Lab 5

CSE 274

Import the `Application.java` file in Eclipse, run the application and make sure that you receive no error.

I. Backward-Shifting

Erase the body of the `main` method, copy the following lines of code into the body of the `main` method, run the application and make sure that you see the expected output:

```
HashTable myhashtable = new HashTable(52);
myhashtable.ProbeANDinsertNotForever("FL");
myhashtable.ProbeANDinsertNotForever("CO");
myhashtable.ProbeANDinsertNotForever("ME");
myhashtable.ProbeANDinsertNotForever("MI");
myhashtable.ProbeANDinsertNotForever("SC");
myhashtable.ProbeANDinsertNotForever("NH");
myhashtable.displayTable();
```

The expected output is printed below:

```
** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** FL CO ME ** MI SC
   NH ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** **
   ** ** ** ** ** ** ** ** **
```

As it can be seen from the above output, two blocks of data exists in the `InternalArray`. Inserting `ND` in the `InternalArray` causes these blocks of data to merge. Erase the body of the `main` method, copy the following lines of code into the body of the `main` method, run the application and make sure that you see the expected output:

```
HashTable myhashtable = new HashTable(52);
myhashtable.ProbeANDinsertNotForever("FL");
myhashtable.ProbeANDinsertNotForever("CO");
myhashtable.ProbeANDinsertNotForever("ME");
myhashtable.ProbeANDinsertNotForever("MI");
myhashtable.ProbeANDinsertNotForever("SC");
myhashtable.ProbeANDinsertNotForever("NH");

myhashtable.ProbeANDinsertNotForever("ND");

myhashtable.displayTable();
```

The expected output is printed below:

```
** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** FL CO ME ND MI SC
   NH ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** **
   ** ** ** ** ** ** ** ** **
```

Now, lets delete ME from the InternalArray. Erase the body of the main method, copy the following lines of code into the body of the main method, run the application and make sure that you see the expected output:

```
HashTable myhashtable = new HashTable(52);
myhashtable.ProbeANDinsertNotForever("FL");
myhashtable.ProbeANDinsertNotForever("CO");
myhashtable.ProbeANDinsertNotForever("ME");
myhashtable.ProbeANDinsertNotForever("MI");
myhashtable.ProbeANDinsertNotForever("SC");
myhashtable.ProbeANDinsertNotForever("NH");

myhashtable.ProbeANDinsertNotForever("ND");

myhashtable.ProbeANDdeleteButNoGapNotForever("ME");

myhashtable.displayTable();
```

The expected output is printed below:

```
** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** FL CO ND MI SC NH
   ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** **
   ** ** ** ** ** ** ** ** **
```

As it can be seen above, ME is successfully deleted from the InternalArray. Now, lets delete MI from the InternalArray. Erase the body of the main method, copy the following lines of code into the body of the main method, run the application and make sure that you see the expected output:

```
HashTable myhashtable = new HashTable(52);
myhashtable.ProbeANDinsertNotForever("FL");
myhashtable.ProbeANDinsertNotForever("CO");
myhashtable.ProbeANDinsertNotForever("ME");
myhashtable.ProbeANDinsertNotForever("MI");
myhashtable.ProbeANDinsertNotForever("SC");
myhashtable.ProbeANDinsertNotForever("NH");

myhashtable.ProbeANDinsertNotForever("ND");

myhashtable.ProbeANDdeleteButNoGapNotForever("ME");

myhashtable.ProbeANDdeleteButNoGapNotForever("MI");


myhashtable.displayTable();
```

The expected output is printed below:

```
** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** FL CO ND MI SC NH
   ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** **
   ** ** ** ** ** ** ** ** **
```

As it can be seen above, the ProbeANDdeleteButNoGapNotForever method is unable to delete MI, and therefore is buggy. Lets see what is the reason behind having this bug.

Previously, we learned that a delete method should not create gaps in the InternalArray, as a gap results in bugs. We used a specific mechanism to prevent creation of gaps. Namely, for deleting a specific key from the InternalArray we overwrite the key with the next entry in the InternayArray and keep shifting all entries of the corresponding block of data one index backward. For instance, let say that the InternalArray has the below arrangement:

⋮	
**	15
FL	16
CO	17
ME	18
ND	19
MI	20
SC	21
NH	22
**	23
⋮	

Deleting ME using the backward-shifting mechanism results in the following arrangement in the InternalArray:

⋮	
**	15
FL	16
CO	17
ND	18
MI	19
SC	20
NH	21
**	22
**	23
⋮	

Notice that, the backward-shifting mechanism is indeed effective in preventing creation of gaps in the above block of data.

Nevertheless, note that after completion of backward-shifting process MI is now placed at Index 19, which is one index behind the hashIndex corresponding to MI. More precisely, the hashIndex for MI is hashFunc("MI")=20. But now, MI is placed at Index=19 which is one index behind hashIndex=20. When we call the delete method to delete MI, the delete method calls hashFunc and obtains the hashIndex of 20 for MI and begins searching for MI starting at Index=20. Since MI is now at Index=19, the delete method is unable to delete MI.

From the above discussion, deletion with backward-shifting mechanism works fine, as long as blocks of data in the InternalArray do not merge. When using backward-shifting, one should always develop and use a DoesItMerge method to make sure that blocks of data are not merging.

In the continue, we develop new insert and delete methods that work perfectly fine even if blocks of data merge. We use a new mechanism for deleting keys from the InternalArray. This new mechanism is referred to as *no-shifting* mechanism, and can be used to delete keys from the InternalArray without creating gaps and without shifting keys.

II. No-Shifting

The deleteMinus method is capable of deleting keys without creating gaps and without shifting keys. For performing the actual deletion of a key, the deleteMinus replaces the key with -- characters. The code for deleteMinus is provided below:

```
public void deleteMinus(String key)
{
int hashIndex = hashFunc(key);
while (!InternalArray[hashIndex].equals("**"))
      {
      if (InternalArray[hashIndex].equals(key))
            {
            InternalArray[hashIndex]="--";
            return;
            }
      ++hashIndex;
      hashIndex= hashIndex % arraySize;
      }
return;
}
```

As it can be seen above, the deleteMinus method performs the following tasks:

- Receives a key as an argument.
- Calculates the hashIndex corresponding to the key and begins searching the InternalArray for key, starting at the said hashIndex.
- Once the key is found in the InternalArray, the deleteMinus method proceeds with actual deletion of the key and return. The method performs the actual deletion by replacing the key with -- characters.
- The method stops searching for key when reaching an entry ** in the InternalArray.

From above, -- characters indicate a deleted entry in the InternalArray. Obviously, the deleteMinus method successfully deletes the key without creating a gap and without shifting entries.

III. The insertMinus Method

We need to develop an insertMinus method that performs the following tasks:

- Receives a key as an argument.
- Calculates the hashIndex corresponding to the key.
- Begins searching for the key in the InternalArray starting at the said hashIndex. The method stops searching when finding the key or when reaching a ** entry.
- If the key was found in the InternalArray the method return.

- If the key was not found in the InternalArray the method performs actual insertion of the key in the InternalArray. The method begins searching for an entry with -- in the InternalArray, starting at the said hashIndex. Once such an entry with -- was found, the method performs the actual insertion by replacing -- characters with the key. If there was no entry with -- characters in the block of data, the method inserts the key into the ** entry at end of the block of data.

Develop the insertMinus method with the above features. Use the following logic/pseudocode for implementing insertMinus method:

```
int hashIndex = hashFunc(key)
while (InternalArray[hashIndex] is not equal to "**")
      if (InternalArray[hashIndex] is equal to key)
            return

      ++hashIndex
      hashIndex= hashIndex % arraySize
hashIndex = hashFunc(key)
while (InternalArray[hashIndex] is not equal to "**" and "--")
      ++hashIndex
      hashIndex= hashIndex % arraySize
InternalArray[hashIndex] = key
return
```

To test the developed method, erase the body of the main method, copy the following lines of code into the body of the main method, run the application and make sure that you see the expected output:

```
HashTable myhashtable = new HashTable(52);
myhashtable.insertMinus("FL");
myhashtable.insertMinus("CO");
myhashtable.insertMinus("ME");
myhashtable.insertMinus("MI");
myhashtable.insertMinus("SC");
myhashtable.insertMinus("NH");

myhashtable.insertMinus("ND");

myhashtable.deleteMinus("CO");
myhashtable.deleteMinus("MI");

myhashtable.displayTable();
```

The expected output is printed below:

```
** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** FL -- ME ND -- SC
   NH ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** **
   ** ** ** ** ** ** ** ** **
```

IV. Preventing Infinite Loops

To prevent infinite loops in insertMinus and deleteMinus methods, there is a need for at least one entry with ** in the InternalArray. As we have discussed before, num is the number of non ** entries in the InternalArray. Add new lines of code to the insertMinus method to make num keep track of the number of non ** in the InternalArray, and to prevent insertion into a ** entry

when there is only one ** in the InternalArray. Notice that, the deletMinus method does not need to be edited.
To test the methods, erase the body of the main method, copy the lines of code in the Test.txt file into the body of the main method, run the application and make sure that you see the expected output. The expected output is printed below:

```
VA WA CA WV WI DC GA DE IA ** AK AL ID LA MA HI CO AR FL IL ME CT
   IN MD MI AZ MN MO KS NE MS MT NH NJ KY NV NM NY NC ND OH OK OR
   PA PR RI SC SD TN TX UT VT
```

V. The findMinus Method

Develop the following method for the HashTable class:

```
public boolean findMinus(String key)
```

The above method receives a key, and return true only when the key is existing in the InternalArray. The method should search for the key only in the specific block of data corresponding to the key. To test the developed method, erase the body of the main method, copy the following lines of code into the body of the main method, run the application and make sure that you see the expected output:

```
HashTable myhashtable = new HashTable(52);
myhashtable.insertMinus("FL");
myhashtable.insertMinus("CO");
myhashtable.insertMinus("ME");
myhashtable.insertMinus("MI");
myhashtable.insertMinus("SC");
myhashtable.insertMinus("NH");

myhashtable.insertMinus("ND");

myhashtable.deleteMinus("CO");
myhashtable.deleteMinus("MI");

System.out.println(myhashtable.findMinus("CO"));
System.out.println(myhashtable.findMinus("MI"));
System.out.println(myhashtable.findMinus("ND"));
System.out.println(myhashtable.findMinus("ME"));
```

The expected output is printed below:

```
false
false
true
true
```

VI. Discussion

Whether using backward-shifting or no-shifting mechanism, we will have blocks of data in the InternalArray when the hashFunc is not ideal. The advantage of backward-shifting mechanism is that the length of blocks are shorter as there is no -- characters in the InternalArray. The disadvantage is that one needs to use DoesItMerge method before inserting any key into the InternalArray to make sure that blocks of data do not merge. In contrast, blocks of data are permitted to merge for the case of no-shifting mechanism. The disadvantage of no-shifting mechanism is that blocks of data could be longer as deleted keys are replaced by `--` characters. For the case of no-shifting mechanism, the blocks of data merge, but never split.

VII. Submitting the Assignment

Before submitting your response to this assignment, erase the body of the `main` method, copy the lines of code in the `Test.txt` file into the body of the `main` method, run the application and make sure that you receive no error.