TME 7 : Interface graphique

Objectifs pédagogiques : utilisation de SWING, boucle d'affichage, Design Pattern *Observer*

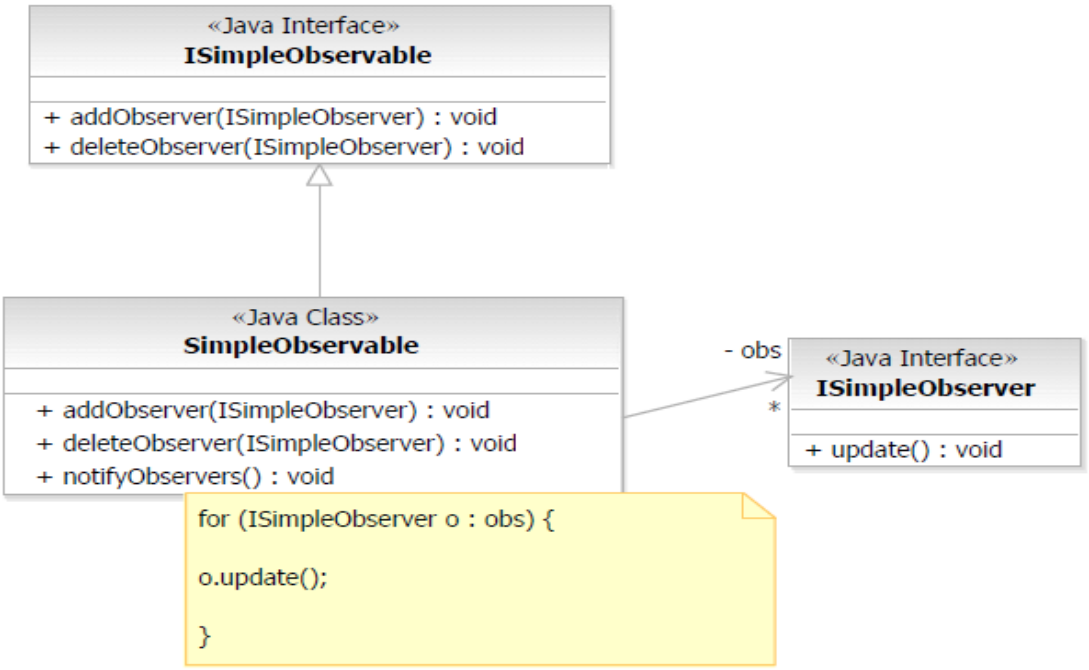
L'objet de ce TME est de permettre de visualiser le déplacement d'un agent dans le labyrinthe par l'utilisation du *Design Pattern*[1] *Observer*. Le DP *Observer* définit une relation entre objets de type un-à-plusieurs de façon à ce que lorsque un objet A (*Observable*) change d'état, tous les objets (*Observer*) qui se sont abonnés à la liste de diffusion de A soient notifiés et mis à jour automatiquement.

Dans notre cas, c'est la classe de simulation *Agent* que l'on souhaite rendre *Observable,* et c'est la classe permettant de visualiser l'agent qui jouera le rôle d'*Observer*. On fera donc dériver la classe agent de *SimpleObservable*, et l'interface graphique de *ISimpleObserver*. Ce schéma simple permet d'assurer la notification des *Observer* (les interfaces graphiques) sans créer de dépendance cyclique entre le modèle (ici la classe *Agent*) et les interfaces.

Mise en oeuvre du DP Observer

1°) Créez un projet « tme7 » et chargez-y l'archive de votre TME6.

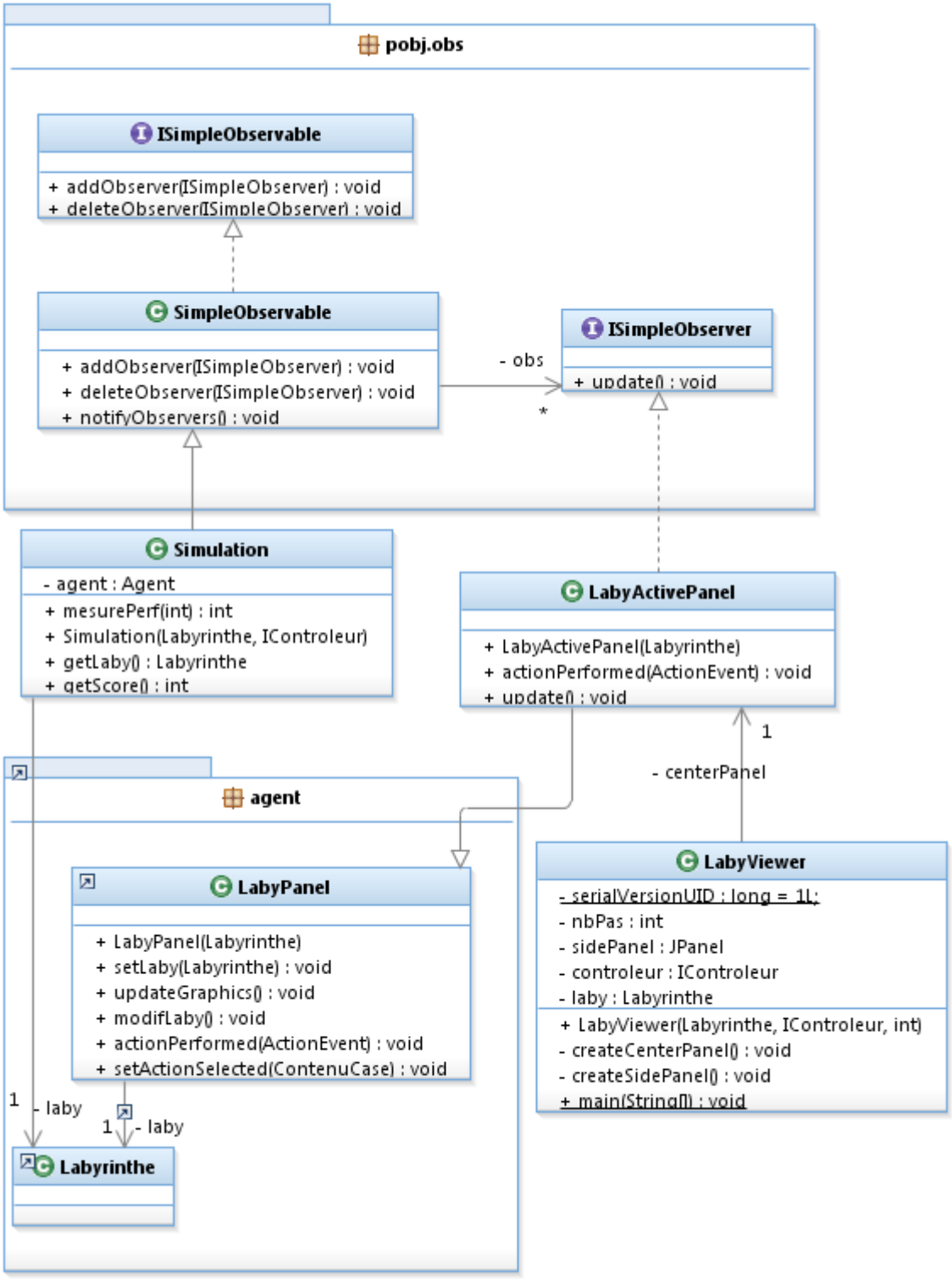
2°) Vous allez déployer une instance du *Design Pattern* (DP) *Observer* afin de synchroniser l'affichage et l'état de la simulation. Pour cela, dans un package *pobj.obs*, construisez les classes et interfaces suivantes:



Les méthodes *addObserver(ISimpleObserver)* et *deleteObserver(ISimpleObserver)* servent respectivement à ajouter ou supprimer un *Observer de la liste « obs »* (*List<ISimpleObserver>*). Les observers s'abonnent aux notifications de l'objet *Observable* en invoquant *addObserver()*. Ils sont ensuite notifiés des changements d'état de l'objet *Observable* (appelé également « sujet ») à chaque fois que celui-ci invoque « *notifyObservers()* ». Chaque *Observer*, dans le corps de sa méthode *update(),* peut alors décider comment traiter cette notification.

1 : En français, on parle de *Patron de conception*

On souhaite pouvoir visualiser le comportement du meilleur agent obtenu à l'issue du processus de sélection génétique. Il faut donc modifier la méthode Simulation.*mesurePerf()* afin de pouvoir accéder à l'état d'une simulation entre deux invocations de *agent.faitUnPas()* et rendre ainsi la simulation *observable.* Pour cela, procédez comme suit :



1. Mise à jour de la classe Simulation en la rendant observable

Faites dériver *Simulation* de *SimpleObservable. V*ous invoquerez *notifyObservers()* après chaque invocation de *faitUnPas()*.

Pour afficher les déplacements de l'agent à chaque pas, il faut pouvoir afficher l'état courant du

labyrinthe, donc le contenu de chaque case. Pour cela, on interrogera la simulation pour obtenir l'état du labyrinthe. Rappelons qu'une simulation **modifie** le labyrinthe sur lequel elle opère. Il faut donc parfois copier le labyrinthe avant de lui passer.

2. LabyActivePanel : un observateur pour une Simulation

Créez une nouvelle classe *LabyActivePanel* qui dérive du *LabyPanel* et implémente *ISimpleObserver.* C'est cette classe que nous utiliserons pour dessiner le centre de la *JFrame* en lieu et place du *LabyPanel* de *LabyBuilder.*

Dans cette classe, surchargez *actionPerformed* de façon à ce que les clics sur les cases du labyrinthe n'aient plus d'effet. (variante à faire si vous avez assez de temps : le clic sur une case doit positionner l'agent à cette position initiale). Implémentez également l'opération *update()* de Observer en forçant l'affichage à se rafraîchir (cf. la classe *LabyPanel*).

Remarque :
Pour ralentir l'affichage, vous pourrez ajouter un *sleep()* dans l'opération *update()* de mise à jour de votre *LabyActivePanel*.

```
try {       Thread.sleep(500); }// en millisecondes

catch (InterruptedException e) { e.printStackTrace(); }
```

3. Ajout du conteneur Swing LabyViewer

Dans le package *agent.laby.interf*, en vous inspirant du code de la classe *LabyBuilder* fournie dans le package *agent.laby.interf*, créez une classe *LabyViewer* qui permet de visualiser le labyrinthe.

1°) Le constructeur de *LabyViewer* prend un agent et un contrôleur en paramètre. Adaptez le constructeur récupéré de la classe *LabyBuilder* en fonction.

2°) Pour assurer l'affichage mis à jour du labyrinthe, on va maintenant modifier le centerPanel de la Frame(cf. *CreateCenterPanel() qui crée un LabyPanel*) pour utiliser à la place un *LabyActivePanel*.

2°) Implémentez une méthode *main()* dans cette classe. Vous y intégrerez les traitements jusqu'à présent effectués dans vos méthodes *main()* de test : charger le labyrinthe depuis un fichier, créer une population aléatoire, la faire évoluer sur un certain nombre de générations, puis extraire le meilleur individu de la Population.

Vous récupérerez alors le meilleur contrôleur après simulation et vous lancerez l'interface graphique *LabyViewer* afin de visualiser graphiquement le fonctionnement du meilleur individu dans le labyrinthe.

3°) La classe *LabyViewer* servant uniquement à afficher le labyrinthe, veillez à supprimer les menus (cf. *createMenus()*) inutiles. De même, certains comportements n'ont pas d'intérêt ici (cf. *createSidePanel()*). Remplacez ceux-ci par un seul bouton « Play » permettant de lancer l'exécution de l'agent. Celui-ci doit déclencher la simulation, en invoquant *Agent.mesurePerf()*. Chaque clic doit engendrer les actions suivantes : 1. Copier le labyrinthe. 2.Créer une Simulation sur ce labyrinthe copié. 3. Positionner le labyrinthe obtenu comme modèle du *LabyPanel* en invoquant *setLabyrinthe()* et 4. abonner le *LabyActivePanel* à la simulation en invoquant *addObserver().* 5. Démarrer la simulation (*mesurePerf*).

Attention : En raison de la façon dont sont implémentés les contrôles Swing, il faut que le clic sur le bouton « play » soit terminé pour rafraîchir l'affichage. On peut invoquer une opération d'affichage dans un nouveau Thread et permettre ainsi de ne pas bloquer l'animation graphique. Utilisez telle quelle cette syntaxe anonyme un peu barbare pour lancer le traitement (*mesurePerf*) dans un nouveau Thread:

```
new Thread(new Runnable(){

	public void run() {

		// invoquer un traitement long ou une animation

	}

}).start();
```

Modifiez votre code en conséquence.

5. Améliorations

Ajoutez une zone de texte (*JTextArea*) portant la description (*toString()*) du contrôleur d'agent utilisé.

Ajoutez des contrôles (*JtextField, Slider*...) permettant de choisir graphiquement les options de l'évolution : nombres de générations, d'individus... et un bouton permettant de lancer une sélection génétique.

Vous essaierez de créer pour cela une nouvelle classe dérivée de *JPanel* et dédiée à la configuration de la sélection génétique.

6. Remise du TME

Question : envoyez un screenshot (jpg) de votre application finale à la fin d'une simulation.