Cartouche du document

Année : ING 2
Matière : Conception
Activité : Travail dirigé

Objectifs

Dans un premier temps, on étudie le premier principe de la COO : Programmer pour une interface, non pour un développement particulier.

Ensuite on s'interesse à la création des objets. **Bien concevoir commence par bien créer les objets que l'on va utiliser dans nos différentes briques logicielles**

On étudie dans cet exercice un ensemble de patterns dont le rôle est **Créateur** (Ce rôle désigne tous les patterns dont le rôle est de créer des objets.) dans la terminologie de **GoF** (Ce terme est une abréviation de l'expression anglaise **Gang of four**. Il désigne quatre personnes connues dans le monde de la conception car ils ont créés les principaux design patterns.

- Erich Gamma
- Richard Helm
- Ralph Johnson
- John Vlissides

) .

Sommaire des exercices

Corps des exercices

1 - Interface : Passer de la programmation spécifique à la programmation générique

Énoncé :

Dans le projet Mini Système Expert, on a le diagramme de classe suivant :

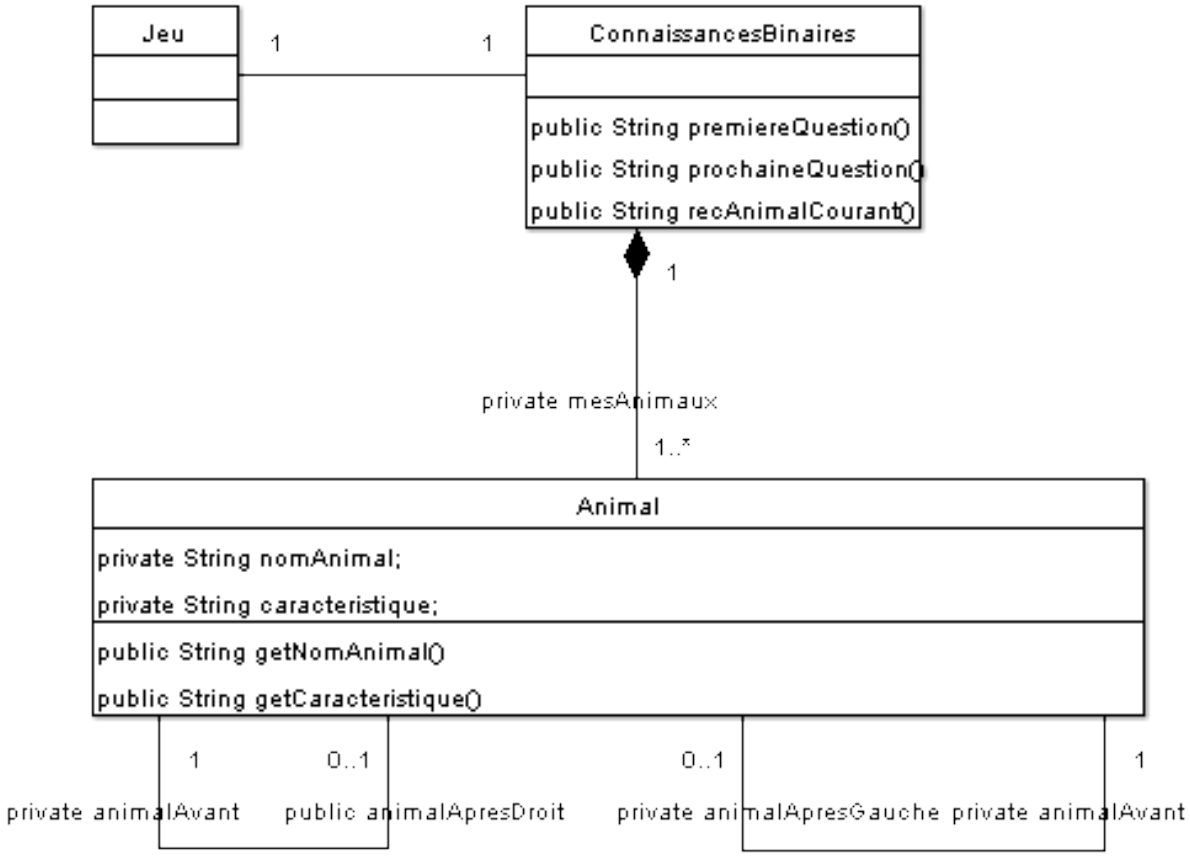

Diagramme de classes spécifique : Jeu, Connaissances et animaux

Question 1)

Énoncé de la question

Ce diagramme de classes est spécifique à certain mode de structuration des connaissances. On vous demande de définir un autre diagramme qui permet de découpler la classe Jeu et la classe ConnaissancesBinaires afin d'envisager un autre mode de structuration des connaissances.

Question 2)

Énoncé de la question

Que faut-il ajouter dans le diagramme de classes pour que l'on puisse créer l'objet de connaissances sans que le client Jeu connaisse le type de cet objet ?

2 - Pattern singleton

Énoncé :

On s'interesse au **Design Pattern Singleton** (Ce pattern garantit qu'on ne peut instancier qu'un seul objet d'une classe.

C'est un pattern créateur.) .

Question 1)

Énoncé de la question

Conception

Donner la définition complète de ce Design Pattern.

Question 2)

Énoncé de la question

Langage Java

Donner une solution générique de ce pattern en Java.

Question 3)

Énoncé de la question

Conception

Dans une application qui fait entre autre référence à des données persistantes, donnez un exemple d'utilisation de ce Pattern pour accéder à la base de données.

Question 4)

Énoncé de la question

Conception

D'un point de vue plus général, quand on est confronté à un problème de centralisation d'informations, on vous demande d'expliquer comment le pattern Singleton peut servir à la résolution de ce problème.

3 - Les patterns de fabrique

Énoncé :

L'objectif de cet exercice est de montrer comment on conçoit une application qui permet à des objets X de créer et d'utiliser d'autres objets Y sans que X connaissent les classes des différents objets Y.

Question 1)

Énoncé de la question

Conception

On désire concevoir un jeu vidéo qui simule le déplacement de voitures.

- doit pouvoir demarrer
- doit pouvoir s'arreter
- doit pouvoir accelerer
- doit pouvoir freiner
- doit pouvoir tourner

• pouvoir fabriquer des voitures sans en connaître leur type. Il doit donc déléguer la fabrication à une sous classe.
• savoir utiliser ces voitures (pour simuler leur comportement) toujours sans en connaître leur type

Donner une solution à ce problème en français et en UML en supposant que l'on utilise des ferraris comme voitures. .

Question 2)

Énoncé de la question

Conception

La question précédente est un cas particulier du pattern **Fabrication** (**L'intention de ce pattern** est de définir une classe abstraite X qui déclare la méthode de fabrication des objets qu'elle utilise en laissant à une sous classe le soin de créer réellement ces objets. Cette classe X implémente l'utilisation des objets.

La méthode de fabrication permet de déléguer l'instanciation d'objet à des sous classes.

Les constituants sont

- Une interface Produit qui déclarent les opérations que l'on peut faire avec les objets créés.
- Une classe ProduitConcret qui implémente l'interface Produit.
- Une classe Facteur qui déclare la méthode de fabrication des produits et qui implémente les méthodes d'utilisation des produits en se conformant à l'interface Produit.
- Une classe FacteurConcret qui dérive de la classe Facteur et implémente la méthode de fabrication des produits.

C'est un pattern créateur.) .

Définir la solution générique de ce pattern en UML

Question 3)

Énoncé de la question

Langage Java

On vous demande d'implémenter une solution générique en Java du pattern.

Question 4)

Énoncé de la question

Conception

On s'interesse maintenant au pattern de **fabrique abstraite** (**L'intention de ce pattern** est de définir une interface pour la création d'une famille d'objets interdépendants sans avoir à spécifier leurs classes concrètes.

Les constituants sont

- Une interface dite fabrique abstraite qui déclare les opérations de créations des différents types d'objets abstraits.
- Une classe qui implémente la fabrique abstraite.
- Plusieurs interfaces dites "produit abstrait" qui déclarent ce que l'on peut faire avec les produits.
- Plusieurs classes qui implémentent les interfaces "produit abstrait".
- Une classe Client qui n'utilise que les interfaces fabrique abstraite et produit abstrait.

C'est un pattern créateur.) .

Question 5)

Énoncé de la question

Langage Java

On vous demande une implémentation générique en Java le pattern de **fabrique abstraite** (**L'intention de ce pattern** est de définir une interface pour la création d'une famille d'objets interdépendants sans avoir à spécifier leurs classes concrètes.

Les constituants sont

- Une interface dite fabrique abstraite qui déclare les opérations de créations des différents types d'objets abstraits.
- Une classe qui implémente la fabrique abstraite.
- Plusieurs interfaces dites "produit abstrait" qui déclarent ce que l'on peut faire avec les produits.
- Plusieurs classes qui implémentent les interfaces "produit abstrait".
- Une classe Client qui n'utilise que les interfaces fabrique abstraite et produit abstrait.

C'est un pattern créateur.) .

4 - Pattern Monteur

Énoncé :

On s'intéresse au **Design PatternMonteur** (**L'intention de ce pattern** est de dissocier la construction d'un objet complexe de sa représentation interne. Ainsi le même processus de construction permet des représentations différentes.

Les constituants sont

- Une interface dite Monteur qui déclare les opérations de création des parties de l'objet complexe.
- Une classe MonteurConcret qui
  - implémente l'interface Monteur pour la fabrication des parties de l'objet.
  - définit la représentation interne de l'objet qu'il crée et stocke cette représentation.
  - fournit une interface de récupération de cette représention à l'objet complexe.
- Une classe Directeur qui construit un objet en utilisant l'interface Monteur.
- Une classe Produit dont un objet représente l'objet complexe en cours de construction.

C'est un pattern créateur.) .

On va dans un premier temps s'interesser à un exemple pour mieux comprendre le principe. Ensuite on devra généraliser pour modèliser complètement le pattern en UML.

Question 1)

Énoncé de la question

Conception

Un lecteur de format d'échange de documents

RTF (Rich Text Format) est un format d'échange de documents. Un lecteur pour ce format d'échange est un programme qui permet de convertir des fichiers au format RTF en de nombreux autres formats comme par exemple

- format texte ASCII
- format word
- format T_EX...

Chaque document résultat est un objet complexe formé donc de parties. D'un format à l'autre, les parties sont différentes et en conséquence l'assemblage de ces parties.

Pour être plus précis, un document RTF est composé de signes. Un signe peut être soit du texte pur et soit un mot de contrôle RTF. Lorsque que le lecteur RTF reconnaît un signe, il emet une requête au convertisseur de texte afin qu'il convertisse ce signe. Le convertisseur doit effectuer la conversion de données et représenter le signe dans un format particulier.

Par exemple, un convertisseur de texte ASCII ignorera toutes les requêtes sauf celle relative à un signe de texte pur. Un convertisseur T_EX implémentera en plus des requêtes de texte pur toutes les requêtes qui peuvent produire une représentation T_EX.

Toutes les conversions

- ont en commun la lecture signe par signe du document en entrée
- diffèrent dans la conversion des différents signes

En tenant compte de cette remarque, on vous demande de trouver une solution exprimée en UML qui permet de distinguer la construction du document résultat de sa représentation interne.

Pour se fixer les idées, on distinguera trois signes

- le signe de texte pur
- le signe de contrôle qui indique un changement de police
- le signe de contrôle qui indique un nouveau paragraphe

Question 2)

Énoncé de la question

Conception

Donner en UML, la définition complète du **Design Pattern Monteur** (**L'intention de ce pattern** est de dissocier la construction d'un objet complexe de sa représentation interne. Ainsi le même processus de construction permet des représentations différentes.

Les constituants sont

- Une interface dite Monteur qui déclare les opérations de création des parties de l'objet complexe.
- Une classe MonteurConcret qui
  - implémente l'interface Monteur pour la fabrication des parties de l'objet.
  - définit la représentation interne de l'objet qu'il crée et stocke cette représentation.
  - fournit une interface de récupération de cette représention à l'objet complexe.
- Une classe Directeur qui construit un objet en utilisant l'interface Monteur.
- Une classe Produit dont un objet représente l'objet complexe en cours de construction.

C'est un pattern créateur.) .

Question 3)

Énoncé de la question

Langage Java

Donner une implémentation générique de ce pattern en Java.