POO – Programmation Orientée Objet en Java

Fiche de TP numéro 3

Des balles, des lignes et des interfaces...

On souhaite développer une application Java reprenant les règles du jeu *Glines*. *Glines*, ou "Cinq et plus" est un jeu de logique où vous devrez aligner 5 billes de même couleur. Au début de chaque tour, l'ordinateur place 3 billes de couleur différente au hasard sur le plateau. Ensuite, vous devez cliquer sur une bille pour la sélectionner et cliquer à l'endroit où vous désirez la déplaçer. Lorsque 5 billes de la même couleur sont alignées, elles disparaissent, et vous avez même un tour de répit sans billes supplémentaires. Le but du jeu est donc de survivre le plus lontemps possible : si le plateau est recouvert de billes, vous avez perdu. Le joueur doit faire attention, car une bille peut être coincée par les autres, et ne pas pouvoir être déplaçable partout !

Puisque vous ne possédez pas encore toutes les connaissances nécessaires à la réalisation de ce genre de projet, le travail vous a été simplifié. L'application a été divisée en deux parties distinctes :

— une partie graphique, qui permet d'interagir avec l'utilisateur en affichant la grille de billes eten permettant de déplacer celles-ci par de simples clics ;
— une partie modèle, qui contient la modélisation du jeu.

L'interface graphique du jeu vous est fournie par l'intermédiaire d'un `package` sous la forme d'un fichier `guilines.jar`. Votre travail sera donc de réaliser la modélisation de l'application, i.e. d'implémenter les différentes classes permettant de contenir les informations de la grille, puis d'agir sur ces données our les mettre à jour, suite à un déplacement par exemple.

Exercice 1 : Démarrage

Récupérez le fichier `guilines.jar` sur Moodle. Ce fichier, contient le `package` de même nom, et dans le paquetage `guilines` se trouve l'ensemble des classes et interfaces nécessaires au bon fonctionnement de l'interface graphique. Enregistrez-le dans le répertoire `bin`.

Notez que l'exécution de la commande `jar -tvf guilines.jar` vous permettra de lister les différentes classes présentes dans le fichier `.jar`. Mais la seule chose qui vous importe pour le moment est de savir qu'il contient l'interface `guilines.InterfaceDesBalles` que vous allez devoir implémenter.

Exercice 2 : Implémenter l'interface

Puisque l'interface graphique est déjà implémentée, il ne s'agit pas de se lancer au hasard dans la conception du modèle. Les concepteurs de l'interface graphique ont fourni dans le package une interface java reprenant l'ensemble des méthodes nécessaires au bon fonctionnement du jeu (et surtout à une bonne communication interface graphique vers modèle). La documentation de cette interface vous est fourni sur Moodle. Consultez-la.

Votre travail va maintenant être de réaliser le contrat de cette interface java, c'est-à-dire d'écrire une classe java implémentant cette interface et les méthodes nécessaires au bon fonctionnement du jeu. Les exercices suivants vous guident pour cette implémentation.

Exercice 3 : Une première implémentation

Nous allons dans un premier temps réaliser une implémentation simpliste du modèle. Celle-ci ne permettra que de représenter la grille de billes, de déplacer une bille puis d'en faire apparaître de nouvelles. Pour cela :

— Déclarez une classe `MonJeu.java` par exemple, qui dit implémenter l'interface qui vous est fournie. N'oubliez pas d'importer le package `guilines` contenant l'interface.
— L'interface `InterfaceDesBalles`, puisqu'elle se trouve dans la paquetage `guilines`, doit être importée `import guilines.InterfaceDesBalles;`
— La grille de billes sera représentée par un tableau d'entiers à deux dimensions. Chaque entier représente la couleur de la bille présente dans la case de coordonnées `[i][j]`. La valeur `-1` représente une case vide.Le package qui vous est fourni propose neuf couleurs différentes, codées sur des entiers de 0 à 8. Dans un premier temps, trois de ces couleurs sont suffisantes pour obtenir une version simplifiée du jeu.
— Une fois les attributs déclarés, écrivez les constructeurs et accesseurs nécessaires à l'implémentation de l'interface.
— Écrivez une méthode permettant de générer aléatoirement de nouvelles billes. La classe `java.util.Random` vous fournira toutes les méthodes nécessaires à la génération aléatoire des couleurs et positions de ces billes. pensez à utiliser la documentation de l'API ! Utilisez cette méthode pour initialiser la grille.
— Écrivez une méthode permettant de déplacer une bille. Dans un premier temps nous dirons qu'un déplacement est valide si la première case fournie contient une bille et sui la deuxième case fournie est vide. Après une déplacement, n'oubliez pas d'ajouter à la grille les nouvelles billes. Pensez à utiliser la méthode déjà implémentée pour générer ces nouvelle billes.
— Écrivez une méthode qui retourne les cases de la grille modifiées dernièrement. L'interface ne rafraîchit que les zones concernées par un changement (les coordonnées des cases concernées par un déplacement). Dans un premier temps, pour obtenir un rendu rapide, cette méthode retournera l'intégralité des coordonnées de la grille. Vous allez avoir besoin de la documentation de l'API pour utiliser deux nouvelles classes et une nouveleinterface :
  — Le retour est une liste (`java.util.List`) de `java.awt.Point` (i.e. `List<Point>`). Consultez la documentation de la classe `Point` et de l'interface `List`
  — Pour retourner une liste, il va falloir la créer. Vous avez donc besoin d'une classe qui implémente `List` : choisissez la classe `java.util.Vector`. Consultez la documentation. Comment la construire ? Comment ajouter un élément ?
  — Dans la méthode `getNouveaux()` vous instancierez une variable locale comme un vecteur de points : `Vector<Point> maListe = new Vector<Point>();`
  — N'oubliez pas les importations nécessaires.

Exercice 4 : Une classe principale

Après avoir débuté l'implémentation de l'interface, vous ressentirez (peut-être) l'envie de tester votre première réalisation. Il suffit pour cela de réaliser une classe `DemoLines` qui contiendra la méthode `main` dont la tâche est de :

— créer une instance du modèle, donc de `MonJeu`
— puis de lancer l'interface graphique `Lines` en lui passant le modèle en paramètre.

Voilà le résultat :

```
public class DemoLines {
    public static void main(String[] args) {
    InterfaceDesBalles monJeu = new MonJeu();
    // creer la fenetre
    Lines fenetre = new Lines("LILines",monJeu);
    }
}// DemoLines
```

Pour la compilation et l'exécution, il faudra spécifier à la machine virtuelle dans la variable `classpath`

qu'elle doit utiliser le fichier `guilines.jar` :

```
javac -cp src:bin:bin/guilines.jar src/*.java
java -cp bin:bin/guilines.jar DemoLines
```

Exercice 5 : Contrôler le déroulement du jeu

Déplacer les billes, c'est rigolo. . .mais si personne ne s'occupe de l'alignement des billes, le jeu risque de vite devenir lassant. Par définition, il est du devoir du modèle de réaliser ce genre de contrôle sur le jeu. C'est donc dans votre partie ! À vous maintenant de vérifier s'il y a bien `nbBillesAlignees` après un déplacement, puis, si c'est le cas, de mettre à jour la grille. Attention, la bille déplacée peut-être au milieu de deux alignements (un vertical et un horizontal). N'oubliez pas que lorsqu'on a `nbBillesAlignees`, les nouvelles billes ne tombent pas.

Vous pourrez profiter de la possibilité qui est donnée au modèle de choisir le nombre de billes alignées (cela ne concerne pas l'interface graphique) : la règle du jeu c'est 5 billes alignées, mais 3 c'est plus agréable pour tester.

Exercice 6 : Pour aller plus loin. . .

Vous avez maintenant réalisé un jeu fonctionnel, mais basique et limité. À vous maintenant de l'améliorer (au moins pour respecter les règles du jeu !) :

- Pour le moment, votre modèle ne contient qu'une seule classe (`MonJeu`). Ce n'est pas une conception très objet. . .Implémentez une classe `Bille` qui modélisera un élément de la grille. `MonJeu` contiendra maintenant un tableau à deux dimensions de `Bille`. Cela vous aidera par la suite.
- Dans la version implémentée actuellement, un déplacement de bille est possible si la case d'arrivée est libre. Dans la version originale, le déplacement n'est possible que s'il existe un chemin de cases vides entre la bille choisie et la case d'arrivée (une bille ne de déplace qu'horizontalement ou verticalement). Modifiez le code pour prendre en compte cette règle ;
- La mise à jour de la grille graphique est assez lourde : pour le moment, à chaque déplacement, c'est la totalité de la grille qui est rafraîchie. Réfléchissez à une façon de ne fournir à l'interface graphique que les coordonnées des cases qui doivent réellement être redessinées (la bille déplacée, les nouvelles billes, . . .)
- Calculez un score (2 points par bille supprimées)
- . . .