Examen de *Design Patterns*
2020–2021

— Durée : 1h
— Type : papier
— Tous documents autorisés.
— 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 (10pts)

1. Quel *design pattern* fournit l'interface qu'un client attend en utilisant les services d'une classe dont l'interface est différente ? (1pt)
2. Quel *design pattern* permet de contrôler l'accès à un objet en fournissant un intermédiaire pour cet objet ? (1pt)
3. Quel *design pattern* fournit un moyen d'accéder de façon séquentielle aux éléments d'un agrégat sans connaître sa structure interne ? (1pt)
4. Quel *design pattern* est illustré à la figure 1 ? (1pt)

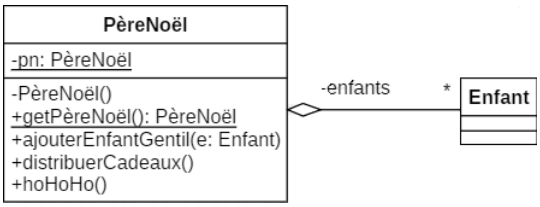

FIGURE 1 – Diagramme de classes

5. Quel *design pattern* est illustré à la figure 2 ? (1pt)
6. Quel *design pattern* est illustré à la figure 3 ? (1pt)
7. Parmi les affirmations suivantes, lesquelles sont vraies ? (2pts)
   (a) Le *design pattern* Décorateur ne peut pas être implémenté sans le polymorphisme.
   (b) Un *design pattern* offre une implémentation qui solutionne un problème récurrent.
   (c) Chaque classe pouvant être visitée doit mettre à disposition une méthode publique `accept(Visiteur)`.
   (d) Un observateur ne peut pas observer plus d'un sujet.
8. Quel est l'impact d'un changement de sous-système sur les clients d'une façade ? (1pt)
9. Qu'est-ce que le *design pattern* Commande permet de faire ? (1pt)

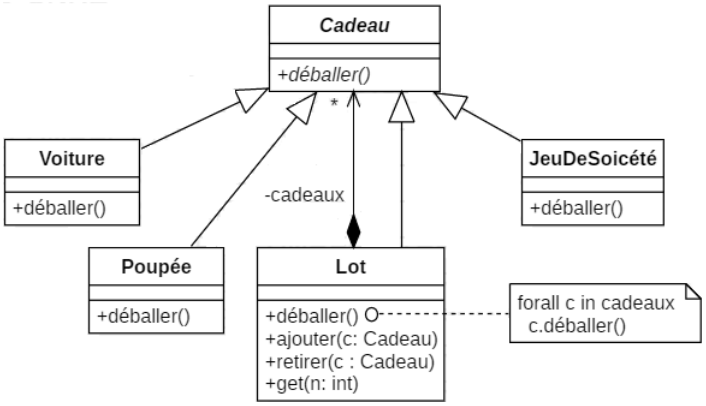

FIGURE 2 – Diagramme de classes

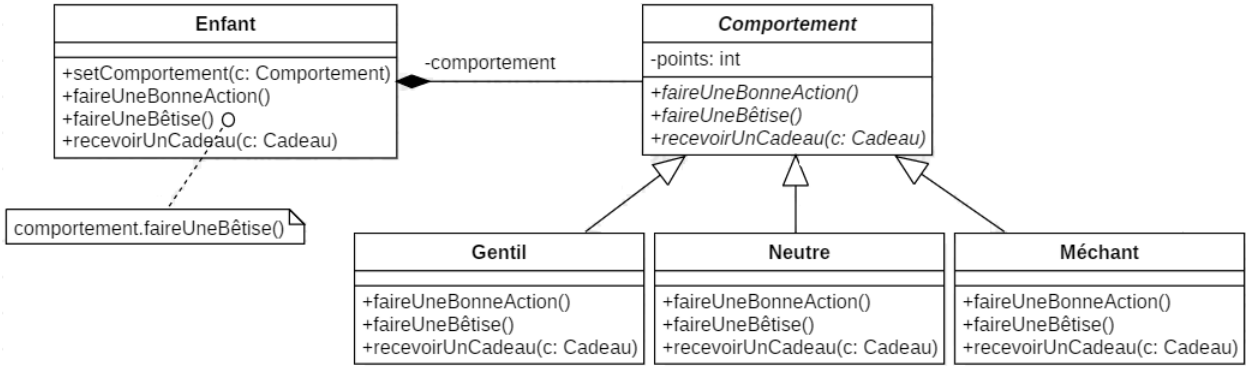

FIGURE 3 – Diagramme de classes

Super Mario Bros. (10pts)

Pour fêter le 35ème anniversaire de la sortie de *Super Mario Bros.* sur NES, Shigeru Miyamoto a décidé de reprendre le code du jeu et de développer 2 nouvelles fonctionnalités pour les fans. Un extrait du digramme de classes (fictif) du jeu est illustré à la figure 4.

1. La première fonctionnalité est de permettre au joueur de jouer aux différents niveaux de *Super Mario Bros.* avec les nouvelles classes de *Mario*, *Goomba*, *Koopa* et *Bowser* développées pour les versions plus récentes de la série. Ainsi, au démarrage du jeu, le joueur pourra choisir la version de *Super Mario Bros.* (*e.g.*, NES, SNES, 64...) avec laquelle il souhaite jouer (non demandé dans l'exercice). Pour cela, vous disposez des classes `MarioNES`, `MarioSNES`, `Mario64`. Idem pour les autres personnages du 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. (2pts)
   (c) Écrivez en Java une implémentation des classes du diagramme et modifiez, si nécessaire, le code existant. (2.5pts)
2. La seconde fonctionnalité est de permettre au joueur de jouer avec le personnage *Bowser* à la place de *Mario* ou *Luigi*. Malheureusement, l'interface de la classe `Bowser` n'est pas directement compatible avec celle de la classe `Joueur`.
   (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 du diagramme. (2.5pts)

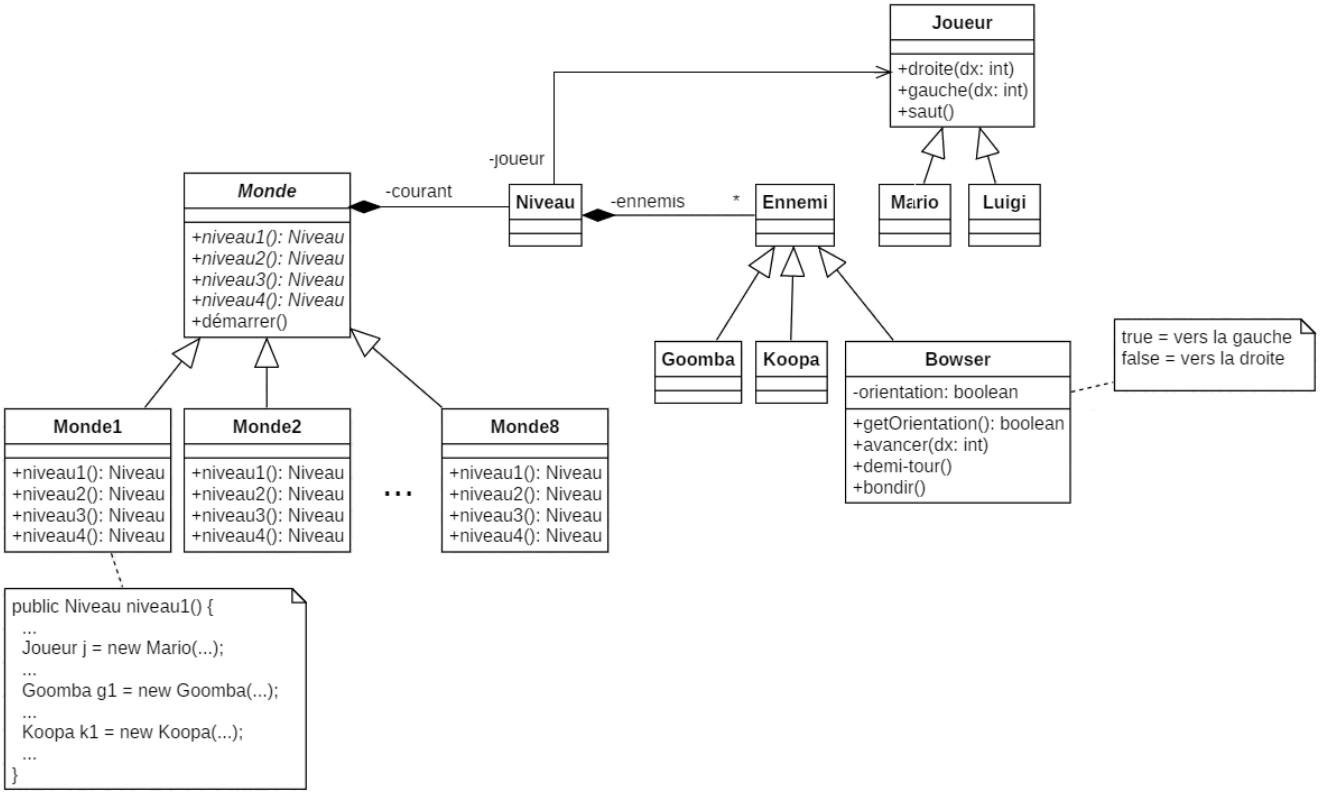

FIGURE 4 – Extrait du diagramme de classes du jeu *Super Mario Bros.*