Projet informatique intégré

Bloc 2

Haute-École de Namur-Liège-Luxembourg

Design Patterns

- Exercices -

Objectifs

- Identifier le Design Pattern à appliquer dans un contexte donné
- Appliquer un Design Pattern donné et construire le diagramme de classes correspondant
- Implémenter en Java un Design Pattern

Exercice 1 : Incident sur site industriel de type Seveso[1]

Soit un site industriel de type Seveso qui regroupe plusieurs usines, fabriques et entreprises de production d'énergie.

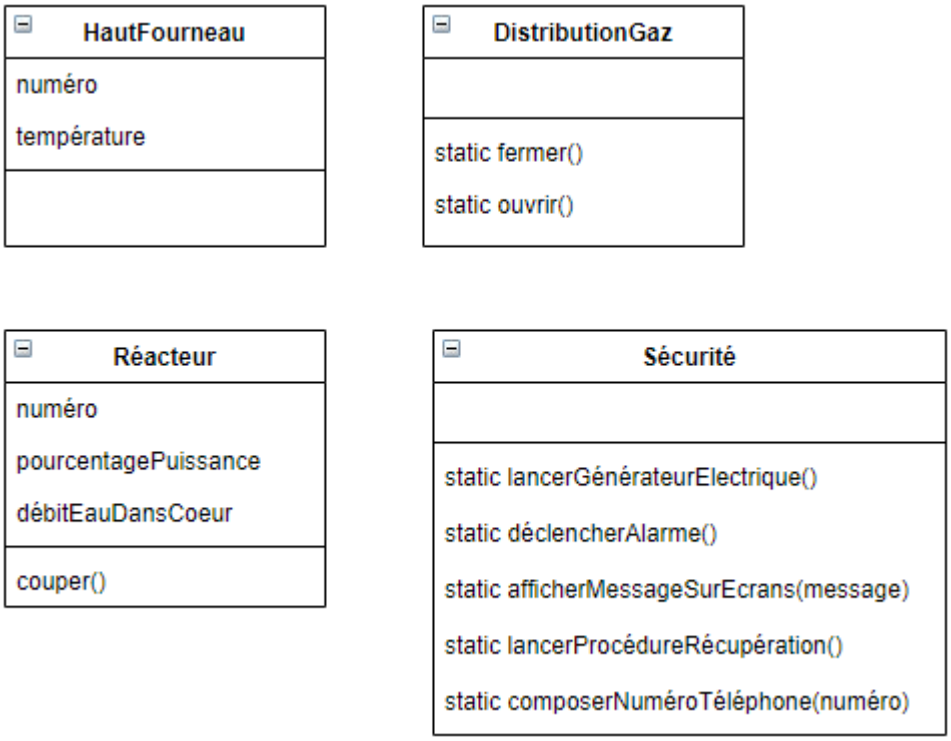
En cas d'accident sur le site, une procédure complexe doit être appliquée de façon stricte ; les différentes étapes de la procédure doivent impérativement être effectuées dans un ordre précis.

La séquence des opérations de sécurisation du site en cas d'accident est la suivante :

1. Couper les réacteurs 1 et 3
2. Couper la distribution de gaz
3. Diminuer à 10% la puissance du réacteur 2
4. Augmenter de 50% le débit de l'eau dans le cœur du réacteur 2 (pour accélérer le refroidissement)
5. Baisser la température du haut fourneau numéro 1 à 200^0 C
6. Déclencher l'alarme
7. Afficher sur tous les écrans le message « Évacuation immédiate »
8. Composer le numéro d'urgence 112
9. Lancer le générateur électrique
10. Lancer la procédure de récupération

[1] Les sites industriels présentant des risques d'accidents majeurs sont appelés « sites Seveso » en référence à la catastrophe qui a eu lieu en 1976 à Seveso en Italie. Le niveau de prévention y est élevé et les procédures en cas d'accidents sont strictes.

Soit le diagramme de classes :



Question 1

Quel Design Pattern appliqueriez-vous ?

Question 2

Implémenter en java la classe qui découlerait de l'application de ce Design Pattern.

Exercice 2. Catégories d’abonnement

Une entreprise de services propose à ses clients plusieurs catégories d’abonnement, chaque abonnement donnant accès à plusieurs services.

Un client ne peut acheter qu'un seul abonnement et a accès aux services correspondants.

Question 1

Quel Design Pattern vous aiderait à permettre les accès aux services auxquels ont droit les clients ?

Question 2

Donnez le diagramme de classes adapté à ce cas.

Exercice 3. Organisation des fichiers en dossiers

Les fichiers sont rassemblés en dossier.

Tout dossier peut contenir 0, un ou plusieurs fichier(s) et 0, un ou plusieurs dossier(s).

Question 1

Quel Design Pattern appliqueriez-vous pour aider à organiser les fichiers en dossiers ?

Question 2

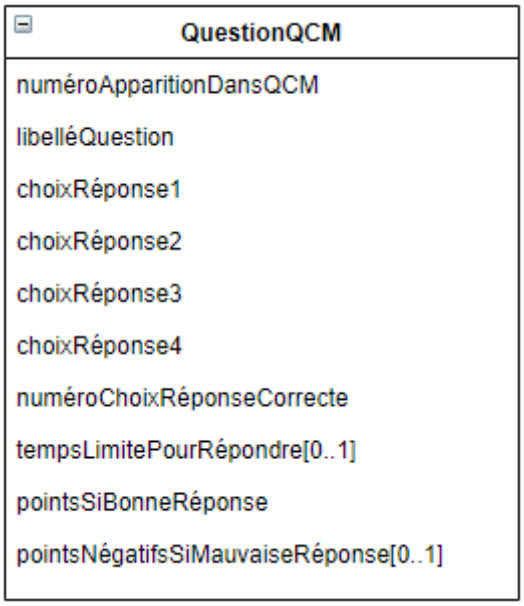
Donnez le diagramme de classes adapté à ce cas.

Exercice 4. Production de QCM

Dans un centre de compétences, des formations sont proposées à des apprenants qui sont ensuite évalués à la fin de leur formation à travers des QCM. Pour ce faire, les formateurs doivent régulièrement produire plusieurs séries de QCM équivalents en termes de difficulté. Ils reprennent en général les mêmes questions en les modifiant éventuellement légèrement.

Les formateurs disposent d'une large bibliothèque de questions pour QCM. Chaque question propose 4 choix de réponse, une seule réponse étant correcte.

Soit la classe *QuestionQCM* :



Les formateurs souhaitent donc disposer d'un moyen aisé de créer des nouvelles questions à partir de questions existantes, qu'ils reprendront telles quelles ou qu'ils pourront ensuite modifier légèrement s'ils le souhaitent.

Question 1

Quel Design Pattern appliqueriez-vous ?

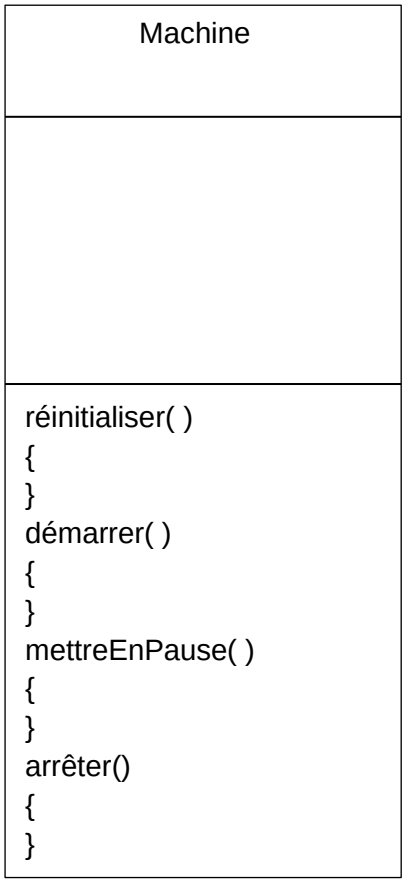
Question 2

Implémenter en java ce Design Pattern dans la classe *QuestionQCM*.

Exercice 5. Gestion d'un parc de machines

Une entreprise doit gérer un parc de machines qui ont trois modes de fonctionnement : mode normal, mode éco et mode rapide. On peut réinitialiser, démarrer, mettre en pause ou arrêter chacune de ces machines.

Soit une première ébauche de la classe *Machine* :



Question 1

Quel Design Pattern appliqueriez-vous sachant que l'algorithme des méthodes *réinitialiser, démarrer, mettreEnPause* et *arrêter* varie en fonction de l'état courant (normal, éco ou rapide) ?

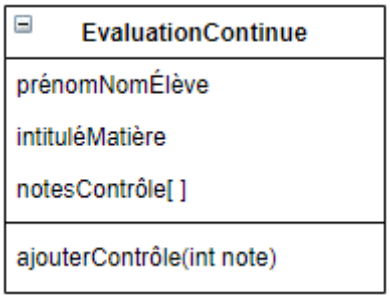
Question 2

Compléter le diagramme ci-dessus en appliquant le Design Pattern choisi.

Exercice 6. Evaluation continue

Dans une école, un enseignant évalue ses élèves sur une matière sous forme de contrôles (interrogations) réguliers ; une note est attribuée par l'enseignant aux élèves pour chaque contrôle.

Soit la classe *EvaluationContinue* :



N.B. La classe EvaluationContinue ne contient qu'un seul constructeur qui permet de créer une évaluation sans aucun contrôle (la collection notesContrôle est vide).

Question 1

Quel Design Pattern appliqueriez-vous pour offrir au programmeur qui utilise la classe *EvaluationContinue* la possibilité de boucler sur la collection des notes des contrôles ?

Question 2

Implémenter en java ce Design Pattern dans la classe *EvaluationContinue*, en implémentant la collection *notesContrôles* sous forme de tableau.

Question 3

Implémenter en java ce Design Pattern dans la classe *EvaluationContinue*, en implémentant la collection *notesContrôles* sous forme de ArrayList.

Exercice 7. Manipulation d'objets en 3 dimensions

Un programme doit manipuler des objets en 3 dimensions, à savoir, des cubes, des sphères, des cylindres et des pyramides. Quatre classes sont donc créées : *Cube, Sphere, Cylinder* et *Pyramid*.

Chaque manipulation d'un objet 3D nécessite trois étapes : une étape d'initialisation, une étape d'affichage et une étape de finition. Les étapes d'initialisation et de finition sont des étapes longues et complexes mais leurs algorithmes sont les mêmes pour tous les objets 3D ; seul l'algorithme d'affichage diffère d'un objet 3D à l'autre.

Voici le code des 4 classes :

```
public class Cube {
   public void manipulation() {
         initialisation() ;
         … // code d'affichage du cube
         finishing() ;
   }
   public void initialisation() {
         …
  }
   public void finishing() {
         …
  }
}
```

```
public class Sphere {
   public void manipulation() {
         initialisation() ;
         …  // code d'affichage de la sphère
         finishing() ;
   }
   public void initialisation() {
         …
  }
   public void finishing() {
         …
  }
}
```

```
public class Cylinder {
   public void manipulation() {
         initialisation() ;
         …  // code d'affichage du cylindre
         finishing() ;
   }
   public void initialisation() {
         …
  }
   public void finishing() {
         …
  }
}

public class Pyramid {
   public void manipulation() {
         initialisation() ;
         … // code d'affichage de la pyramide
         finishing() ;
   }
   public void initialisation() {
         …
  }
   public void finishing() {
         …
  }
}
```

Question 1

Quel Design Pattern appliqueriez-vous pour éviter la redondance de code ?

Question 2

Donnez le diagramme de classe de ce Design pattern appliqué au cas traité.

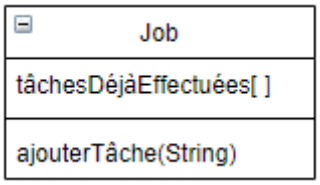
Question 3

Corrigez le code des classes Java en appliquant ce Design Pattern.

Exercice 8. Récupération des tâches déjà effectuées

Une entreprise a structuré ses travaux sous forme de jobs. Il est important pour l'entreprise de pouvoir retrouver le plus précisément possible la liste des tâches déjà effectuées pour chaque job.

Soit la classe *Job* :



L'entreprise souhaite lancer à intervalles réguliers une procédure de sauvegarde des tâches déjà effectuées sur chaque job en vue de pouvoir restaurer une telle liste en cas de suppression malencontreuse de la liste des tâches déjà réalisées sur un job donné.

Question 1

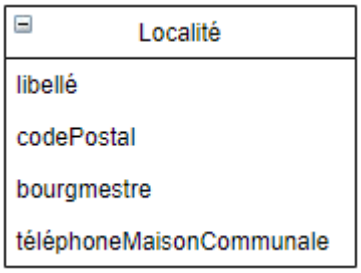
Quel Design Pattern appliqueriez-vous ?

Question 2

Implémenter en java ce Design Pattern, tout en simulant dans la méthode main la récupération de la dernière liste des tâches réalisées sur un job donné.

Exercice 9. Localités de Belgique

Un programme doit gérer une quantité non négligeable d'adresses en Belgique et donc créer des objets de la classe Localité que voici :



Bon nombre des clients habitent une même localité. Par conséquent, une fois la localité créée, les programmeurs souhaitent pouvoir récupérer et utiliser ce même objet pour tous les futurs clients qui habitent cette localité. Ils souhaitent ainsi ne pas devoir réécrire le nom du bourgmestre et le numéro du central téléphonique de la maison communale des localités qu'ils ont déjà créées précédemment dans le programme.

Question 1

Quel Design Pattern appliqueriez-vous ?

Question 2

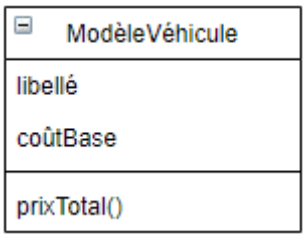
Implémenter en java ce Design Pattern.

Exercice 10. Choix des options d'un véhicule

Un garage vend différents modèles de véhicule de la même marque.

A chacun des modèles peuvent être ajoutées des options choisies par l'acheteur.

Soit la classe *ModèleVéhicule* :



Question 1

Quel Design Pattern appliqueriez-vous afin que la méthode *prixTotal()* prenne en compte dans son calcul les suppléments de prix des options éventuelles ajoutées par l'acheteur.

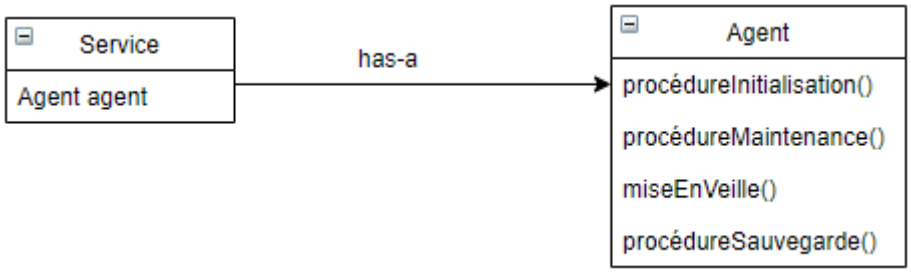
Question 2

Complétez le diagramme de classe en appliquant ce Design Pattern.

Exercice 11. Nouvelle version disponible

Une classe *Service* appelle les méthodes d'un objet de type *Agent*.

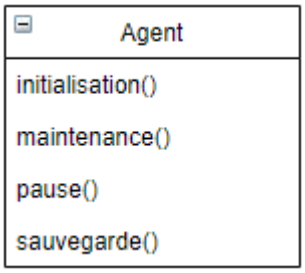
Soit le diagramme suivant :



La classe *Service* possède beaucoup de méthodes (non représentées sur le schéma ci-dessus dans un souci de visibilité) ; bon nombre d'entre elles appelant par délégation une ou plusieurs méthodes de la classe *Agent* via la variable d'instance *agent* (ex : *agent.miseEnVeille()*).

Or, une nouvelle version de la classe *Agent* est disponible proposant des méthodes plus performantes. Les programmeurs de la classe *Service* souhaitent profiter de cette nouvelle version, car les performances de la classe *Service* seront augmentées elles aussi sensiblement. Malheureusement, les noms des méthodes de la nouvelle version de la classe *Agent* ont été modifiés.

Soit la nouvelle classe *Agent* :



Modifier les appels de méthodes correspondant directement dans le code de la classe *Service* est impensable car cela nécessiterait la révision d'un nombre trop important de lignes de code.

Question 1

Quel Design Pattern appliqueriez-vous afin de faciliter ce travail de mise à niveau ?

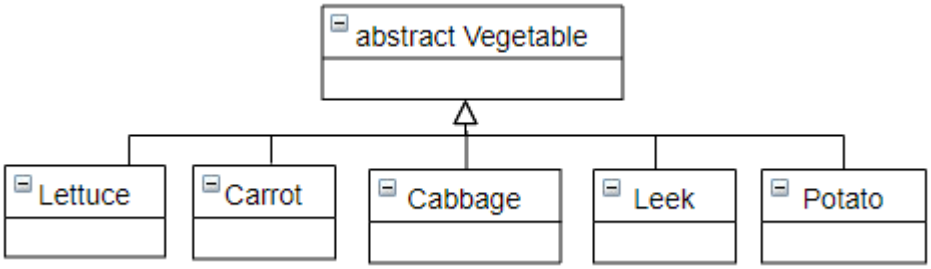
Question 2

Complétez le diagramme de classe en appliquant ce Design Pattern.

Exercice 12. Des légumes bien de chez nous

Un maraîcher sème et récolte des pommes de terre, des choux, des carottes, des salades et des poireaux.

Soit la hiérarchie de classes suivante :



Voici le code de la classe Maraîcher (MarketGardener) qu'un étudiant en informatique a tenté d'écrire et qui contient les méthodes semer (sow) et récolter (harvest).

```
public class MarketGardener {

    public void sow (char code) {
        ...     // portion de code 1
        Vegetable vegetable;
        switch (code) {
            case 'A': vegetable = new Cabbage();
            case 'B': vegetable = new Carrot();
            case 'C': vegetable = new Leek();
            default :  vegetable = new Potato();
        }
        ...     // portion de code 2
    }

    public Vegetable harvest (char code) {
        ...     // portion de code 3
        Vegetable vegetable;
        switch (code) {
            case 'A': vegetable = new Cabbage();
            case 'B': vegetable = new Carrot();
            case 'C': vegetable = new Leek();
            default :  vegetable = new Potato();
        }
        ...     // portion de code 4
    }

}
```

Question 1

Quel Design Pattern appliqueriez-vous pour optimiser le code proposé ci-dessus ?

Question 2

Créez le diagramme de classe correspondant à ce Design Pattern appliqué au cas traité.

Question 3

Réécrivez la classe *MarketGardener* sur base de ce Design Pattern.

Exercice 13. Il ne peut y avoir qu'un seul roi

Corrigez le code de la classe ci-dessous sachant que la classe *King* résulte de l'application du Design Pattern Singleton.

```
public class King {
   King uniqueInstance;
   public King ( ) {
        … // code du constructeur
   }
   public uniqueInstance getOccurence( ) {
        if ( uniqueInstance = = null )
              { uniqueInstance = new King( ) ; }
        return uniqueInstance;
   }
}
```