

CORBEILLE DÉCOUVERTE DES DESIGN PATTERN OPALE

Tout afficher + Tout réduire -

SUJET

✓ Énoncé

EXERCICE : CORBEILLE DÉCOUVERTE DES DESIGN PATTERN


Question

Pour cette corbeille, vous allez développer en C# une petite application console tout en utilisant, pas à pas et selon les besoins, différents Design Pattern pour décrire vos objets.

Pour cela vous allez utiliser l'environnement de développement logiciel que vous avez installé et configuré pendant la séquence 0. Cet environnement de travail est composé de :

- Visual Studio 2019
- NET Core 3.0
- Git

Vous n'oublierez donc pas de réaliser les activités proposées dans cette corbeille en utilisant l'outil Git de gestion des versions que vous avez installé dans votre environnement.

Si besoin, l'outil gratuit ArgoUML que vous avez utilisé lors du workshop de la séquence précédente pourra vous aider pour travailler la conception UML de votre application : <https://argouml.fr.softonic.com/> 

Question

1. Créez un projet Application Console .NET Core
2. Créez deux Namespace appelés Élément et Univers dans l'application Console

Sujet :

Nous voulons réaliser une application qui va permettre de modéliser un Univers à deux dimensions (X, Y) contenant des Éléments chimiques. Les éléments peuvent être ajoutés dans l'univers, et se sont dotés d'une position.

L'univers a des dimensions finies, elles doivent être spécifiées. Vous traitez toutes les incohérences par des exceptions ou gestion des erreurs :

- Éléments en double dans l'univers,
- Éléments à la même position,
- Éléments en dehors de l'univers.

Il est possible de parcourir l'univers pour récupérer l'ensemble des éléments, ou de connaître l'élément qui se trouve à une certaine position.

Les éléments ont trois propriétés : un nom, un symbole chimique, et une masse.

Vous commencez par un petit échantillon d'atomes :

Atome	Symbole	Masse atomique
Carbone	C	12,01074 u
Hydrogène	H	1,00794 u
Lithium	Li	6,941 u

Les classes qui concernent l'univers et les éléments doivent être dans deux Namespace séparés.

Voilà comment vous procéderez :

1. Modéliser les différents objets de votre l'application (diagramme de classes).

Envisagez au maximum les évolutions de votre modèle en prenant bien soin de décomposer vos abstractions de vos implémentations (design pattern Bridge), ça vous servira par la suite !

Question

Vous allez maintenant déporter l'instanciation des éléments (le mot clé new) dans une fabrique. La fabrique sera accessible grâce à un singleton, et permettra de créer des éléments pré configurés (nom, masse, symbole) à partir du simple nom de l'élément chimique.

Vous allez aussi ajouter des événements qui seront levés :

- Sur l'univers lors de l'ajout d'un élément dans l'univers
- Sur l'univers lors du retrait d'un élément de l'univers
- Sur les éléments lors du déplacement spatial

Voilà comment vous procéderez :

1. Implémentez la fabrique abstraite (en DP Singleton) et le DP Observer (3 events)
2. Mettez à jour la modélisation

Vous allez maintenant ajouter un autre type d'élément dans l'univers : les molécules.

Les molécules sont formées à partir d'un assemblage d'au moins deux atomes. La masse des molécules est égale à la somme des masses de l'ensemble de ses atomes. Quant au symbole de ces éléments, ils sont composés à partir de leur composition atomique. Ainsi, l'eau est composée de 2 atomes d'hydrogène et d'un atome d'oxygène, sa formule est donc « H2O ». Si vous êtes motivé, vous pouvez implémenter aussi le nom des molécules (« Oxyde de dihydrogène » pour l'eau).

Assurez de rendre la création des atomes la plus simple possible : on entend par là que le constructeur doit comporter le moins possible d'arguments. N'oubliez pas qu'au sein de l'univers, vos atomes et vos molécules doivent être parfaitement substituables en comportement.

Question

Nous allons bientôt afficher notre univers, mais nous allons également tenter de coder le déplacement des éléments avant de lancer le programme.

Pour cela, vous allez extraire de l'interface des éléments tout ce qui attrait au déplacement (propriétés, méthodes, events ...) dans une autre interface IMovable.

Vous allez faire un peu de refactoring pour isoler tout le comportement de déplacement. Ajoutez ensuite à l'interface créée une propriété pour matérialiser la vitesse de déplacement de l'élément dans l'espace, et une méthode UpdateLocation() ne retournant rien.

Vous ajouterez ensuite une stratégie de déplacement (qui dépendra de votre nouvelle interface uniquement). Cette stratégie sera appelé par la méthode UpdateLocation() de l'élément, et déplacera l'élément dans l'univers en prenant en compte son vecteur de déplacement. Quand les éléments atteindront les limites de l'univers, le vecteur sera inversé pour qu'ils ne puissent en sortir. Vous ferez ensuite une autre stratégie qui ne se bornera pas aux limites de l'univers : elle fera réapparaître les éléments à droite dès qu'ils sortent à gauche par exemple.

Question

Nous allons maintenant rajouter une représentation visuelle à notre application console. Faites-en sorte d'afficher l'univers et ses éléments dans la console en ASCII sous la forme d'un tableau qui sera rafraîchi.

On va ensuite rajouter un moteur, qui donnera vie à l'univers. Ce moteur donnera le tempo, les éléments auront donc effectué des déplacements à chaque pas de temps. On implémentera un moteur avec un Thread, avec une boucle infinie et un temps d'attente entre chaque mise à jour. Ce temps d'attente devra être configurable.

Attention, le moteur de rendu et le moteur de tempo devront uniquement dépendre d'interfaces.

Écriture du Main de l'application. Il s'articule autour des étapes suivantes.

1. Création de l'univers avec ses dimensions
2. Peupler des éléments dans l'univers
3. Création du moteur d'affichage, en lui passant l'univers en paramètre
4. Création du moteur de tempo, en lui passant le moteur d'affichage et le délai
5. Lancement du thread

Alors, ça marche ?

▼ Ressources pour les étudiants

