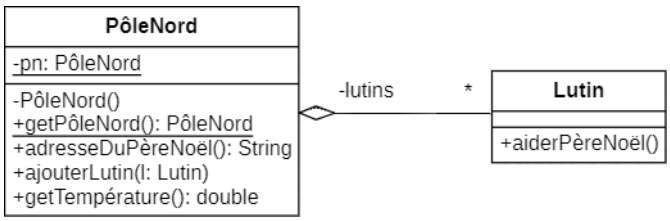
Examen de *Design Patterns*

2022–2023

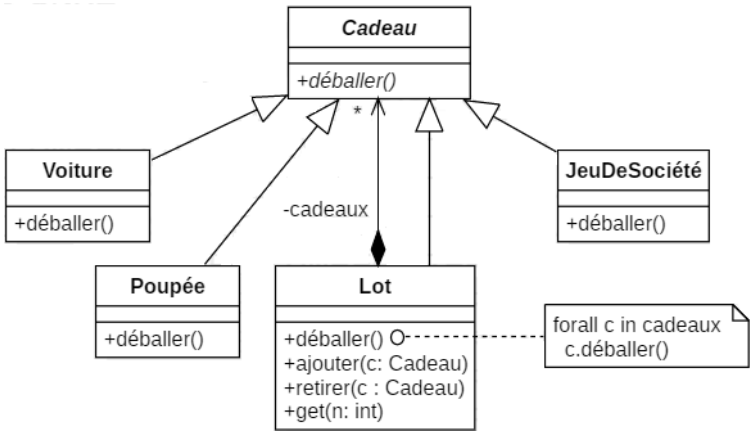
— Durée : 1h30
— Type : papier
— Aucun document autorisé.
— Toutes vos affaires (sacs, vestes, *etc.*) doivent être placées à l'avant de la salle.
— Aucun téléphone ne doit se trouver sur vous ou à proximité, même éteint.
— Les déplacements et les échanges ne sont pas autorisés.
— Aucune question ne peut être posée aux enseignants, posez des hypothèses en cas de doute.

Question de cours (5pts)

1. Vrai ou faux ? (2pts)
   (a) Un *design pattern* offre une implémentation qui solutionne un problème récurrent.
   (b) Le *design pattern* Décorateur peut être implémenté sans le polymorphisme.
   (c) Il n'y a aucun impact sur les clients d'une façade lors d'un changement du sous-système.
   (d) Un observateur ne peut pas observer plus d'un sujet.
2. Quel est la différence entre un itérateur interne et un itérateur externe ? (1pt)
3. Quel *design pattern* définit une famille d'algorithmes encapsulés dans des objets, permettant ainsi de les interchanger durant l'exécution ? (0.5pt)
4. Quel *design pattern* encapsule une requête dans un objet, permettant ainsi de gérer un historique de requêtes ? (0.5pt)
5. Quel *design pattern* est illustré ici ? (0.5pt)



6. Quel *design pattern* est illustré ici ? (0.5pt)



Avatar : la voie des *Design Patterns* (15pts)

N.B. **: les phrases en italique correspondent à des dialogues de film (parfois adaptées), elles ne sont pas utiles pour la compréhension et la résolution des problèmes.**

Vous n'êtes plus à CY Tech, vous êtes sur Pandora !

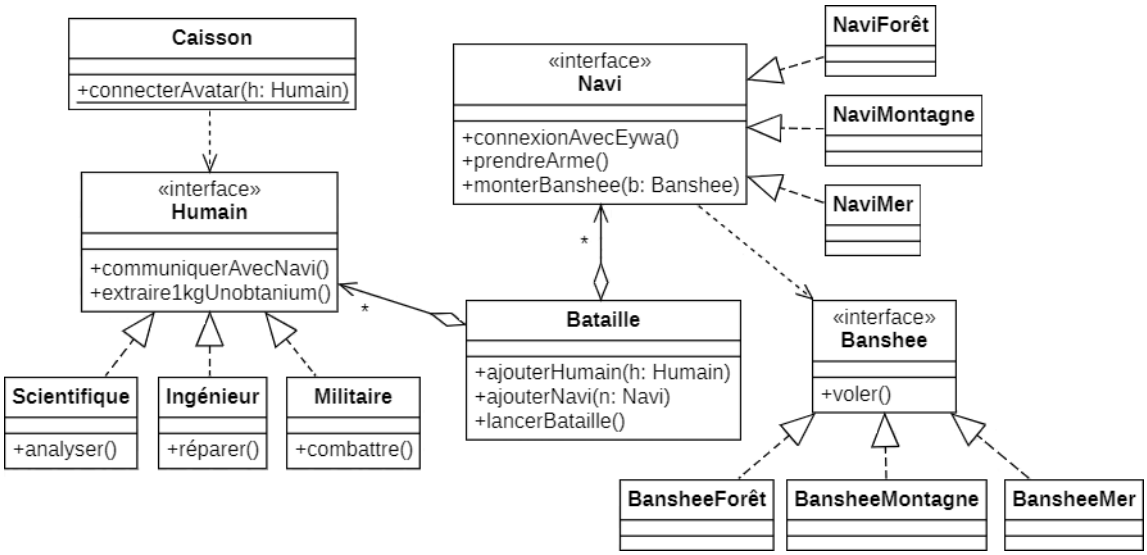


FIGURE 1 – Diagramme de classes *Avatar*

Pour la sortie au cinéma de la suite d'*Avatar*, James CAMERON souhaite développer un jeu vidéo dans cet univers. Pour cela, il dispose déjà de plusieurs classes (sans les sources), illustrées à la figure 1, représentant différents types d'humain (*i.e.*, scientifiques, ingénieurs, militaires), de Na'vi (des autochtones de Pandora) et de Banshee (des créatures de Pandora). La classe `Caisson` contient une méthode `connecterAvatar(Humain)` permettant de connecter un humain (passé en paramètre) à son avatar. Une fois connectés, les humains communiquent par l'intermédiaire des avatars avec les Na'vis afin de gagner leur confiance dans le but d'extraire l'Unobtanium présent sur Pandora, un minerai qui pourrait résoudre la crise énergétique sur Terre. Les méthodes `communiquerAvecNavi()` et `extraire1kgUnobtanium()` de l'interface `Humain` sont appelées un nombre aléatoire de fois par la méthode `connecterAvatar(Humain)` de la classe `Caisson`. Un exemple d'utilisation de ces classes, ainsi que les logs obtenus dans la console, sont donnés ci-dessous :

```
Caisson.connecterAvatar(new Scientifique());
Caisson.connecterAvatar(new Ingenieur());
Caisson.connecterAvatar(new Militaire());
Caisson.connecterAvatar(new Scientifique());
```

```
>Console :
Un scientifique communique avec les Na'vis.
Un scientifique communique avec les Na'vis.
Un ingénieur communique avec les Na'vis.
Un ingénieur extrait 1kg d'Unobtanium.
Un militaire extrait 1kg d'Unobtanium.
Un militaire communique avec les Na'vis.
Un militaire extrait 1kg d'Unobtanium.
Un scientifique extrait 1kg d'Unobtanium.
```

Les classes fournies à la figure 1 ne sont ni modifiables, ni dérivables.

Les questions suivantes peuvent être répondues indépendamment.

I'll be back (mais pas aux rattrapages !)

1. James Cameron souhaite pouvoir envoyer un Terminator (*i.e.*, un cyborg) sur Pandora *via* le caisson. Pour cela, il a récupéré une classe `Terminator` (sans les sources) d'un autre jeu vidéo, afin de ne pas avoir à la récrire. La documentation de cette classe est la suivante :
   - `Terminator(String)` crée un terminator de la série passée en paramètre ;
   - `String getSérie()` renvoie la série du terminator (*e.g.*, T-800, T-1000, T-X, T-3000) ;
   - `String replique()` renvoie l'une des 17 répliques d'Arnold Schwarzenegger dans le film *Terminator* ;
   - `void imiterVoix(String, Voix)` effectue la synthèse vocale du texte donné avec une voix spécifique (*i.e.*, `Voix.HOMME`, `Voix.FEMME`, `Voix.ENFANT`) ;
   - `void exécuterMission(String objectif)` fait exécuter au terminator les actions nécessaires pour atteindre l'objectif donné (*e.g.*, « Protéger John Connor »), grâce à son intelligence artificielle ultra développée basée sur le modèle de *Skynet*.

   Malheureusement, l'interface de la classe `Terminator` n'est pas directement compatible avec celle d'`Humain`.
   (a) Quel *design pattern* pouvez-vous utiliser pour résoudre le problème ? (0.5pt)
   (b) Modélisez le problème sous la forme d'un diagramme de classes UML. (2pts)
   (c) Écrivez en Java une implémentation des classes non fournies du diagramme. (2pts)
   (d) Créez un terminator T-800 et connectez-le à un avatar *via* le caisson. (0.5pt)

Je te vois.

2. James Cameron souhaite pouvoir comptabiliser le nombre total (en kg) d'Unobtanium que les humains ont réussi à extraire sur Pandora dans le jeu.
   (a) Quel *design pattern* pouvez-vous utiliser pour résoudre le problème ? (0.5pt)
   (b) Modélisez le problème sous la forme d'un diagramme de classes UML. (1pt)
   (c) Écrivez en Java une implémentation des classes non fournies du diagramme. (2.5pts)
   (d) Modifiez et complétez le code donné en exemple au début afin d'afficher en plus dans la console le nombre total de kg d'Unobtanium extrait. (1pt)

Parfois, tout un examen se résume à des révisions folles.

3. Soit le code suivant dans la classe `Jeu.java` :

```
public static void batailleFinale(int nbNavis, int nbHumains) {
  Bataille b = new Bataille();
  for (int i = 0; i < nbNavis; i++) {
    Navi navi = new NaviForet();
    navi.prendreArme();
    navi.monterBanshee(new BansheeForet());
    b.ajouterNavi(navi);
  }
  ... // ajout des humains a la bataille finale
  b.lancerBataille();
}
```

   Dans les prochains films d'*Avatar*, James Cameron dévoilera de nouveaux clans de Na'vi et types de Banshee (*e.g.*, ceux des mers dans le deuxième film). Il souhaite alors pourvoir rejouer la bataille finale entre les humains et les Na'vis des forêts du premier film, avec ces nouveaux Na'vis et Banshees à venir, mais sans avoir à modifier ni dupliquer le code de la classe `Jeu` à chaque fois.
   (a) Quel *design pattern* pouvez-vous utiliser pour résoudre le problème ? (0.5pt)
   (b) Modélisez le problème sous la forme d'un diagramme de classes UML. (2pts)
   (c) Écrivez en Java une implémentation des classes non fournies du diagramme. (1pt)
   (d) Modifiez le code de la classe `Jeu` pour répondre au problème. (1.5pts)