Université de Paris

Rapport TP6

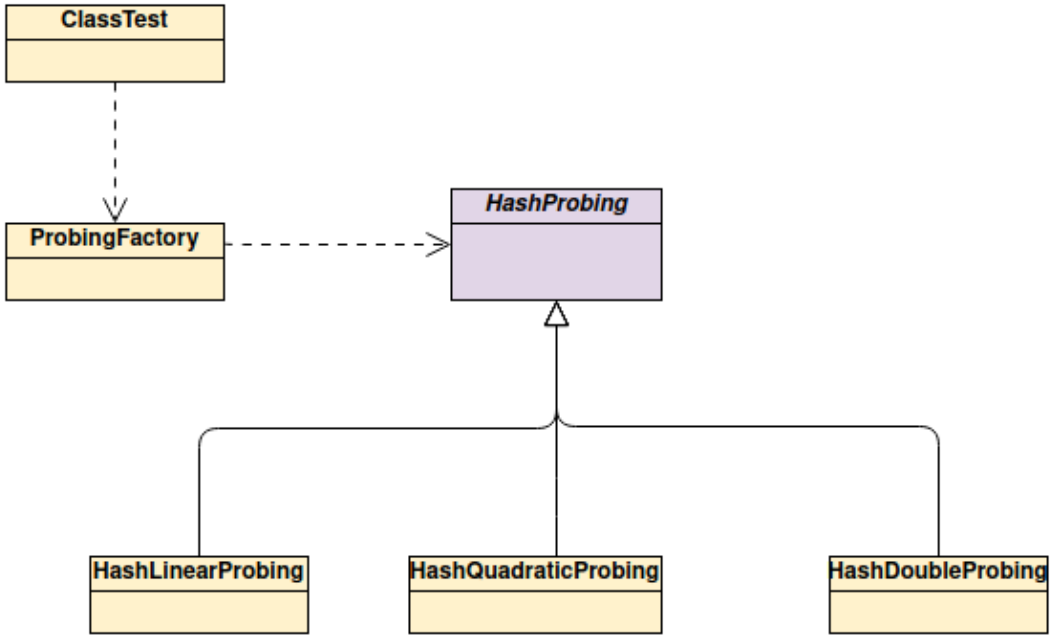
Architecture Base de données

Etudiants :

BIBI thilleli thillelibibi@gmail.com
KHEMSINE Lydia lydiakhemsine@outlook.com

Structure de TP4 :

Diagramme de classe :



notre projet JAVA est structuré en 3 classe et une interface:

1. **HashProbing:**

c'est une interface que les trois autres classes doivent implémenter .dans notre interface on décrit le prototype de 3 fonctions :
PUT : prends deux paramètres : un caractère et un entier
Get : prends un seul paramètre : un caractère
REMOVE : prends un seul paramètre

```
1  package HashProbingPackage;
2
3  public interface HashProbing {
4
5     public void put(char key,int value) ;
6     public int get (char key) ;
7     public void remove (char key) ;
8
9  }
10
```

2. HashLinearProbing :

- HASH: c'une fonction de hachage elle prends en paramètre une clé et elle renvoie la valeur hachée de cette clé en faisant clé modulo 11 , cette fonction est ensuite utilisée pour faire l'insertion, la recherche et la suppression.

- PUT : elle insère une clé et sa valeur dans les tableaux *Keys* et *values.*
pour insérer une clé elle vérifie le *hash* de cette clé dans le tableau Keys , tant que le tableau keys contient une clé à cette position (hash) on avance de 1 la position pour essayer d'insérer plus loin jusqu'à ce qu'on trouve une case libre dans Keys.

```
while(keys[index]!=0) {
    index = (index + 1) % M;
}

keys[index]=key;
values[index]=value;
```

-GET : elle renvoie la valeur d'une clé passée en paramètre si celle ci existe -1 sinon, pour rechercher une clé on avance de 1 le *hash* jusqu'à ce que on trouve la clé concerné.

```
@Override
public int get(char key) {
    int index=hash(key);
    while((keys[index]!=key) &&(keys[index]!=0)) {
        index = (index + 1) % M;
    }
    if (keys [index] == key){
        int m = values[index];
        return m;
    }
    else {
        return -1;
    }
}
```

REMOVE : supprimer une clé passée en paramètre,pour faire on supprime d'abord la clé et sa valeur, après on avance pour récupérer les lignes non vides une par une et on les déplace en appelant put pour les mettre dans les case vide d'avant.

```
public void remove(char key) {
    int index=hash(key);

    while((keys[index]!=key) &&(keys[index]!=0)) {
        index = (index + 1) % M;
    }
    keys[index]=0;
    values[index]=0;

    index=(index + 1)%M;

    while(keys[index]!= 0) {
        char savedKey=keys[index];
        int  savedValue=values[index];
        keys[index]=0;
        values[index]=0;

        put(savedKey, savedValue);
        index=(index+1)%M;

    }
```

-

3.HashQuadraticProbing :

Dans cette classe on applique la même fonction de hachage que la classe (*HashLinearProbing*) Aprés le hachage :

-PUT : pour insérer une clé si collision, on avance suivant un path probing , par exemple pour hash=0 le probing path est (0,1,4,9,16,25…) on le calcule comme suivant:

```
int i = 1 ;
while (keys[newhash] != 0 && keys[newhash]!=key) {
    //newhash = hash ((char)(hash+(i*i)));
    newhash = (hash + i*i) % M;
    i++;
}
```

-GET: pour rechercher une clé si collision entre 2 clés on utilise le path probing pour trouver la clé comme dans put.

-

```
@Override
public int get(char key) {
    int hash=hash(key);
    int index =hash;
    int i =1;
    while((keys[index]!=key) &&(keys[index]!=0)) {
        index = (hash + i*i) % M;
        i++;
    }
    if (keys [index] == key){
        int m = values[index];
        return m;
    }
    else {
        return -1;
    }
}
```

REMOVE : pour la suppression d'une clé, on supprime d'abord la clé et on met cette case a '-' puis sa valeur et la case correspondante dans values à 0 pour combler les vides.

```
    while((keys[index]!=key &&(keys[index]!=0))) {
        index = (hash + i*i) % M;
        i++;
    }

    if(keys[index]==key ) {
            keys[index]='-';
            values[index]=0;
        }

    }
```

4)HashDoubleProbing :

Dans cette classe en plus de la première fonction de hachage, on applique une autre fonction de hachage :

```
private int hash2(char key) {
    return 7-((int)key%7);
}
```

-PUT : pour insérer une clé, on avance suivant un index (hash = (hash +i*hash2)%M).

```
@Override
public void put(char key, int value) {
    int hash = hash(key);
    int hash2 = hash2(key);

    int i = 0 ;
    while (keys[hash] != 0 && keys[hash]!=key) {
        //newhash = hash ((char)(hash+(i*i)));
        hash = (hash + i*hash2) % M;
        i++;
    }


    keys[hash] = key ;
    values[hash] = value ;

}
```

-GET : pour rechercher une clé, on utilise le même path probing que l'insertion et on récupère la clé et sa valeur.

-

```
@Override
public int get(char key) {
    int hash = hash(key);
    int hash2 = hash2(key);

    int i = 0;
    while((keys[hash]!=key) &&(keys[hash]!=0)) {
        hash = (hash + i*hash2) % M;
        i++;
    }
    if (keys [hash] == key)
        return values[hash];

    else {
        return -1;
    }
}
```

REMOVE : pour supprimer une clé, on supprime d'abord la clé et on met la case correspondante dans Keys à '-' et sa valeur dans values à 0 pour combler les vides.

```
        @Override
        public void remove(char key) {
            int hash = hash(key);
            int hash2 = hash2(key);
            int i = 0;

            while((keys[hash]!=key &&(keys[hash]!=0))) {
                hash = (hash + i*hash2) % M;
                i++;
            }

            if(keys[hash]==key ) {
                    keys[hash]='-';
                    values[hash]=0;
                }

            }
```

Classetest :

cette classe sert a tester nos trois classes .
elle ne s'occupe pas de l'instanciation de nos trois classes , pour cela elle passe par la classe *ProbingFactory* .

```
1  package HashProbingPackage;
2
3  public class Classetest  {
4
5      public static void main(String []args) {
6
7          ProbingFactory Pb = new ProbingFactory();
8
9   //test LinearProbing
10         char [] R = {'B','O','E','P','V','L','X','N','K','M'};
11         HashProbing hL = Pb.getclass("Linear");
12         //test put
13         for(int i=0;i<R.length;i++)
14              hL.put(R[i], i);
15         //test get
16          System.out.println("Linear --> M = : " + hL.get('M'));
17          //test remove
18          hL.remove('N') ;
19           //get test after remove
20           System.out.println("N (supprimée): " + hL.get('N'));
21           System.out.println("Linear after remove  --> X = : " + hL.get('X'));
22 //test QuadraticProbing
```

ProbingFactory:
elle sert a construire des objets de type HashLinearProbing, HashDoubleProbing et HashQuadraticProbing et les renvoyer à la classe *classetest*, et ça selon le paramètre *Objectname* reçu.

```
package HashProbingPackage;

public class ProbingFactory {

    public HashProbing getclass(String Objectname) {

        if(Objectname.equals("Linear"))
            return new HashLinearProbing();

        if(Objectname.equals("Double"))
            return new HashDoubleProbing();
        else
        return new HashQuadraticProbing();
    }

}
```

JUNIT:
dans le package test , on a des tests pour chacune de nos classes.