

1. HashTable Class Methods Explanations & Worst Case Running Time Complexity of Methods

- a. Inner private class: **Entry**
 - i. This inner private class has three fields key, value, and is used to internally implement linked-lists where keys with the same hash code are conjoined together. A single entry holds a key value pair.
- b. Insert method--public void insert(char key, int value)
 - i. This method inserts an entry element which holds a key value pair. It also checks to make sure that the key is not null.
 - ii. This method is a worst case time complexity of $O(n)$, where n represents the number of keys in the hash table.
- c. Get method-- public int get(char key)
 - i. This method gets the value linked with the specific key in the hashtable. If there is not a value the value null will be returned.
 - ii. This method is a worst case time complexity of $O(n)$, where n represents the number of keys in the hash table.
- d. Remove method--public void remove(char key)
 - i. This method removes the key and its linked value from the hash table. If this does not occur then not is done.
 - ii. This method is a worst case time complexity of $O(n)$, where n represents the number of keys in the hash table.
- e. Contains Key method--public boolean containsKey(char key)
 - i. This method returns true when the specific key has the associated value in the table. If the key is not in the table it will return false.
 - ii. This method is a worst case time complexity of $O(n)$, where n represents the number of keys in the hash table.
- f. Rehash method--private void reHash()
 - i. The nextPrime method is a helper method that is used in rehashing.
 - ii. The rehash method takes linear $O(N)$ time. This is because for separate chaining, we check every entry and every key in each entry once, so we can express the time complexity as $O(N + K)$, where N is the size of the

original table and K is the number of keys. We only rehash when the load factor surpasses a certain threshold (ideally < 1), so the number of keys $K = cN$ where c is a constant < 1 (e.g. 0.75). So, the overall time complexity of rehashing is $O(N + cN) = O(N)$.

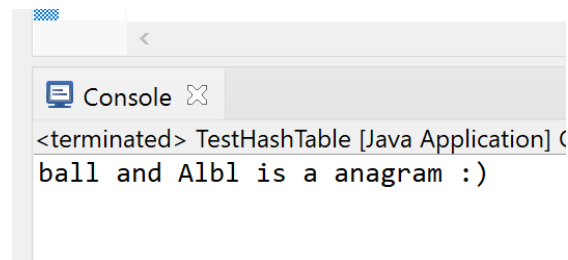
- g. Size method--public int size()- this method returns the number of key and value pairs in the hash table
 - i. This has a worst case time complexity of $O(1)$, since this method just returns the number of items (which is the number of key & value pairs).
 - h. Is Empty method--public boolean isEmpty()
 - i. This method has a worst case time complexity of $O(1)$, since it returns true if the hashTable contains no elements, and false otherwise.
 - i. Private hash method--private int hash(char key)- is a method that returns the hash value of the key using the given hash function. This is private since the user should not have access to the hash function data.
 - i. This has a worst case time complexity of $O(1)$, since this method just returns the hashed value of the key.
2. TestHashTable class & code input and outputs:
- a. The TestHashTable class was built to check if the HashTable class is implemented correctly by checking if two inputted strings (String X and String Y) are anagrams of each other or not.
 - i. The isAnagram method checks if two inputted strings are anagrams of each other or not using the HashTable class. It iterates through each the characters of both inputted strings and frequency value starts with 1 and if the character is already there it increments by 1 for the first String(String X). In the second String(String Y) it decrements the frequency value and if the value is not there it returns false. The time complexity of the isAnagram method is $O(n)$, where n represents the number of characters in the first inputted string (String X).
 - ii. The main method is where all the testing of the isAnagram method is done
 - b. First Test Case: Check if the algorithm is able to find that 2 strings is an Anagram given a mixture of uppercase and lowercase letters. The algorithm should ignore

uppercase and lower case letters and be able to find that the two string are a anagram of each other

- i. Inputting:

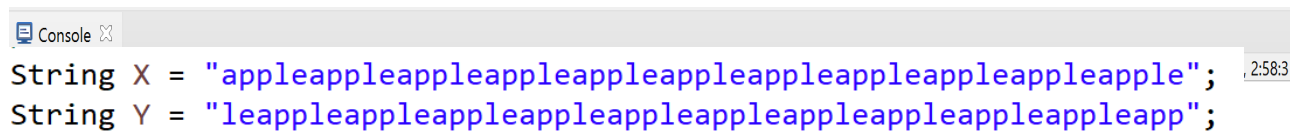
```
String X = "ball";  
String Y = "Alb1";
```

Output:

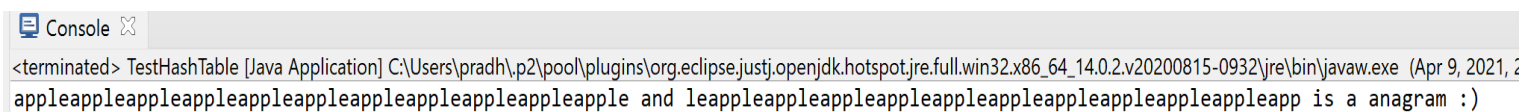


- ii. Check that the algorithm is able to find that two very long strings are anagrams of each other and able to return that it is a anagram

Inputting :



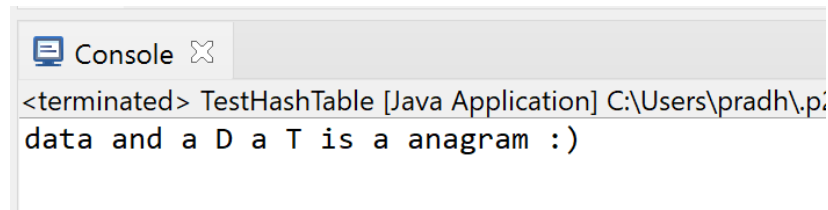
Output:



- iii. Check that the algorithm is able to take in two strings that are anagrams of each other but have spaces and a mixture of upper and lowercase letters and return that it is an anagram.

Inputting: `String X = "data";`
`String Y = "a D a T";`

Outputting:



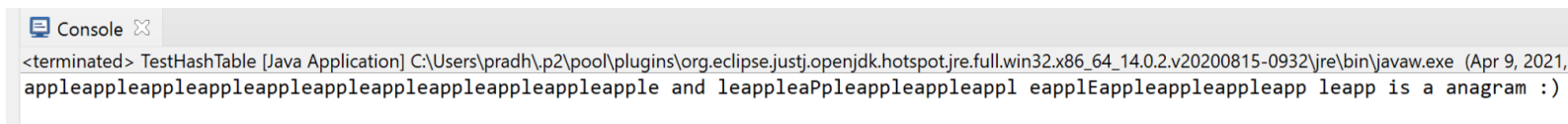
```
Console ✕
<terminated> TestHashTable [Java Application] C:\Users\pradh\p
data and a D a T is a anagram :)
```

- iv. Testing to see if the hashtable works on a very long string with spaces and upper case & lowercase variables and is able to identify if 2 given strings is an anagram.

Inputting: The E and P is uppercased in string Y and it has random spaces

```
String X = "appleappleappleappleappleappleappleappleappleapple";
String Y = "leappleaPpleappleappleappl eapplEappleappleappleapp leapp";
```

Outputting:



```
Console ✕
<terminated> TestHashTable [Java Application] C:\Users\pradh\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_14.0.2.v20200815-0932\jre\bin\javaw.exe (Apr 9, 2021,
appleappleappleappleappleappleappleappleappleappleapple and leappleaPpleappleappleappl eapplEappleappleappleapp leapp is a anagram :)
```

All of the examples above despite having upper case or spaces or a really long length, the algorithm is able to identify that it is an anagram.

Now testing for the fail condition:

- c. Second Test Case (fail condition): Check if the algorithm is able to find the 2 strings is NOT an anagram of each other
 - i. Testing that upper and lower case does not matter & able to identify if 2

given strings is NOT a anagram

Inputting:

```
String X = "balLB";  
String Y = "Albl";
```

Outputting:

```
Console X  
<terminated> TestHashTable [Java Application] C:\U  
balLB and Albl is NOT a Anagram :(
```

- ii. Testing to see if the hashtable works on a very long string & able to identify if 2 given strings is NOT a anagram

Inputting:

```
String X = "appleappleappleappleappleappleappleappleappleappleappleapple";  
String Y = "leappleappleappleappleappleappleappleappleappleappleappleappel";
```

Outputting:

```
Console X  
<terminated> TestHashTable [Java Application] C:\Users\pradh\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_14.0.2.v20200815-0932\jre\bin\javaw.exe (Apr 9, 2021, 10:09:20 PM - 10:  
appleappleappleappleappleappleappleappleappleappleappleapple and leappleappleappleappleappleappleappleappleappleappleappleappleappel is NOT a Anagram :(
```

- iii. Testing that spaces & upper and lower case do not matter & able to identify two given strings is NOT a anagram

Inputting:

```
String X = "data";  
String Y = "a D a T A";
```

Outputting:

```
Console X  
<terminated> TestHashTable [Java Application] C:\Users\pradh\  
data and a D a T A is NOT a Anagram :(
```

- iv. Checking to see if the hashtable works on a very long string with spaces and upper and lowercase & it able to identify if 2 given strings is NOT a anagram

Inputting:

```
//  
String X = "appleappleappleappleappleappleappleappleappleappleappleapple";  
String Y = "leappleaPpleappleappleapl eapplEappleappleapple leapp P";  
//
```

Outputting:

Console

```
<terminated> TestHashTable [Java Application] C:\Users\pradh\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_14.0.2.v20200815-0932\jre\bin\javaw.exe (Apr 9, 2021, 10:25:50 PM)  
appleappleappleappleappleappleappleappleappleappleappleapple and leappleaPpleappleappleapl eapplEappleappleapple leapp P is NOT a Anagram :(
```

- d. Third Test Case(bad input data): Check if the algorithm does not break when passed bad data, such as a null String.

Inputting:

```
//  
String X = null;  
String Y = "null";
```

Outputting:

Console

```
<terminated> TestHashTable [Java Application] C:\Users\pradh\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_14.0.2.v20200815-0932\jre\bin\javaw.exe (Apr 9, 2021, 10:25:50 PM)  
Please handle the error: Cannot invoke "String.replaceAll(String, String)" because "X" is null
```