Lab 7

CSE 274

A company has just started operating. As time goes by, the company might start doing business in different states of the country, or might stop doing business in some states. We use a Hash Map data structure to keep a record of the number of employees the company has in each state of the country that the company is doing business in. In this Lab, the keyword "states" refers to the states of the country.
A `Hashmap` class is developed in the `Application.java` file. The developed class is just a draft and needs to be completed to perform the full functionality of a Hash Map data structure. Import the `Application.java` file in Eclipse, run the application and make sure that you receive no error.

I. The Put Method

Below is the signature for `put` method:

```
public void put(String key, int value)
```

The `put` method receives two arguments, a `key` and a `value`, and should perform the following tasks:

- Calculate the `hashIndex` corresponding to the `key`.
- Begin searching for the `key` in the `internalArray`, starting at `hashIndex`. The search should be limited to only one block of data. That means, the search should stop when the `key` is found or when the search reaches to a `**` entry.
- If the `key` is found in the `InternalArray`, the method should `return`.
- If the `key` is not found in the `InternalArray`, the method should search for a place to insert the `key` in the `InternalArray`. Toward this goal, the method should calculate the `hashIndex` corresponding to the `key`, and begin searching for a `**` or `--` entry, starting at `hashIndex`. Once such an entry is found, the method should proceed with actual insertion of the `key` in the `InternalArray`.
- Let `hashIndex` denote the index at which the `key` is inserted in the `internalArray` by the method. The method should insert `value` into the `ValueArray` at `hashIndex`.
- The method should make `num` keep track of the number of non `**` entries in the `internalArray`, and prevent insertion into a `**` entry when there is only one `**` in the `InternalArray`.

Look into the code for `put` method. The developed code performs all the above tasks except for one. Find out which one of the above tasks is not performed by the method, and add new lines of code to the method to fix the bug.

II. The remove Method

The `remove` method has the following signature:

```
public void remove(String key)
```

The method receives a `key`, deletes the `key` from the `InternalArray`, and resets the corresponding entry in the `ValueArray` to -1.

III. The doMapping method

Develop the doMapping method with the following signature:

```
public int doMapping(String key)
```

The method receives a key, find the index of the key in the InternalArray and return the entry of ValueArray that is at the same index. The method return -1 if the key is not in the InternalArray. 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:

```
HashMap myhashmap = new HashMap(52);
myhashmap.put("AK",114);
myhashmap.put("FL",341);
myhashmap.put("IA",419);
myhashmap.put("MN",201);
myhashmap.put("NV",941);
myhashmap.put("VT",388);

myhashmap.remove("VT");
myhashmap.displayMap();

String key;

key="MN";
System.out.println("key:"+ key+" value:"+myhashmap.doMapping(key));

key="VT";
System.out.println("key:"+ key+" value:"+myhashmap.doMapping(key));
```

The expected output is printed below:

```
Table:
(AK: 114) (IA: 419) (MN: 201) (FL: 341) (**: -1) (**: -1) (**: -1)
   (**: -1) (**: -1) (**: -1) (**: -1) (**: -1) (**: -1) (NV: 941)
   (**: -1) (**: -1) (**: -1) (**: -1) (**: -1) (**: -1) (**: -1)
   (**: -1) (**: -1) (**: -1) (**: -1) (**: -1) (**: -1) (**: -1)
   (**: -1) (**: -1) (**: -1) (**: -1) (**: -1) (**: -1) (**: -1)
   (--: -1) (**: -1) (**: -1) (**: -1) (**: -1) (**: -1) (**: -1)
   (**: -1) (**: -1) (**: -1) (**: -1) (**: -1) (**: -1) (**: -1)
   (**: -1) (**: -1) (**: -1)
key:MN value:201
key:VT value:-1
```

IV. The HashMapPair Class

In this section, we develop a HashMapPair class which implements a Hash Map data structure, using only an InternalArray without using ValueArray. Fig. 1 shows the internal of the HashMapPair class:

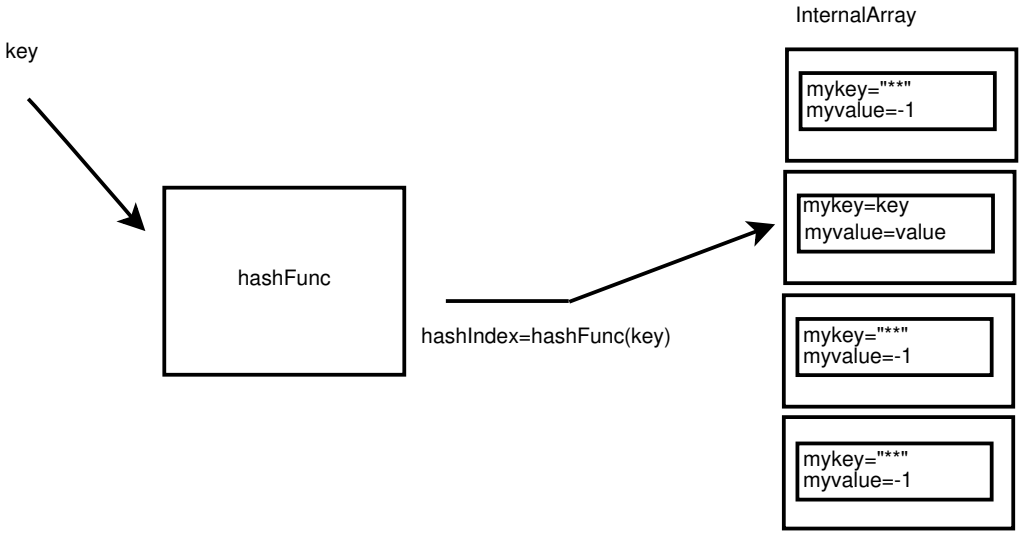

Fig. 1: Internal of the `HashMapPair` class.

The `HashMapPair` uses the following `Pair` class:

```
class Pair{
      public String mykey;
      public int myvalue;

      Pair(){
            mykey="**";
            myvalue=-1;
      }
}
```

The constructor of `Pair` class initializes the variables `mykey` and `myvalue` to `**` and `-1`, respectively. The `HashMapPair` class has an `InternalArray` that stores objects of `Pair` class:

```
class HashMapPair{
      private Pair[] InternalArray;
      private int arraySize;
      private int num;

      public HashMapPair(int size)
      {
            num=0;
            arraySize = size;
            InternalArray = new Pair[arraySize];

            for (int j = 0; j < arraySize; j++) {
                  InternalArray[j]= new Pair();
            }
      }
}
```

As it can be seen above, the constructor of `HashMapPair` class creates an object of `Pair` class for each entry `InternalArray[j]` in the `InternalArray` and makes `InternalArray[j]`-s references

to these objects. Accordingly, we can place a `key` and a `value` in the `InternalArray[j]` using the following lines of code:

```
InternalArray[j].mykey=key;
InternalArray[j].myvalue=value;
```

Develop the following methods for the `HashMapPair` class:

```
public void putPair(String key, int value)
public void removePair(String key)
public int doMappingPair(String key)
public void displayMap()
```

The `putPair` method receives a `key` and a `value`, and inserts them in `hashIndex=hashFunc(key` of `InternalArray` using no-shifting mechanism. The `removePair` method receives a `key`, and deletes the `key` from `InternalArray`. The `doMappingPair` receives a `key` and `return` the `value` corresponding to the `key`. To test the developed methods, 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:

```
HashMapPair myhashmap = new HashMapPair(52);
myhashmap.putPair("AK",114);
myhashmap.putPair("FL",341);
myhashmap.putPair("IA",419);
myhashmap.putPair("MN",201);
myhashmap.putPair("NV",941);
myhashmap.putPair("VT",388);

myhashmap.removePair("VT");
myhashmap.displayMap();

String key;

key="MN";
System.out.println("key:"+ key+" value:"+myhashmap.doMappingPair(
   key));

key="VT";
System.out.println("key:"+ key+" value:"+myhashmap.doMappingPair(
   key));
```

The expected output is printed below:

```
Table:
(AK: 114) (IA: 419) (MN: 201) (FL: 341) (**: -1) (**: -1) (**: -1)
   (**: -1) (**: -1) (**: -1) (**: -1) (**: -1) (**: -1) (NV: 941)
   (**: -1) (**: -1) (**: -1) (**: -1) (**: -1) (**: -1) (**: -1)
   (**: -1) (**: -1) (**: -1) (**: -1) (**: -1) (**: -1) (**: -1)
   (**: -1) (**: -1) (**: -1) (**: -1) (**: -1) (**: -1) (**: -1)
   (--: -1) (**: -1) (**: -1) (**: -1) (**: -1) (**: -1) (**: -1)
   (**: -1) (**: -1) (**: -1) (**: -1) (**: -1) (**: -1) (**: -1)
   (**: -1) (**: -1) (**: -1)
key:MN value:201
key:VT value:-1
```

Hint: For developing `putPair` method, you can copy the body of `put` method from the `HashMap` class into the body of `putPair` method in `HashMapPair` class, and make required adjustments to the body of `putPair` method.

V. Submitting the Assignment

Keep the above lines of code into the body of the `main` method when submitting your response to the assignment.