **Hashmapint** class. It creates a new hash map with a default capacity of 16.

}

Hashmapint(int capacity) {

This is an alternate constructor for the **Hashmapint** class that allows the caller to specify the initial capacity of the hash map.

}

put(K key, V value) {

This method inserts a key-value pair into the hash map. It calculates the index in the array where the entry should be stored based on the key's hash code, and then either adds the entry to the beginning of the linked list at that index or updates the value of an existing entry with the same key.

}

get(K key) {

This method retrieves the value associated with a given key in the hash map. It calculates the index in the array where the entry should be stored based on the key's hash code and then searches the linked list at that index for the key. If the key is found, the associated value is returned. If the key is not found, **null** is returned.

}

remove(K key) {

This method removes a key-value pair from the hash map. It calculates the index in the array where the entry should be stored based on the key's hash code and then searches the linked list at that index for the key. If the key is found, the entry is removed from the linked list.

display() {

This method prints the contents of the hash map to the console. It iterates through each element in the array and, for each non-empty linked list, it prints the key and value of each entry in the list.

}

toString() {

This method returns a string representation of the hash map. It iterates through each element in the array and, for each non-empty linked list, it appends the key and value of each entry in the list to the string.

}

index(K key){

This method calculates the index in the array where a key-value pair should be stored based on the key's hash code. It handles the case where the key is **null** by returning 0. For non-null keys, it returns the absolute value of the key's hash code modulo the capacity of the hash map.

}

public static void main(String[] args) {

The **main** method reads input from a file and processes it using several data structures and classes. The input is in the form of commands, each starting with a letter that specifies the type of operation to perform. The **main** method reads each line of the input file and processes it according to the command specified at the start of the line.

The **main** method uses the following data structures and classes:

- **personMap**: a hash map that stores **Person** objects. The keys are integers and the values are **Person** objects.
- **songMap**: a hash map that stores **Song** objects. The keys are integers and the values are **Song** objects.
- **likes**: a hash map that stores strings representing the names of people and songs in the format "person_name song_name". The keys are integers and the values are strings.
- **personList**: a linked list that stores **Person** objects.
- **songList**: a linked list that stores **Song** objects.

- **ss**: a linked list that stores strings representing the names of people and songs in the format "person_name song_name".

The **main** method processes the input commands as follows:

- **I**: Insert a person into the program. The name of the person is specified after the **I**. The **main** method creates a new **Person** object with the specified name and adds it to the **personList** linked list. The **main** method also adds an entry to the **personMap** hash map, using the index of the **Person** object in the **personList** as the key and the **Person** object as the value.

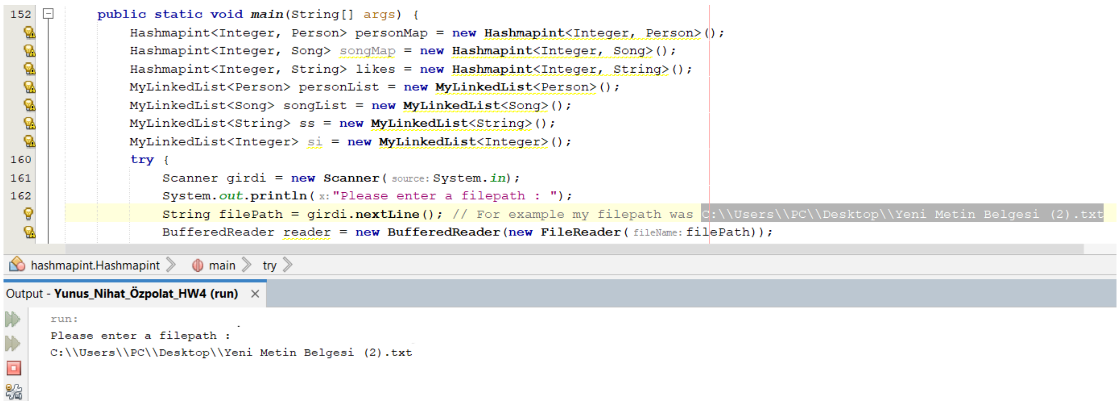
- **L**: Record that a person likes a song. The name of the person and the name of the song are specified after the **L**. The **main** method checks if the specified person is in the **personList** linked list. If the person is not in the list, the **main** method prints an error message. If the person is in the list, the **main** method adds an entry to the **likes** hash map, using a new integer as the key and a string representing

- **E**: Erase a person's liking of a song. The name of the person and the name of the song are specified after the E. The main method checks if the specified person is in the personList linked list and if the specified song is in the songList linked list. If either the person or the song is not in the corresponding list, the main method prints an error message. If both the person and the song are in their respective lists, the main method removes an entry from the likes hash map. It is not clear how the main method determines which entry to remove.
- **D**: Delete a person from the program. The name of the person is specified after the D. The main method searches the personList linked list for the specified person. If the person is not in the list, the main method prints an error message. If the person is in the list, the main method removes the person from the personList linked list and the personMap hash map. The main method also removes all entries from the likes hash map that contain the name of the deleted person.
- **P**: Lists the songs of the person name likes.
- **X**: Exit the program.

}

}

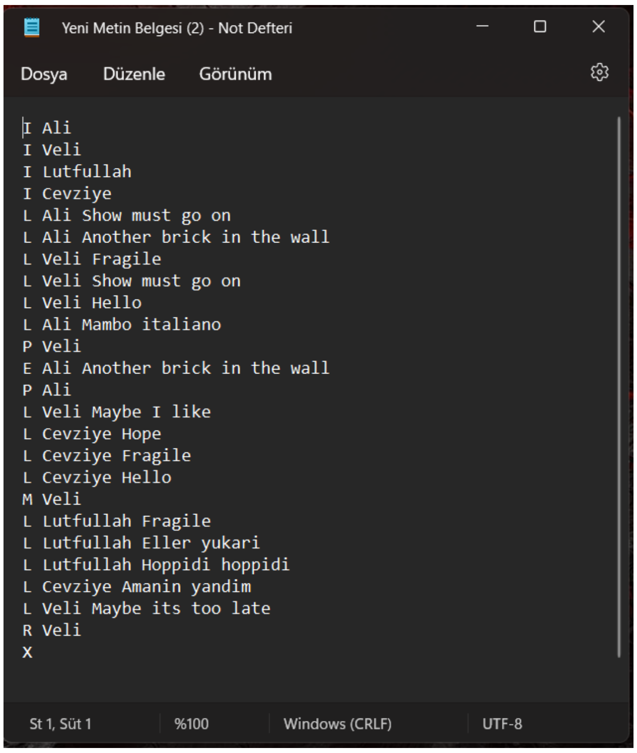
When we run the code, we have to write filepath of our .txt.



```
    public static void main(String[] args) {
        Hashmapint<Integer, Person> personMap = new Hashmapint<Integer, Person>();
        Hashmapint<Integer, Song> songMap = new Hashmapint<Integer, Song>();
        Hashmapint<Integer, String> likes = new Hashmapint<Integer, String>();
        MyLinkedList<Person> personList = new MyLinkedList<Person>();
        MyLinkedList<Song> songList = new MyLinkedList<Song>();
        MyLinkedList<String> ss = new MyLinkedList<String>();
        MyLinkedList<Integer> si = new MyLinkedList<Integer>();
        try {
            Scanner girdi = new Scanner(System.in);
            System.out.println("Please enter a filepath : ");
            String filePath = girdi.nextLine(); // For example my filepath was C:\\Users\\PC\\Desktop\\Yeni Metin Belgesi (2).txt
            BufferedReader reader = new BufferedReader(new FileReader(filePath));
```

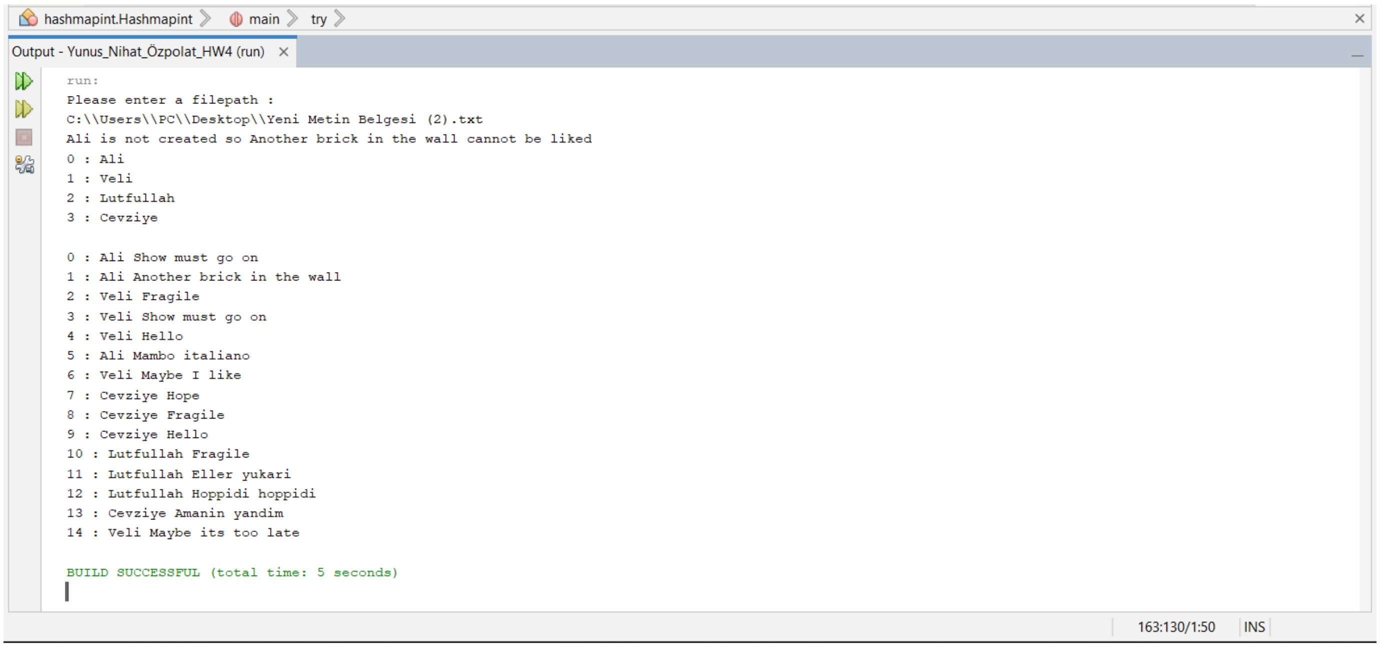
```
run:
Please enter a filepath :
C:\\Users\\PC\\Desktop\\Yeni Metin Belgesi (2).txt
```

This is my input txt:



```
I Ali
I Veli
I Lutfullah
I Cevziye
L Ali Show must go on
L Ali Another brick in the wall
L Veli Fragile
L Veli Show must go on
L Veli Hello
L Ali Mambo italiano
P Veli
E Ali Another brick in the wall
P Ali
L Veli Maybe I like
L Cevziye Hope
L Cevziye Fragile
L Cevziye Hello
M Veli
L Lutfullah Fragile
L Lutfullah Eller yukari
L Lutfullah Hoppidi hoppidi
L Cevziye Amanin yandim
L Veli Maybe its too late
R Veli
X
```

This one is output of my codes:



```
run:
Please enter a filepath :
C:\\Users\\PC\\Desktop\\Yeni Metin Belgesi (2).txt
Ali is not created so Another brick in the wall cannot be liked
0 : Ali
1 : Veli
2 : Lutfullah
3 : Cevziye

0 : Ali Show must go on
1 : Ali Another brick in the wall
2 : Veli Fragile
3 : Veli Show must go on
4 : Veli Hello
5 : Ali Mambo italiano
6 : Veli Maybe I like
7 : Cevziye Hope
8 : Cevziye Fragile
9 : Cevziye Hello
10 : Lutfullah Fragile
11 : Lutfullah Eller yukari
12 : Lutfullah Hoppidi hoppidi
13 : Cevziye Amanin yandim
14 : Veli Maybe its too late

BUILD SUCCESSFUL (total time: 5 seconds)
```