Algorithmes et Pensée Computationnelle

Programmation orientée objet - Exercices avancés

Le but de cette séance est de se familiariser avec un paradigme de programmation couramment utilisé : la Programmation Orientée Objet (POO). Ce paradigme consiste en la définition et en l'interaction avec des briques logicielles appelées **Objets**. Dans les exercices suivants, nous manipulerons des objets, aborderons les notions de classe, méthodes, attributs et encapsulation. Au terme de cette séance, vous serez en mesure d'écrire des programmes mieux structurés. Afin d'atteindre ces objectifs, nous utiliserons principalement le langage **Java** qui offre une panoplie d'outils pour mieux comprendre ce paradigme de programmation.
Le code présenté dans les énoncés se trouvent sur Moodle, dans le dossier **Ressources**.

1 Manipulation de graphes en POO (Java)

Cette partie constitue un rappel sur la notion de graphe orienté. Avant de traiter les exercices de la partie 3, veuillez répondre aux questions suivantes :

1. Quels sont les composantes d'un **graphe orienté** ?
2. Combien de classes sont nécessaires pour représenter un graphe orienté ainsi que toutes ses composantes ?
3. Sachant que vous avez une classe qui représente les **arêtes** et que les **sommets** sont représentés par des chaînes de caractères. Quels pourraient être les attributs de la classe **graph**.

Maintenant que vous avez une idée des attributs de la classe **graph** à définir, déterminer quelles méthodes sont nécessaires au fonctionnement de cet objet. Pour ce faire, vous pouvez répondre aux questions suivantes :

1. Est-ce que l'utilisateur aura besoin de modifier l'objet une fois celui-ci initialisé ?
2. Est-ce que l'utilisateur aura besoin de vérifier l'état de l'objet (existence d'attributs, vérification de valeurs...) ou de certaines parties de l'objet ?

Conseil

1. Revenir sur le cours de la semaine 7 sur Moodle.

Voici les réponses aux questions posées ci-dessus.
Certaines questions peuvent avoir plus d'une réponse étant donné qu'il existe plusieurs manières de représenter un graphe en programmation.

1. (a) Les **sommets**.
   (b) Les **arêtes**.
   (c) Le **poids** de chaque arête.
2. Il faut 2 classes (ou 3 si on cherche à avoir plus d'informations dans les sommets) :
   (a) Une classe **Edge** qui va représenter les arêtes.
   (b) Une classe **Graph**.
3. La classe **Graph** va avoir 2 attributs : un ensemble contenant les **sommets** et un autre ensemble contenant les **arêtes** de celui-ci.

Le code de la classe **Edge** est fourni dans le dossier ressources sur Moodle. Utiliser le fichier **Main.java** dans le dossier **Ressources** sur Moodle pour effectuer des tests. Une fois exécuté, il devrait afficher :

```
The vertex number 1 has a value of: Lausanne
The vertex number 2 has a value of: Geneve
The vertex number 3 has a value of: Berne
{from_vertex=Geneve, weight=35.0, to_vertex=Lausanne}
{from_vertex=Lausanne, weight=100.0, to_vertex=Berne}
{from_vertex=Geneve, weight=120.0, to_vertex=Berne}
Edge between Geneve and Berne has been deleted.
The vertex number 1 has a value of: Lausanne
The vertex number 2 has a value of: Geneve
The vertex number 3 has a value of: Berne
{from_vertex=Geneve, weight=35.0, to_vertex=Lausanne}
{from_vertex=Lausanne, weight=100.0, to_vertex=Berne}

Process finished with exit code 0
```

Question 1: (🕒 *20 minutes*)

Voici une partie de la classe **graph** codée. Implémentez les méthodes **update_weight()**, **new_edge()** et **edge_exist()**.

```
1  import java.util.List;
2  import java.util.HashMap;
3  import java.util.Vector;
4
5  public class graph_empty {
6
7     // Les attributs de la classe graphe
8     public List<Edges> edges = new Vector();   // Utilisation de vector car il faut que l on puisse rajouter ou supprimer des é
          léments de la liste
9     public List<String> vertices = new Vector();
10
11    // Méthode qui permet l ajout d un sommet au graphe.
12    public void add_vertex(String name){
13       this.vertices.add(name); // Méthode qui permet d ajouter un sommet au graphe
14    }
15
16    // Cette méthode va tester si le sommet demandé existe dans le graphe. Si oui retourne le poids, sinon retourne 0.
17
18    public double edge_exist(String from_vertex, String to_vertex){
19
20       // Ecrire votre code ici
21
22    }
23
24    // La méthode ci-dessous vous permet de générer une arête lorsque vous cherchez à en ajouter une à votre graphe. Elle
          fait aussi le test si jamais les sommets utilisés font partis du graphe ou non. Si non, elle va les ajouter au graphe. Cette
          méthode peut être utile dans la méthode new_edge
25
26    private void generate_edge(String from_vertex, String to_vertex, double weight){
27       if (this.vertices.contains(from_vertex) & this.vertices.contains(to_vertex)){
28          Edges new_edge = new Edges(from_vertex,to_vertex,weight);
29          this.edges.add(new_edge);
30       }
31       else {
32          if (!this.vertices.contains(from_vertex)){
33             this.vertices.add(from_vertex);
34          }
35          if (!this.vertices.contains(to_vertex)){
36             this.vertices.add(to_vertex);
37          }
38          Edges new_edge = new Edges(from_vertex,to_vertex,weight);
39          this.edges.add(new_edge);
40       }
41
42    }
43
44    public void update_weight(String from_vertex, String to_vertex, double weight){
45       // Ecrire votre code ici
46    }
47    // Méthode qui va ajouter l arête dans le graphe.
48    public void new_edge(String from_vertex, String to_vertex, double weight){
49       // Ecrire votre code ici
50    }
51
52    // Méthode nous permettant de supprimer une arête du graphe.
53    public void del_edge(String from_vertex, String to_vertex){
54       for(Edges edge : this.edges ){
55          if (edge.from_vertex == from_vertex & edge.to_vertex == to_vertex){
56             this.edges.remove(edge);
57             System.out.println("Edge between " + from_vertex + " and " + to_vertex + " has been deleted.");
58             break;
59          }
60       }
61    }
62
63    // Fonction qui permet d imprimer les composants d un graphe
64    public void print(){
65       for(int i=0; i< this.vertices.size(); ++i){
66          System.out.println("The vertex number " + (i+1) + " has a value of: " + this.vertices.get(i));
```

```
67        }
68        for (Edges edge : this.edges){
69            edge.print();
70        }
71    }
72 }
```

1. La méthode **update_weight()** doit prendre en paramètres : le **sommet** d'origine, le **sommet** d'arrivée ainsi que le poids d'une **arête**. Si cette **arête** existe alors elle change son poids. Sinon, elle imprimera une phrase indiquant que l'**arête** n'existe pas.
2. La méthode **edge_exist()** prendra en paramètre le **sommet** d'origine et le **sommet** d'arrivée. Si cette **arête** est dans le **graph** alors la méthode renvoie son poids, sinon, elle renvoie 0.
3. La méthode **new_edge** doit créer une instance de **Edge** et l'ajouter à l'ensemble edges si la connexion n'existe pas déjà. Si elle existe avec un autre poids mettre à jour le poids. Si elle existe de façon identique alors retournez la dans la console avec print. Enfin, si on est dans aucun des deux cas précédents utiliser la méthode **generate_edge** qui vous est donnée pour créer et ajouter cette arête au **graph**. La méthode **new_edge** aura pour paramètres : le **sommet** d'origine, le **sommet** d'arrivée, le poids.

Conseil

1. Utiliser une boucle **for** pour parcourir toutes les **arêtes** dans le **graph**. Faire un test sur les attributs de **Edge** pour changer le poids.
2. Il faut tester pour chaque **arête** (itération) si elle est égale à celle donnée en paramètres.
3. Il faut utiliser les méthodes **edge_exist()**, **update_edge()** et **generate_edge()** pour écrire cette méthode. Il y a 4 tests à effectuer :
   (a) Si l'**arête** existe.
   (b) Si l'**arête** existante a le même poids que celle indiquée en paramètre de la méthode.
   (c) Si l'**arête** existante n'a pas le même poids que celle indiquée en paramètre de la méthode.
   (d) Utilisez le résultat de **edge_exist** pour simplifier ces tests.

Solution

Java :

```
1  import java.util.List;
2  import java.util.HashMap;
3  import java.util.Map;
4
5  public class Edge {
6     public String from_vertex; // node de départ
7     public String to_vertex;  // node d arrivée ( chaîne de caractère)
8     public double weight;  // poids de l arête
9
10    public Edge(String from_vertex, String to_vertex, double weight) {
11       this.from_vertex = from_vertex;
12       this.to_vertex = to_vertex;
13       this.weight = weight;
14    }
15
16    public void print(){
17       Map<String, String> edge_rep = new HashMap <String,String>(); // Création d un dictionnaire pour
           pouvoir afficher une arête
18       edge_rep.put("from_vertex",this.from_vertex);
19       edge_rep.put("to_vertex",this.to_vertex);
20       edge_rep.put("weight", String.valueOf(this.weight));
21       System.out.println(edge_rep);
22    }
23
24 }
```

Solution

Java :

```
1  public class Graph {
2    // Attributs de la class graph
3    public List<Edge> edges = new Vector();   // Utilisation de vector car il faut que l'on puisse rajouter ou
       supprimer des éléments de la liste
4    public List<String> vertices = new Vector();
5
6    // Methode qui permet l'ajout d'un sommet au graph.
7    public void add_vertex(String name){
8      this.vertices.add(name); // Méthode qui permet d'ajouter un sommet au graph
9    }
10
11   // Cette méthode va tester si le sommet demandé existe dans le graph. Si oui retourne le poids, sinon retourne 0.
12   public double edge_exist(String from_vertex, String to_vertex){
13     for (Edge edge : this.edges) {
14       if (edge.from_vertex == from_vertex & edge.to_vertex == to_vertex) {
15         return edge.weight;
16       }
17     }
18     return 0;
19   }
20   // Cette méthode permet de générer une arête lorsque vous cherchez à en ajouter une à votre graphe. Elle fait
       aussi le test si jamais les sommets utilisés font partis du graphe ou non. Si non, elle va les ajouter au
       graphe. Cette méthode peut être utile dans la méthode new_edge
21   private void generate_edge(String from_vertex, String to_vertex, double weight){
22     if (this.vertices.contains(from_vertex) & this.vertices.contains(to_vertex)){
23       Edge new_edge = new Edge(from_vertex,to_vertex,weight);
24       this.edges.add(new_edge);
25     }
26     else {
27       if (!this.vertices.contains(from_vertex)){
28         this.vertices.add(from_vertex);
29       }
30       if (!this.vertices.contains(to_vertex)){
31         this.vertices.add(to_vertex);
32       }
33       Edge new_edge = new Edge(from_vertex,to_vertex,weight);
34       this.edges.add(new_edge);
35     }
36   }
37
38   public void update_weight(String from_vertex, String to_vertex, double weight){
39     for (Edge edge : this.edges){
40       if(edge.from_vertex == from_vertex & edge.to_vertex == to_vertex){
41         edge.weight = weight;
42         System.out.println("Weight of" + edge + "has been updated");
43       }
44       else {
45         System.out.println("The vertex between the two nodes given does not exist, it will be created.");
46       }
47     }
48   }
49   // Méthode permettant d'ajouter l'arête dans le graphe.
50   public void new_edge(String from_vertex, String to_vertex, double weight){
51     double test_existence = this.edge_exist(from_vertex,to_vertex); // Peut valoir soit le point de l'arête soit 0 si
       elle n'existe pas.
52     if ( test_existence == weight){
53       System.out.println("Edge between" + from_vertex + " and " + to_vertex + " with the same weight already
       exists");
54     }
55     else{
56       if ( test_existence != 0) {
57         System.out.println("Edge between" + from_vertex + " and " + to_vertex + "exists but with a different
       weight and will be overwritten");
58         this.update_weight(from_vertex, to_vertex, weight);
59       }
60       else{
61         this.generate_edge(from_vertex, to_vertex, weight);
62       }
63     }
64   }   // Suite à la page suivante
```

Solution

Java :

```
1   // Méthode nous permettant de supprimer une arête du graph.
2   public void del_edge(String from_vertex, String to_vertex){
3       for( Edge edge : this.edges ){
4           if (edge.from_vertex == from_vertex & edge.to_vertex == to_vertex){
5               this.edges.remove(edge);
6               System.out.println("Edge between " + from_vertex + " and " + to_vertex + " has been deleted.");
7               break;
8           }
9       }
10  }
11  public void print(){
12      for(int i=0; i< this.vertices.size(); ++i){
13          System.out.println("The vertex number " + (i+1) + " has a value of: " + this.vertices.get(i));
14      }
15      for (Edge edge : this.edges){
16          edge.print();
17      }
18  }
19  }
20  public void update_weight(String from_vertex, String to_vertex, double weight){
21      for (Edge edge : this.edges){
22      if(edge.from_vertex == from_vertex & edge.to_vertex == to_vertex){
23      edge.weight = weight;
24      System.out.println("Weight of" + edge + "has been updated");
25          }
26      else {
27      System.out.println("The vertex between the two nodes given does not exist, it will be created.");
28          }
29      }
30
31  }
32  // "Méthode qui va ajouter l'arête dans le graph."
33  public void new_edge(String from_vertex, String to_vertex, double weight){
34      double test_existence = this.edge_exist(from_vertex,to_vertex); // Peut valoir soit le point de l'arête soit 0 si
          elle n'existe pas.
35      if ( test_existence == weight){
36      System.out.println("Edge between" + from_vertex + " and " + to_vertex + " with the same weight already
          exists");
37      }
38      else{
39      if ( test_existence != 0) {
40      System.out.println("Edge between" + from_vertex + " and " + to_vertex + "exists but with a different weight
          and will be overwritten");
41      this.update_weight(from_vertex, to_vertex, weight);
42      }
43      else{
44      this.generate_edge(from_vertex, to_vertex, weight);
45          }
46      }
47
48  }
49  // "Méthode nous permettant de supprimer une arête du graph."
50  public void del_edge(String from_vertex, String to_vertex){
51      for( Edge edge : this.edges ){
52      if (edge.from_vertex == from_vertex & edge.to_vertex == to_vertex){
53      this.edges.remove(edge);
54      System.out.println("Edge between " + from_vertex + " and " + to_vertex + " has been deleted.");
55      break;
56          }
57      }
58  }
59  public void print(){
60      for(int i=0; i< this.vertices.size(); ++i){
61      System.out.println("The vertex number " + (i+1) + " has a value of: " + this.vertices.get(i));
62          }
63      for (Edge edge : this.edges){
64      edge.print();
65          }
66      }
67  }
```