TP 6 : Simuler le modèle d'usine robotisée grâce au MVC

Dominique Blouin, Télécom Paris, Institut Polytechnique de paris
dominique.blouin@telecom-paris.fr

Au cours du TP précédent, vous avez modifié votre modèle objet d'usine robotisée pour lui faire implémenter les interfaces modèle de l'interface graphique Canvas Viewer qui vous a été fournie, afin de visualiser votre modèle sous forme de figures 2D de différentes couleurs, styles et formes géométriques.

Au cours de ce TP, vous allez définir la **vue comportementale** de votre modèle afin de pouvoir le simuler. Puis, pour visualiser cette simulation avec l'interface graphique *Canvas Viewer*, vous devrez fournir une classe qui implémente le contrôleur de l'interface graphique afin qu'elle puisse contrôler l'affichage de votre modèle et sa simulation.

Spécifier le comportement du modèle

Vous allez maintenant modéliser la partie **comportementale** de l'usine robotisée. Cela consiste à définir des méthodes dans les différentes classes de composants de l'usine qui décriront comment les états, représentés par les différentes valeurs des attributs des composants, peuvent évoluer en fonction du temps et des différents événements pouvant se produire dans l'usine.

D'abord définir une nouvelle méthode nommée *behave()* dans la classe *Component* de votre modèle. Le contenu de cette méthode sera vide, et chaque composant de l'usine **dont l'état peut changer au cours du temps** pourra redéfinir cette méthode, et ainsi spécifier ce que le composant fait (i.e. son comportement) lorsque l'usine fonctionne.

Dans la classe *Factory* il faudra redéfinir la méthode *behave()* héritée de la classe *Component*, et pour chacun des composants contenus dans l'usine appeler la méthode *behave()* du sous-composant de l'usine. Ainsi, toute méthode redéfinie par une sous-classe de la classe *Component* sera exécutée, ce qui aura pour conséquence de simuler le comportement spécifique à chacune des sous-classes.

Déplacer les robots

Spécifier d'abord le comportement de la classe *Robot*. Pour l'instant, celui-ci ne consiste qu'à se déplacer d'un composant à un autre dans l'usine. A cette fin, ajouter un attribut à la classe *Robot* pour contenir une **liste de composants** que le robot devra visiter lorsqu'il se déplace dans l'usine.

Dans cette même classe *Robot*, redéfinissez la méthode *behave()* héritée de la classe *Component*. Dans cette méthode, récupérer le premier composant de la liste des composants à visiter par le robot. Puis, appeler une méthode nommée *move()* que vous définirez tel que décrit au paragraphe suivant. Toutefois, avant d'appeler cette méthode *move()*, il faudra vérifier si le robot a atteint le composant à visiter, c'est-à-dire vérifier si la position du robot coïncide avec celle du composant à visiter. Dans ce cas, il faudra récupérer le composant

suivant de la liste des composants à visiter, et faire en sorte que le robot se déplace maintenant vers ce composant. Lorsque la fin de la liste est atteinte, le robot devra se diriger à nouveau vers le premier composant de la liste des composants à visiter.

Définir la méthode *move()* de la classe *Robot*. Dans cette méthode, incrémenter (ou décrémenter selon le cas) les coordonnées *x* et *y* du robot d'une valeur telle que définie par un attribut spécifiant la vitesse du robot, de manière que le robot se rapproche du composant à visiter. Pour l'instant, considérer qu'il n'y a pas d'obstacle entre le composant à visiter et le robot.

Les observateurs

Ouvrir la bibliothèque *canvas-viewer.jar* située dans le répertoire *libs* de votre projet Eclipse de simulateur d'usine robotisée. Dans le package *fr.tp.inf112.projects.canvas.controller* se trouvent les interfaces *CanvasViewerController*, *Observable* et *Observer*.

Ouvrir la classe *CanvasViewer* de la bibliothèque *canvas-viewer.jar*. Vous constaterez que cette classe, qui joue le rôle de la vue (et donc de l'observateur) du MVC, implémente l'interface *Observer*. Cette interface ne définie qu'une seule méthode (*modelChanged*) qui devra être appelée par votre modèle à chaque fois que ses données auront changées, afin que la ou les vues puissent réafficher les données pour que l'affichage reste cohérent avec les données du modèle.

Examiner le corps de la méthode *modelChanged* dans la classe *CanvasViewer*. Celle-ci ne fait qu'appeler les méthodes *repaint* de la barre de menu et du panneau qui affiche les figures de votre modèle. Lors de l'affichage des menus de la barre de menus, les menus *Start Animation* et *Stop Animation* seront alors activés ou non, selon que votre modèle est en cours de simulation ou non, tel que déterminé par un appel de la méthode *isAnimationRunning()* de l'interface de contrôleur présentée plus loin dans ce document.

Implémenter l'interface de l'observable

Afin de notifier la vue lorsque ses données changent, votre modèle devra implémenter l'interface *Observable* :

```
public class Factory extends Component implements Canvas, Observable {
```

Tel que vu en classe, cette interface ne contient que deux méthodes servant à ajouter ou à enlever un observateur de l'observable. Afin de les implémenter, votre classe *Factory* devra donc déclarer un attribut servant à contenir ses différents observateurs.

Notifier la ou les vue(s) lorsque les données du modèle ont changé

Puisque le simulateur s'appuie sur une architecture MVC, il faudra modifier le code de votre modèle pour que tout changement de ses donnés, telles que les coordonnées d'un robot qui se déplace par exemple, puisse notifier les vues qui auront été enregistrées comme observatrices du modèle. Ainsi, ces dernières pourront se réafficher pour visualiser les données modifiées du modèle.

A cette fin, dans la classe *Factory*, il faudra créer une méthode *notifyObservers()* qui appellera la méthode *modelChanged()* de chaque observateur qui se sera enregistré auprès du modèle.

Cette méthode *notifyObservers()* devra être appelée par le modèle à chaque fois que ses données seront **modifiées**, et ce pour toutes les classes de votre modèle devant être visualisées par l'interface graphique.

Il y a plusieurs façons de réaliser cela. Une manière simple consiste à ajouter un attribut de type *Factory* à la classe *Component*, de manière à ce que tout composant puisse appeler la méthode notifyObservers() de la classe *Factory* lorsque ses données sont changées via les appels de ses mutateurs (setters).

Implémenter l'interface du contrôleur

Ouvrir l'interface *CanvasViewerController* de la bibliothèque *canvas-viewer.jar*. Il vous faudra créer une classe *SimulatorController* qui implémentera cette interface, et dont une instance sera donnée, via constructeur, à l'instance de la classe *CanvasViewer* que vous utiliserez pour visualiser votre modèle.

L'interface *CanvasViewerController* hérite de l'interface *Observable*. Cela veut dire que la vue s'enrtegistre auprès du modèle **indirectement** via le controleur. Il vous faudra donc dans votre contrôleur maintenir une référence vers le modèle et implémenter les méthodes de l'interface *Observable* en appleant les méthodes correspondantes du modèle.

En lisant la Javadoc des méthodes de l'interface *CanvasViewerController*, fournissez des implémentations des différentes méthodes requises par l'interface. Le contrôleur sert à démarrer et arrêter la simulation via les méthodes *startAnimation()* et *stopAnimation()*, qui sont appelées par la vue lorsque les menus du même nom sont sélectionnés. Par ailleurs, un appel à la méthode *isAnimationRunning()* permet de dire à la vue si la simulation est en cours d'exécution ou non, afin que celle-ci puis gérer l'activation des menus de contrôle de l'animation.

Afin d'implémenter les méthode *startAnimation()*, *stopAnimation()* et *isAnimationRunning()* de votre classe contrôleur, ajouter un attribut à la classe *Factory* qui mémorisera si la simulation a été démarrée ou non. Ajouter également des méthodes *startSimulation()*, *stopSimulation()* et *isSimulationStarted()* à la classe *Factory* pour gérer cet attribut. N'oubliez pas de notifier les observateurs lorsque la valeur de cet attribut est changée.

Dans la méthode *startAnimation()* du contrôleur, copier le code suivant :

```
factoryModel.startSimulation();

while (factoryModel.isSimulationStarted()) {
    factoryModel.behave();

    try {
        Thread.sleep( 200 );
    }
    catch (InterruptedException ex) {
        ex.printStackTrace();
    }
}
```

L'interface graphique prendra soin de lancer la simulation de votre modèle dans une tâche (*thread* ou *processus léger*) dédiée afin de ne pas perturber son affichage, qui s'exécute dans une tâche spécifique de la JVM nommée *Event Dispatch Thread*.

La méthode *sleep* sert à attendre 200 millisecondes entre chaque exécution de la méthode *behave()* de l'usine, afin d'éviter que l'animation ne s'exécute trop rapidement et ainsi permettre à l'utilisateur de mieux visualiser la simulation. Cette valeur peut être changée au besoin.

Noter les instructions *try – catch* permettant de gérer l'exception *InterruptedException*, qui sera levée si une autre tâche venait à interrompre la tâche dans laquelle s'exécute le simulateur.

Finalement, implémenter les méthodes *stopAnimation()* et *isAnimationRunning()* de votre classe contrôleur en déléguant ces opérations au modèle.

Lancement de la simulation

Afin de tester votre simulation, dans une classe test, instancier une usine contenant un robot, deux machines et une station de recharge. Ajouter ces trois composants à la liste des composants à visiter par le robot.

Instancier la classe *CanvasViewer* de l'interface graphique, en utilisant cette fois le constructeur prenant en paramètre un contrôleur, et vérifier que votre robot se déplace correctement en visitant successivement les deux machines et la station de recharge.