Université Paris Diderot
L2

Programmation Orientée Objet
Année 2022-2023

TD - Séance n°10

Contrôle Continu : Modélisation de réseau

Durée : **1h30**. Les notes manuscrites et les ordinateurs **ne sont pas** autorisés.

L'objectif de ce sujet est de modéliser le comportement d'un réseau de machines indépendantes. Un réseau est une population de *process* (êtres vivants, ordinateurs, processeurs,...) pouvant communiquer les uns avec les autres. Ici nous voulons que le réseau puisse résoudre des problèmes grâce à la communication. Nous partirons de l'hypothèse que cette communication s'effectuera en rounds synchronisés (plus d'explications par la suite). On part de la classe suivante :

```
public abstract class Reseau {
        protected final int population;
        protected final Process[] processes;

        public Reseau(int n) {
                population = n;
                processes = new Process[population];
                for (int i=0; i<population; i++) {
                        processes[i]=new Process(i);
                }
        }
}
```

Le réseau a pour attributs une population fixe de process. Le raisonnement est le suivant : à force de rounds de communication, un process peut arriver à un *état* où il connaît suffisamment bien le réseau pour résoudre un problème. Nous chercherons donc à bien définir le comportement général d'un réseau pour pouvoir ensuite définir des résolutions spécifiques aux problèmes grâce à des interfaces. Allons-y pas à pas.

1. Commençez par définir la classe interne **Process**. Nous aimerions pouvoir manipuler cette classe seulement dans les classes qui héritent de **Reseau** ou sont dans le même package. Quel modificateur d'accès doit avoir la déclaration de **Process** ?
2. Déclarez son unique attribut **etat** qui représente, par un tableau d'entiers de la taille de la population, ce que le process sait sur chacun des process du réseau. On veut lui aussi pouvoir le manipuler dans les classes héritant de **Reseau**. Quel modificateur doit-il avoir ?
3. On peut voir l'indice **i** de la case dans laquelle est rangée un process dans **processes** comme son identifiant (cf constructeur de **Reseau**). Alors pour l'**etat** du process, seule la case correspondant à son identifiant sera initialisée à 1. Définissez ainsi le constucteur de **Process**.
4. Le Réseau s'occupera de produire les messages que reçoivent chacun des process. Dans la classe **Reseau**, déclarez une méthode abstraite prenant en argument un indice désignant un process et renvoyant un tableau d'entiers à deux dimensions.

5. Dans un autre fichier java, définissez une interface fonctionnelle : **Stepper**. Son unique méthode **step** renvoie un nouvel état de process en fonction d'un ancien état et de messages passés en arguments.
6. Dans un autre fichier java, définissez une autre interface fonctionnelle : **Responder**. Son unique méthode **conclusion** renvoie un booléen (réponse à une question par oui ou non) d'après un état de process passé en argument.
7. Vous pouvez ainsi définir les deux méthodes suivantes pour la classe **Process** :
   — **miseAJour** met à jour l'état de process selon des messages et un **Stepper** passés en arguments.
   — **reponse** répond au problème par oui ou non selon un **Responder** passé en arguments.
8. Nous avons fini d'implémenter la classe **Process**. Finissez à présent la classe **Reseau** en implémentant la méthode **resolution** : après un nombre de rounds passé en argument durant lesquels des messages sont produits et chaque process se met à jour, on renvoie un tableau contenant la réponse de chacun des process. A part le nombre de rounds, quels sont les arguments nécessaires ?
9. Dans un nouveau fichier, commencez à présent l'implémentation de la classe instanciable **LOCAL** qui hérite de **Reseau**. Le modèle LOCAL sert à modéliser le fait que dans la plupart des réseaux, les process ne peuvent pas tous directement communiquer entre eux. Ce phénomène est modélisé par un graphe indiquant quels process peuvent échanger des messages. Ce graphe est un attribut de la classe et il est sous la forme d'un tableau de booléens tel que **graph[i][j]** est **true** si et seulement si les process **i** et **j** peuvent communiquer entre eux. Codez un constructeur qui copiera un graphe passé en argument (ce sera d'ailleurs le seul argument) dans l'attribut **graph**.
10. Vous implémentez à présent la méthode **messages** :
    — Si le process **j** ne peut pas communiquer avec le process **i**, **messages(i)[j]** est une ligne de zéros.
    — Si à l'inverse, les process **i** et **j** peuvent communiquer, **messages(i)[j]** contient l'état du process **j**.
11. On aimerait maintenant s'assurer que le graphe passé en argument du constructeur est légal. Déclarez une classe **IllegalGraphException** interne à **LOCAL** qui hérite d'**Exception**. Modifiez le constructeur pour qu'il lève une telle exception si le tableau passé en argument n'a pas la même dimension horizontale que verticale. Est-ce une exception "checked" ou "unchecked" ? De quelle classe aurait-elle dû hériter pour être de l'autre sorte ?
12. Pour finir cette modélisation, nous allons résoudre le problème de la connexité dans un graphe : en se déplaçant de voisin en voisin, est-il possible d'aller de n'importe quel process à n'importe quel process ?
    Déclarez la classe **LOCALConnected** qui a comme attribut un réseau **LOCAL**. Le constructeur initialise un réseau à 5 process d'après un tableau de booléens dont chaque case est tirée aléatoirement. On rappelle que l'instruction suivante attribue un booléen aléatoire à la variable **b** :

```
Random r = new Random();
boolean b = r.nextBoolean();
```

Il est également important de préciser que l'on ne veut pas que ce constructeur lève d'exception.

13. Enfin, implémentez une méthode **resolution** sans argument. Elle fait appel à la méthode du même nom de la classe **Reseau**. Aidez-vous :
— d'une classe anonyme héritant de **Stepper**. Sa méthode **step** marche ainsi :

```
public int[] step(int[] e, int[][] msgs) {
        for (int i=0; i<5; i++) {
                for (int j=0; j<5; j++) {
                        if(msgs[i][j]==1) e[j]=1;
                }
        }
        return e;
}
```

Vous pouvez remplacer ce bloc de code par un simple *.
Pour vous aider à comprendre ce bloc **(pas nécessaire pour cet examen)** :
La façon dont nous avons initialisé nos états et défini nos messages font que si msgs[i][j] vaut 1, c'est que le process courant (celui qui appelle **miseAJour**) communique avec le process **i** (sinon la ligne **msgs[i]** entière vaut 0) et que (par récursivité) ce process **i** a été mis au courant que le process **j** était accessible de voisinage en voisinage. On peut donc actualiser l'**etat[j]** du process courant pour dire que le process **j** est accessible.

— d'une expression lambda implémentant l'interface fonctionnelle **Responder** : si à la fin, les 5 cases de l'état d'un process sont 1, c'est que les 5 process sont accessibles, il faut répondre **true**.