TP-Projet 7 Modéliser un système de production de biens robotisé (vue comportementale)

Au cours du TP précédent, nous avons modélisé la structure du système de production robotisé pour lequel nous voulons développer un simulateur, et nous avons fait implémenter par notre modèle les interfaces fournies par un visualisateur de figures données afin de pouvoir visualiser le modèle en tant que figures géométriques dessinées dans un canevas. Nous avons également implémenté les interfaces nécessaires à cette visualisation via le patron de conception MVC.

Au cours de ce TP, nous allons modéliser la partie **comportementale** de l'usine de production. Cela consiste à définir des méthodes qui décriront comment les états des différents composants de l'usine de production peuvent évoluer en fonction du temps et des différents événements pouvant se produire dans l'usine.

Retélécharger le code de l'interface graphique

Retéléchargez le code de l'interface graphique ici, supprimez de votre workspace l'ancien projet de l'interface graphique et importez le nouveau projet dans le workspace en suivant les instructions du TP 6.

Vous devrez implémenter la nouvelle méthode **`getStyle()`** qui a été ajoutée à l'interface **`Figure`**. Celle-ci demande de retourner une instance d'une classe que vous devrez créer et qui devra implémenter la nouvelle interface **`FigureStyle`** du projet de l'interface graphique. Cette interface spécifie des méthodes qui permettent à l'interface graphique d'obtenir différentes caractéristiques du style d'une figure telles que sa couleur en termes de l'intensité de chaque couleur primaire rouge, vert et bleu.

Composants comportementaux

Nous allons d'abord définir une nouvelle méthode nommée **`public void behave()`** dans la classe **`Component`** de votre modèle de l'usine. 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 héritée, et ainsi spécifier ce que *fait le composant* (i.e. son comportement) lorsque l'usine est en opération (i.e. lors de la production de biens).

La classe Robot

Nous allons d'abord coder le comportement de la classe **`Robot`**. Le comportement d'un robot consiste à déplacer un bien d'une machine à une autre dans l'usine en fonction des différentes étapes de fabrication du bien. A cette fin, ajoutez un attribut à la classe **`Robot`** pour contenir une **liste de composants** que le robot devra visiter lorsqu'il se déplace dans l'usine. Ajoutez également une méthode à la classe **`Robot`** qui permettra d'ajouter des composants à visiter à cette liste.

Dans cette même classe **`Robot`**, redéfinissez la méthode **`behave()`** héritée de la classe **`Component`**. Dans cette méthode, vous devez récupérer le premier composant de la liste des composants à visiter par le robot. Puis, vous appellerez une méthode nommée **`move()`** que

vous implémenterez tel que décrit au paragraphe suivant. Toutefois, avant d'appeler la méthode **`move()`**, il faudra vérifier si le robot a atteint le composant à visiter (c'est-à-dire que sa position coïncide avec celle du composant). Dans ce cas, il faudra alors récupérer le composant suivant de la liste des composants à visiter et faire en sorte que le robot se déplace maintenant vers ce composant.

Définissez ensuite la méthode **`move()`** dans la classe **`Robot`**. Dans cette méthode, incrémentez (ou décrémentez) les coordonnées x et y du robot d'une valeur d'un pixel, de manière à ce que le robot se **rapproche** du composant à visiter. Pour l'instant, nous considérons qu'il n'y a pas d'obstacle entre le composant à visiter et le robot.

Notifier la vue lorsque les coordonnés d'un composant ont changé

Puisque notre simulateur s'appuie sur une architecture MVC, il faudra modifier le code pour que tout changement des donnés des composants telles que les coordonnées d'un robot qui se déplace puisse notifier les vues qui se sont enregistrées comme observatrices du modèle. Ainsi, ces dernières pourront réafficher les composant et permettre de visualiser les nouvelles données du modèle.

Il y a plusieurs manières de réaliser cela. On peut soit faire implémenter l'interface **`Observable`** par la classe **`Component`** et ajouter une liste d'observateurs à cette classe ainsi que les méthodes associées tel que vu en classe, ou tel que fait dans la classe **`Factory`**. Puis il faudra modifier le code de toutes les méthodes des classes des composants qui modifient les données afin d'appeler la méthode **`notifyObservers()`**. Il faudra également modifier les méthodes **`addObserver()`** et **`removeObserver()`** de la classe **`Factory`** afin d'ajouter ou de supprimer les observateurs des composants lorsqu'un observateur est ajouté ou supprimé à l'usine. Même principe pour les méthodes **`addComponent()`** et **`removeComponent()`**.

Une autre manière consiste à ajouter un attribut de type **`Factory`** à la classe **`Component`** et à implémenter **`notifyObservers()`** de la classe **`Component`** en appelant la méthode **`notifyObservers()`** sur l'objet **`Factory`**.

A vous de choisir…

Lancement de la simulation

Nous allons maintenant implémenter la méthode **`startAnimation()`** de la classe **`SimulatorController`** afin que la simulation se lance lorsque vous cliquerez sur le menu *Start Animation* de l'interface graphique. Pour ce faire, ajouter un attribut de type **`boolean`** à la classe **`Factory`** qui mémorisera si la simulation de l'usine a été démarrée ou non. Ajoutez à cette classe des méthodes **`startSimulation()`**, **`stopSimulation()`** et **`isSimulationStarted()`** pour gérer cet attribut. N'oubliez pas de notifier la vue lorsque la valeur de cet attribut est changée.

Dans la méthode **`startAnimation()`**, copiez le code suivant

```
factoryModel.startSimulation();
final Runnable runnable = new Runnable() {
    @Override
    public void run() {
        factoryModel.startSimulation();
```

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

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

new Thread( runnable ).start();
```

Implémentez également les méthodes **stopAnimation()** et **isAnimationRunning()** de votre classe contrôleur en déléguant ces opérations au modèle.

Dans la classe **Factory** il faudra également redéfinir la méthode **behave()** héritée de la classe **Component**. Dans cette méthode, il faudra appeler la méthode **behave()** pour chacun des composants contenus dans l'usine. Ainsi, la méthode **behave()** que vous avez définie dans la classe **Robot** sera exécutée, ce qui aura pour conséquence de déplacer le robot vers les composants à visiter.

Finalement, afin de tester votre implémentation, créez une classe test avec une méthode **main**. Instanciez une usine et ajoutez-y un robot et deux machines. Ajoutez également ces deux machines à la liste des composants à visiter par le robot. Lancez l'interface graphique et vérifiez que votre robot se déplace bien entre les deux machines.