TP : POO et Java

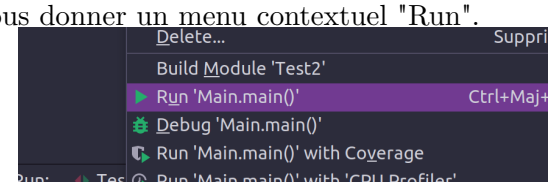
Ce TP se déroulera en utilisant IntelliJ IDEA. Une alternative possible est d'utiliser Eclipse. `Alt+Enter` est votre ami : si c'est rouge, cela peut vous proposer une solution ; ou pas.

1 Hello world ! en Java

Question 1. Créez un nouveau projet Java. S'il manque un JDK, cliquer sur "Download JDK" et laissez IntelliJ installer la machine virtuelle et le compilateur Java.

Question 2. Faire un programme "Hello world !" (une méthode main dans une classe Main elle-même dans un fichier Main.java lui-même dans le répertoire de sources nommé "src/").

Question 3. Exécutez votre programme "Hello world !" : un clic droit sur la classe Main devrait vous donner un menu contextuel "Run".



2 Graphes

Dans cette partie, on se propose de réaliser une librairie de graphe. On rappelle que $G = (V, E)$ avec V l'ensemble des noeuds et E l'ensemble des arrêtes. On souhaite disposer des fonctionnalités suivantes :

1. à partir d'une arête, on trouve les deux noeuds
2. à partir d'un graphe, on peut retrouver l'ensemble des noeuds
3. à partir d'un graphe, on peut retrouver l'ensemble des arêtes
4. à partir d'un noeud et d'un graphe, on trouve l'ensemble des arêtes issues de ce noeud

Comme contrainte supplémentaire, on souhaite que des noeuds puissent être partagés entre deux graphes $G1$ et $G2$.

Question 4. Créez les classes suivantes :

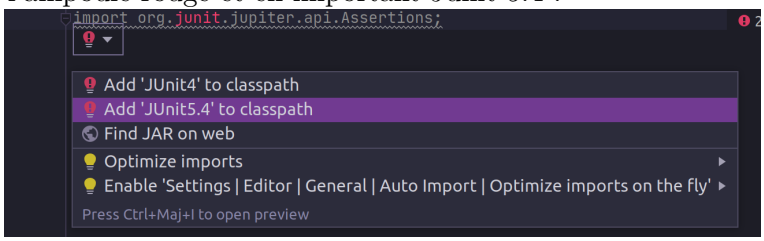
— Node
— Edge
— Graph (avec un constructeur qui prend en paramètre l'ensemble de noeuds et d'arêtes)

La classe Vector<Type> pourra vous aider pour les ensembles :

```
// Exemple d'utilisation de Vector
Vector<Node> v = new Vector<Node>();
Node n = new Node();
v.add(n);
Iterator<Node> it = v.iterator();
while (it.hasNext())
{
      Node mynode = it.next();
}
for (Node mynode : v)
{ }
```

Question 5. Ou va se trouver la méthode `Collection<Edge> getEdges(Node v)` qui correspond à la fonctionnalité 4 ? CLASS GRAPH

Question 6. Testez un peu votre implémentation, par exemple avec le test proposé en annexe A. Si vous avez des problèmes d'import de Junit, IntelliJ vous aidera à résoudre le problème en cliquant sur l'ampoule rouge et en important Junit 5.4 :



Puis, pour lancer le test unitaire, faire bouton droit sur la classe GraphTest et Run.

Question 7. Pensez à réaliser une méthode `toString()` qui affiche votre graphe en console. Comment s'affiche vos noeuds ? Comment générer des identifiants automatiques à vos noeuds (n1, n2, n3...) sans devoir les nommer explicitement et de façon automatique [1] ?

Question 8. (Optionel) Créez un test unitaire pour vos méthodes, notamment celle du graphe [2].

3 (optionnel) Exemple d'algorithme travaillant sur un graphe

Question 9. (Optionnel et long) Coder l'algorithme qui vérifie si un graphe est connexe.

Question 10. (Optionel) Créez un test unitaire pour votre méthode qui teste la connexité.

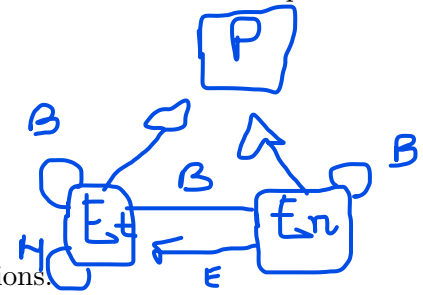
4 Graphes de personnes

On souhaite utiliser notre modèle de graphe pour représenter des relations entre personnes. On a deux types de personnes à envisager :

— Etudiant
— Enseignant

et on part du principe que les relations possibles sont :

— "bois des cafés avec" (possible pour tous)
— "enseigne à" (entre un enseignant et un étudiant)
— "héberge" (entre deux étudiants)



On souhaite créer des classes explicitement pour ces relations.

Question 11. Quelle spécialisation allez vous effectuer pour représenter vos entités et relations ?

Question 12. Introduire une nouvelle classe Personne dans votre modèle. On veut ne pas pouvoir instancier un tel objet (une personne ne peut pas exister, car c'est trop peu précis). Comment faire ? ABSTRACT

Question 13. Où et comment détecter des relations impossibles (pex : enseignant héberge un étudiant ou bien étudiant enseigne à enseignant) en utilisant "instanceof" ? Dans le cas d'une impossibilité on lèvera une exception de type RelationException.

Question 14. Faites quelques tests comme dans l'annexe C.

1. Aide : il vous faudra un attribut statique et surcharger un constructeur...
2. Dans IntelliJ, Vous pouvez vous aider de `https://www.jetbrains.com/help/idea/create-tests.html` et `https://www.jetbrains.com/help/idea/creating-run-debug-configuration-for-tests.html` et peut être du prof, qui a un poil l'habitude...

5 Factory

On veut maintenant modéliser l'université qui forme les enseignants. Ces étudiants deviennent Ingénieur ou bien Enseignant.

Question 15. Ajouter la classe Ingenieur.

Question 16. Créez une interface Universite avec une méthode "former" qui prend un Etudiant en paramètre et renvoie une Personne. Cette université est considérée comme une usine (factory) car elle ne connait pas, à ce stade tous les types de profil qu'elle forme : c'est pour cela qu'elle renvoie des Personne.

Question 17. Créez la class CS qui implémente l'interface précédente. Dans l'implémentation à réaliser implémentez le comportement suivant : 1 fois sur 10, l'étudiant ressort Enseignant, 1 fois sur 10 il ressort Ingenieur et 8 fois sur 10 il reste Etudiant [3].

CS est considéré comme une usine (patron de conception "factory") parce qu'il créé des objets Personne, mais qu'on ne connait pas le vrai type de ces personnes (Ingenieur ou Enseignant). On peut toutefois l'utiliser, comme dans le code suivant que vous pouvez tester :

```
Universite cs = new CS();
for (int i = 0; i<50; i++)
{
    Personne p = cs.former(new Etudiant());
    g.V.add(p); // ajout de la personne dans le graphe
}
```

Question 18. Amusez vous à créer des relations entre vos 50 personnes : tirez deux personnes au hasard et ajoutez au hasard une relation entre ces deux personnes (bois des cafés avec, enseigne à, héberge).

6 Affichage

A ce stade, il serait intéressant de visualiser le résultat. Pas de problème, on peut faire cela en deux temps, trois mouvements !

Question 19. Réalisez dans une méthode `display()` du graphe un dump de votre graphe sous le format utilisé par GraphViz (cf ci-dessous). Testez la visualisation en vous rendant à `https://dreampuf.github.io/GraphvizOnline`. La visualisation de "FDP" (WTF ?) semble donne de bons résultats de visualisation.

```
graph G {

  start -- a0 [label="cafe"] ;;
  start -- b0;
  a1 -- b3;
  b2 -- a3;
  a3 -- a0;
  a3 -- end;
  b3 -- end;
}
```

Question 20. Pour améliorer la visualisation, on souhaite mettre un label qui dépend du type de relations entre les deux noeuds. On pourrait mettre, dans la méthode `display()` du graphe, un test sur le type de l'arête que l'on manipule : si c'est une instance de "héberge", alors on écrirait

3. Ce comportement n'est pas très optimiste, mais pour que la visualisation soit bonne, il faut me garder une bonne proportion d'étudiants pour la suite. On va dire que normalement, le modèle devrait sortir 99% d'ingénieur...

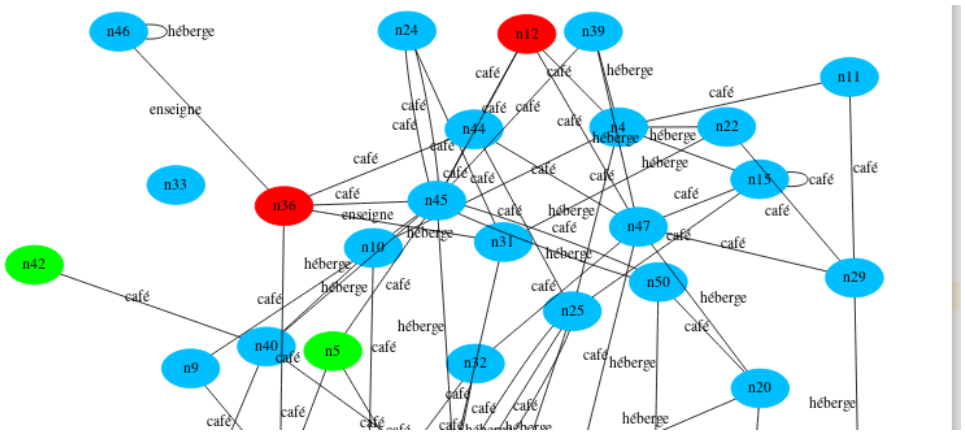


FIGURE 1 – Exemple de sortie graphique (bleu : étudiant, rouge : enseignant, vert : ingénieur)

"héberge". Cependant, ce serait une erreur : le code qui est dans la classe Graphe n'a aucune idée des classes correspondant aux relations "héberge", "bois des cafés", etc. Une solution consiste à créer une méthode `label` sur l'arête, qui renverra la chaine à afficher pour cette arête : il suffit de la surcharger dans les classes filles et le tour est joué.

Question 21. (optionel) Vous pouvez encore améliorer la visualisation en mettant des couleurs aux noeuds, en ajoutant des lignes du type : `n23[color=deepskyblue,style = filled];`. La figure 1 montre la sortie que l'on peut obtenir [4].

Question 22. Remettez au propre le diagramme de classe de l'ensemble de vos classes. On a bien travaillé !

4. On remarquera le truc marrant est que l'étudiant n46 s'héberge lui-même...

7 Pour aller plus loin...

7.1 Généricité

Vous avez remarqué que dans les classes des relations, par exemple "enseign à", les constructeurs utilisent des "Node". En effet, le constructeur doit avoir la même signature que le constructeur parent. On peut résoudre ce problème en utilisant des génériques.

Question 23. Copiez votre code dans un autre package. Essayez de modifier vos arêtes pour qu'elles deviennent génériques.

Question 24. Dans cette version, peut-on mieux gérer plus facilement les contraintes sur les relations entre deux personnes ?

7.2 Définir le comportement d'affichage comme une interface

Question 25. Dans ce TP, nous avons un peu mélangé le fait d'être un graphe, un noeud, une arête et le fait de devoir s'afficher : certaines méthodes du graphe ou de l'arête concernent des aspects d'affichage. Pour être plus propre, on pourrait définir les comportements d'affichage dans une interface. Puis, on pourrait dire que les graphes implémente cette interface. Si l'on veut qu'un graphe ne soit pas obligatoirement afficheable, on peut le faire à un niveau inférieur i.e. un GrapheDePersonne qui implémente cette interface. Cela va de même pour le noeud ou l'arrête : les personnes et les relations impémenteraient l'interface qui définit le comportement d'affichage. Vous pouvez tenter de réaliser cette modification.

Annexes

A Test de getEdges()

```
import java.util.Collection;
import java.util.Vector;
import static org.junit.jupiter.api.Assertions.*;

class GraphTest {

    Node a = new Node();
    Node b = new Node();
    Node c = new Node();

    Edge ab = new Edge(a,b);
    Edge bc = new Edge(c,b);
    Edge ca = new Edge(a,c);

    Graph getGraph() {

        Vector<Node> nv = new Vector<Node>();
        nv.add(a); nv.add(b); nv.add(c);
        Vector<Edge> ne = new Vector<Edge>();
        ne.add(ab); ne.add(bc); ne.add(ca);

        Graph g = new Graph(nv, ne);
        return g;
    }
```

```
    @org.junit.jupiter.api.Test
    void getEdges() {
        System.out.println("Testing getEdges() :)");

        Graph g = getGraph();
        System.out.println("G: " + g);

        // Testing
        Collection<Edge> out = g.getEdges(a);
        assertTrue(out.size() == 2);
        assertTrue(out.contains(ab));
    }
}
```

B Test de connexe()

```
import java.util.Vector;
import static org.junit.jupiter.api.Assertions.*;

public class ConnexeTest {

    Node a = new Node();
    Node b = new Node();
    Node c = new Node();

    Edge ab = new Edge(a,b);
    Edge bc = new Edge(c,b);
    Edge ca = new Edge(a,c);

    Graph getGraph() {
        Vector<Node> nv = new Vector<Node>();
        nv.add(a); nv.add(b); nv.add(c);
        Vector<Edge> ne = new Vector<Edge>();
        ne.add(ab); ne.add(bc); ne.add(ca);

        Graph g = new Graph(nv, ne);
        return g;
    }

    @org.junit.jupiter.api.Test
    void connexe() {
    System.out.println("Testing connexe");

        Graph g = getGraph();
        assertTrue(g.connexe());
        g.E.remove(bc);
        assertTrue(g.connexe());
        g.E.remove(ca);
        assertFalse(g.connexe());
    }
}
```

C Test des relations

```
import static org.junit.jupiter.api.Assertions.*;

public class RelationTest {

    @org.junit.jupiter.api.Test
    void relations() {

        System.out.println("Testing relations");

        Etudiant bob = new Etudiant("Bob");
        Etudiant alice = new Etudiant("Alice");
        Enseignant jf = new Enseignant("JF");
        Enseignant fred = new Enseignant("Fred");
        new BoisDesCafesAvec(jf, fred);
        new BoisDesCafesAvec(alice,jf);
        assertThrows(Exception.class,() -> new EnseigneA(fred, jf) );
        assertThrows(Exception.class,() -> new Heberge(jf, bob) );
    }
}
```