Université Claude Bernard Lyon 1
Polytech Lyon - Département Informatique

4A , 2019–2020

ISI-3

TD4. Design Patterns

Laëtitia Matignon

---

1 Modélisation d'une unité d'un jeu

On s'intéresse à la modélisation d'une unité d'un jeu (par exemple un soldat). L'unité a des capacités de déplacement et de combat modifiables par le joueur. Ainsi, l'unité peut combattre à mains nues ou utiliser un couteau ou un fusil ; l'unité peut marcher ou courir ou encore utiliser une moto.

Question 1 *Proposer une solution pour modéliser cette unité aux comportements interchangeables. Préciser le pattern utilisé, donner le diagramme de classes en détaillant en pseudo-code les éléments nécessaires à la mise en place du pattern.*

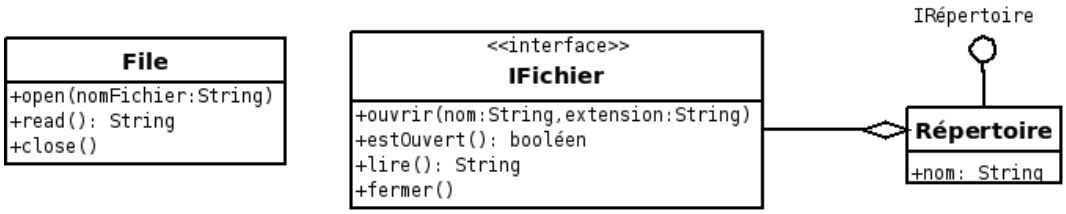
2 Système de vente de véhicules

Au sein d'un système de vente de véhicules, nous voulons représenter des sociétés notamment pour leur proposer des offres de maintenance de leur parc de véhicules. Chaque société possède un parc de véhicule et un cout d'entretien par véhicule. Les sociétés représentées peuvent être avec ou sans filiales. Une société avec filiales peut avoir comme filiale des sociétés elle-mêmes avec filiales. On souhaite pouvoir ajouter des véhicules aux sociétés et calculer le coût d'entretien du parc d'une société (fonction du nombre de véhicules du parc et du coût d'entretien unitaire d'un véhicule). Dans le cas d'une société avec filiales, le cout d'entretien de cette société prend en compte le cout d'entretien de ses filiales. Les sociétés avec filiales ont aussi des attributs et méthodes spécifiques comme par exemple le pourcentage d'actions détenues par la société mère sur chacune de ses filiales.

Question 2 *Proposer une solution pour modéliser ces sociétés. Préciser le pattern utilisé, donner le diagramme de classes en détaillant en pseudo-code les éléments nécessaires à la mise en place du pattern.*

3 Répertoires et fichiers

On dispose des classes présentées sur la figure suivante :



Une application gère ses répertoires avec la classe `Répertoire`. Les fichiers gérés par la classe `Répertoire` doivent implémenter l'interface `IFichier`.

Question 3 *On souhaite intégrer dans l'application une gestion de répertoires via des fichiers de type* `File`*. Proposer une architecture permettant cela en maximisant l'utilisation du code déjà existant (tous les éléments donnés dans la figure sont non-modifiables). On précisera le pattern utilisé, le diagramme de classes en détaillant en pseudo-code les éléments nécessaires à la mise en place du pattern.*

Question 4 *On souhaite maintenant prendre en compte une classe abstraite* `FichierAbstrait` *à la place de l'interface* `IFichier`*. Modifier en conséquence (si besoin) votre architecture.*

4 Pilote automatique

On s'intéresse à un pilote automatique de voiture. Ce pilote doit adapter sa conduite (réaction aux obstacles, virages, distance de sécurité, etc.) aux conditions météo suivantes : pluie ou neige. Cela se traduit par des définitions différentes des méthodes `traiter(obstacle)`, `tourner(angle)`, etc. de la classe `PiloteAuto`.

Des capteurs (classes `CapteurPluie`, `CapteurNeige`) permettent au composant `ConditionMeteo` de savoir dans quelles conditions la voiture évolue à tout instant. `ConditionMeteo` se place comme écouteur d'événements générés par les capteurs et retourne des informations sur leur état *via* les méthodes `getPluie()`, `getNeige()`).

Question 5 *Quel(s) design(s) pattern(s) peut-on utiliser pour faire en sorte que le pilote automatique adapte son comportement en fonction des conditions ? Modéliser à l'aide d'un diagramme de classes en spécifiant les implémentations de certaines méthodes.*

Le pilote automatique doit maintenant adapter sa conduite à de nombreuses conditions météos (vent, pluie, neige) mais aussi à la période de la journée (jour, nuit, crépuscule, ...). Des combinaisons de ces conditions sont donc à envisager.

Question 6 *Modifier votre modélisation pour prendre en compte ces nouveaux paramètres.*

5 Robocup Soccer Simulation

Dans cette partie, nous allons explorer l'utilisation des *design pattern* dans le contexte d'une application similaire à celle de la *Robocup Soccer Simulation*. La *RoboCup Soccer*[1] est un tournoi international de robotique qui a lieu tous les ans. L'un des objectifs de ce tournoi est d'arriver à créer une équipe de football robotisée (cf. figure 1(c)) capable de battre l'équipe de football "humaine" championne du monde d'ici à 2050. Une des divisions de la *RoboCup Soccer* est la ligue *Simulation*[2] qui permet de simuler des parties de football entre deux équipes de 11 joueurs robots simulés (cf. figure 1(a) et (b)).

(a) Simulation 2D (b) Simulation 3D (c) Plateforme Standard (robots Nao)

FIGURE 1 – Exemples de divisions dans la RoboCup Soccer

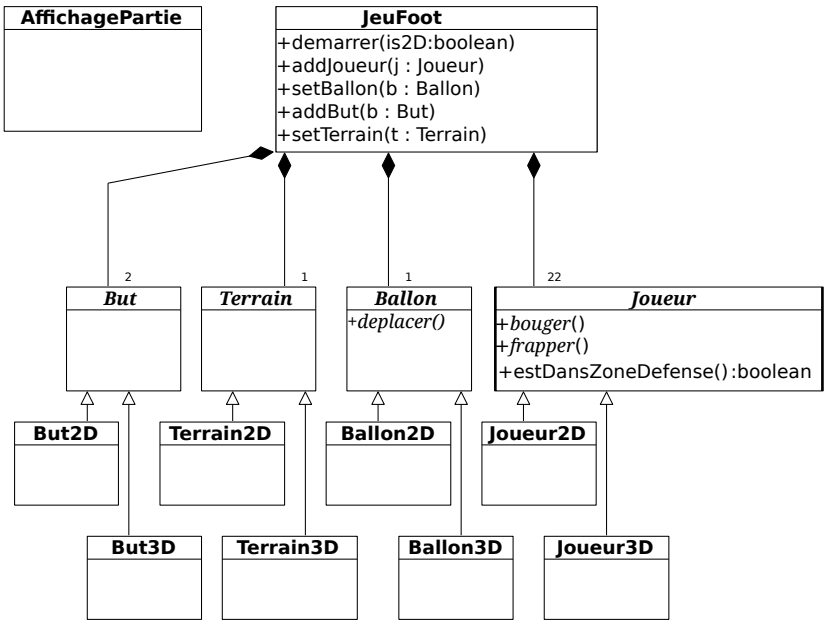


FIGURE 2 – Diagramme de classes (partiel) de l'application

Dans l'application de simulation que l'on souhaite développer ici, le jeu de football est représenté par une classe concrète `JeuFoot` composée des classes abstraites suivantes :

1. http ://www.robocup2014.org/
2. http ://wiki.robocup.org/wiki/Soccer_Simulation_League

`Joueur`, `Ballon`, `Terrain`, `But`. La classe concrète `AffichagePartie` gère l'affichage graphique de la partie. Les différentes composantes du jeu de foot peuvent être créées comme des éléments 2D ou 3D. La figure 2 propose un diagramme de classes partiel pour cette application.

La création d'un jeu 2D ou 3D dépend du choix de l'utilisateur lorsqu'il démarre une partie (méthode *demarrer(boolean is2D)* de la classe `JeuFoot`). On souhaite encapsuler l'instanciation des différents types d'objets dans des classes dédiées.

Question 7 *Pour cela, proposer un ou plusieurs patterns et compléter le diagramme de classe donné à la figure 2. Vous préciserez l'implémentation des classes ajoutées ainsi que de la méthode* `demarrer(boolean is2D)`.

L'interface publique de la classe `Joueur` offre deux comportements de base (qui sont implantés par les classes concrètes `Joueur2D` et `Joueur3D`) : *bouger()* (pour déplacer le joueur) et *frapper()* (pour taper le ballon).

On s'intéresse aux joueurs 2D. Le comportement d'un joueur 2D est spécifique selon sa position sur le terrain : s'il est dans la zone de défense de son équipe, il agit comme un défenseur ; sinon il agit comme un attaquant.

Question 8 *Pour réaliser cette fonctionnalité. proposer un ou plusieurs patterns et compléter le diagramme de classe donné à la figure 2. Vous préciserez l'implémentation des classes ajoutées.*

La classe `AffichagePartie` est responsable d'afficher la position des joueurs et du ballon sur le terrain (ne vous souciez pas de la manière avec laquelle cet affichage s'effectue).

Question 9 *Pour réaliser cette fonctionnalité. proposer un ou plusieurs patterns et compléter le diagramme de classe donné à la figure 2.*