IUT de Paris Descartes – Base de la Programmation Objet

Travaux Pratiques – Sujet n°10

Objectifs

— Retravailler le code source non pas pour ajouter une fonctionnalité supplémentaire au logiciel mais pour améliorer sa qualité (sa lisibilité, simplifier sa maintenance, ou changer sa généricité).
— Favoriser le polymorphisme en évitant des dépendances vers les sous-classes d'une hiérarchie.
— Externaliser la création d'objet dans une fabrique statique.

Les chenilles revisitées

Lors d'une séance de travaux dirigés précédente, nous avons développé une application permettant l'animation des déplacements de chenille. L'architecture de notre application mettait en œuvre essentiellement l'héritage. Le but de cette séance est de retravailler notre code de manière à rendre la classe `Chenille` le plus possible indépendante de l'arbre d'héritage des anneaux. Cela va être réalisé par transformations successives. Le code résultant de la séance précédente est fournie sur le serveur commun.

1. Créez un projet, importez les fichiers sources nécessaires et étudiez les relations de dépendance (héritage, aggrégation et utilisation) qui lient les différentes classes. Dessinez le diagramme de classe (sans détailler les attributs et les méthodes des classes mais en vous focalisant sur leurs dépendances).
2. L'objectif est de rendre la classe utilisatrice (la classe `Chenille`) indépendante des sous-classes de la hiérarchie (la classe `Tête`) de façon à ce que si cette hiérarchie est enrichie la classe cliente puisse en profiter sans être modifiée.

   La première chose à faire consiste à réorganiser les paquetages de l'application. Les classes `Anneau` et `Tête` (la hiérarchie) doit disposer de son propre paquetage. Créez un paquetage nommé `anneaux` et déplacez-y les deux classes.
3. Nous voulons que la classe `Chenille` n'emploie plus directement la classe `Tête` mais uniquement la classe `Anneau`. Pour cela, il est nécessaire que la chenille sache déplacer et dessiner les anneaux quelque soit leur type. Ces deux méthodes doivent avoir des prototypes communs dans les deux classes (l'une spécialisant l'autre).

   Étudiez les classes `Anneau` et `Tête` et déterminer les méthodes manquantes, les données qu'elles nécessitent et la façon d'unifier les prototypes des méthodes. Implémentez votre solution.
4. Corrigez le code de la méthode `déplacer` de la classe `Chenille`.
5. Faites en sorte que la classe `Chenille` ne déclare plus d'attribut de type `Tête` mais uniquement un tableau de `NbAnneaux` + 1 (pour la tête) références de type `Anneau`.
6. Seul le constructeur de la classe `Chenille` emploie encore directement la classe `Tête` lors de la création des objets la composant. Une solution courante consiste à déléguer (sous-traiter) la création des objets à une classe tiers.

   Créez une classe nommée `FabriqueAnneau` dans le paquetage `anneaux`. Cette classe disposera d'une seule méthode de classe (i.e. statique) retournant un anneau. Cet anneau sera créé aux coordonnées passées en paramètre. La méthode recevra aussi un numéro qui lui permettra de décider du type de

l'anneau devant être créé. Un numéro égal à 0 donnera lieu à la création d'une tête. Tout autre numéro verra la création d'un simple anneau.

7. Employez cette nouvelle classe dans le constructeur de la classe `Chenille`.
8. Développez une nouvelle sous-classe d'`Anneau` nommée `AnneauClignotant`. Ces anneaux se déplacent comme des anneaux standard mais se dessinent une fois sur deux en majuscule et l'autre fois en minuscule (voir les méthodes de classe de `java.lang.Character`).
9. Modifiez la fabrique d'anneau de façon à ce qu'un anneau sur trois soit un clignotant. Relancez le programme.
10. Dessinez le nouveau diagramme de classe (sans détailler les attributs et les méthodes des classes mais en vous focalisant sur leurs dépendances).