Université Claude Bernard Lyon 1
Département Informatique

Licence 3, 2023-2024

durée : 1h30

Programmation Orientée Objet

Aucun document autorisé
Barème indicatif, non définitif
Les diagrammes présentés peuvent être partiels pour faciliter leur lecture

---

PARTIE 1 : Diagramme de séquence (4.25 points)

Un étudiant a commencé le diagramme de séquence en Figure 1 pour modéliser une interaction entre certains objets issus du diagramme de classe donné en Figure 2.

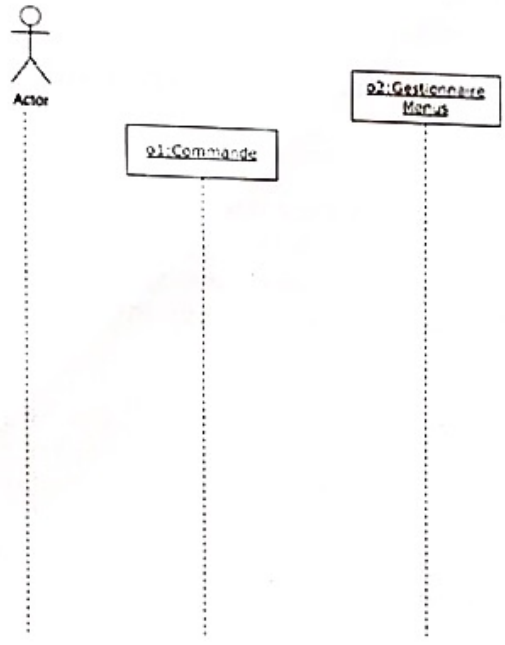


FIGURE 1 – Croquis d'un diagramme de séquence.

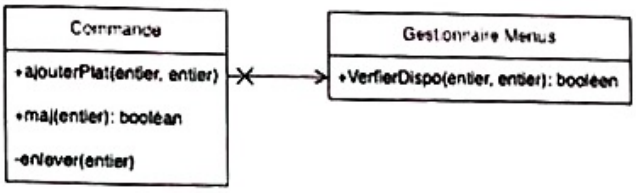


FIGURE 2 – Diagramme de classe

L'étudiant veut modéliser la situation suivante :

1. Un client (Actor) crée un objet `o1` de la classe Commande.
2. Tant que le client n'a pas spécifié la fin de l'opération, les opérations suiva sont effectués dans l'ordre suivant :
   (a) Le client fait appel à l'objet `o1` en lui transmettant `i` (id du plat à ajout la commande) et `n` (le nombre à ajouter à la commande). Cet appel ne r rien.
   (b) Dans son exécution, `o1`, fait appel à `o2` en lui transmettant `i` et `n`. Cet ap rendra un booléen `b1`.
   (c) Dans son exécution, `o2`, fait appel à `o1` en lui transmettant un nombre en positif p si le plat est disponible en quantité suffisante et négatif sinon. (appel rendra un booléen `b2`.
   (d) Dans son exécution, si p est négatif, `o1` fait appel à sa méthode privée `enlev` en lui transmettant i. Cette méthode ne rend rien.

Question 1 *Terminez le diagramme de l'étudiant afin de modéliser la situation décri Vous devez utilisez les méthodes présentes dans le diagramme de classe de la Figure 2*

PARTIE 2 : Diagramme états-transitions (4.25 point

Un cheval peut être à l'arrêt ou à l'une des trois allures qui sont le pas, le trop et galop. Le cavalier peut interagir avec le cheval en tirant sur les rennes ou en serrant l jambes. Serrer les jambes permet de faire passer le cheval de l'arrêt au pas, du pas a trop ou du trop au galop. Quelque soit l'allure du cheval, tirer sur les rennes permet d s'arrêter.

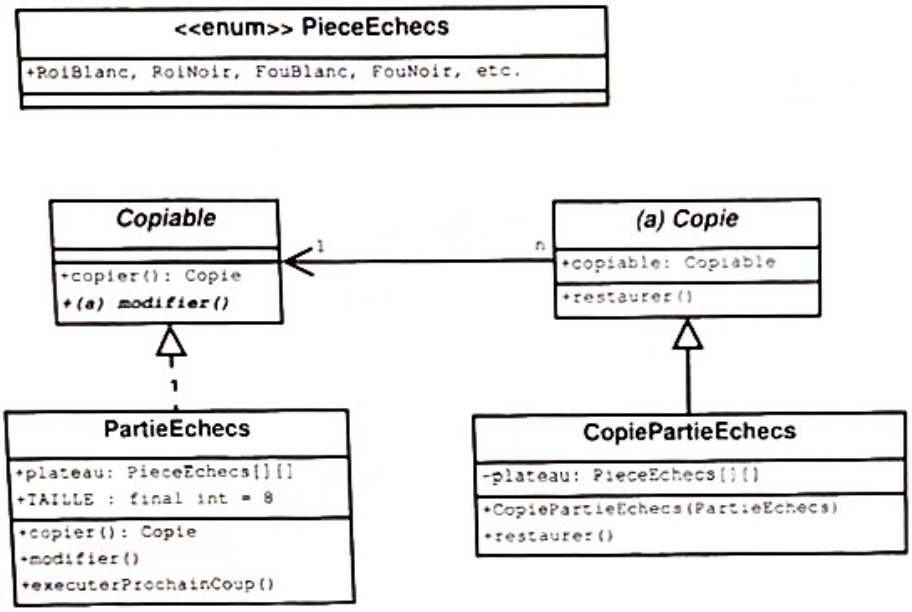
Quand le cheval est à l'une des trois allures, il peut être en place (c'est-à-dire avec l dos bien rond et la tête vers le bas) ou ouvert (c'est-à-dire n'importe comment). Pa défaut, il est ouvert et il se remet dans l'état ouvert au bout de 10 secondes. Le cavalie peut serrer les rennes pour mettre son cheval en place.

Quand le cheval est à l'arrêt, le cavalier peut caresser son cheval pour le récompenser Quand le cavalier a fini, il descend du cheval.

Question 2 *Dessinez un diagramme états-transitions correspondant aux états d'u cheval décrits ci-dessus.*

PARTIE 3 : Gestion de l'historique d'un objet – d et undo() (11.5 points)

Sur le diagramme de classes ci-dessous, l'état d'une partie d'échecs (c'est à di position des pièces, représenté par l'attribut `plateau`) peut être sauvegardé dan objet **CopiePartieEchecs**.



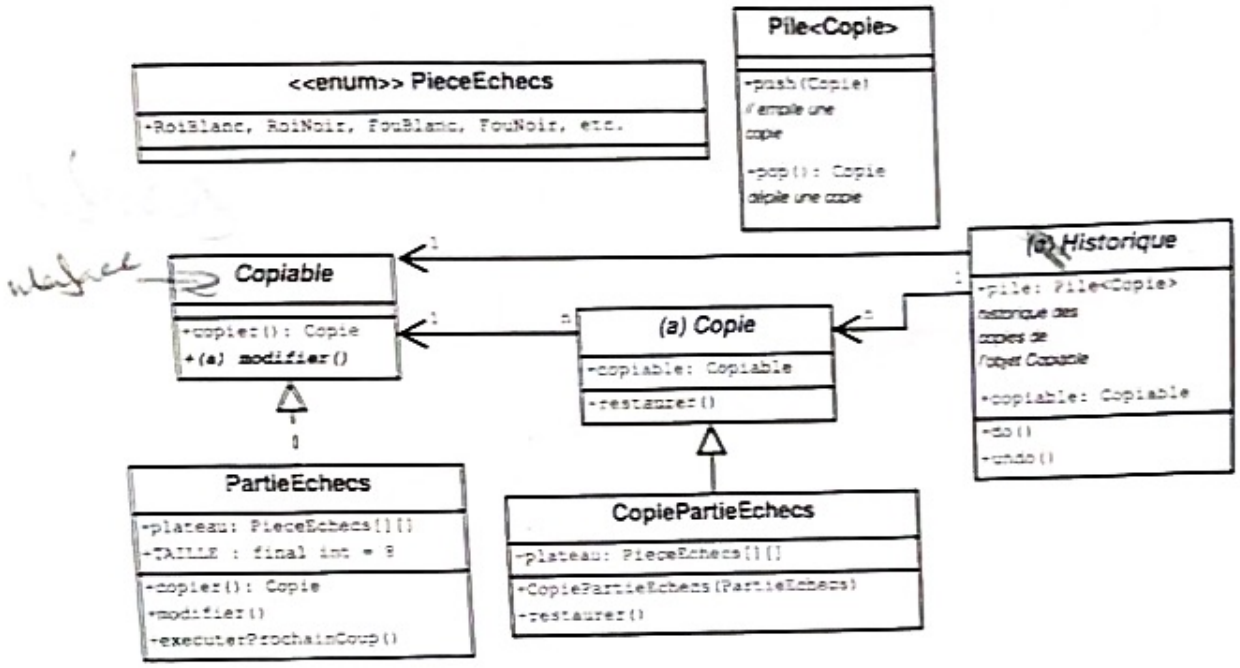
Question 3 *(2.5 points) Proposez l'implémentation de la fonction* `copier()` *classe* ***PartieEchecs*** *et du constructeur* `CopiePartieEchecs(PartieEchecs)` *classe* ***CopiePartieEchecs***.

Sur le diagramme complété ci-dessous, la fonctionnalité `do()` de la classe **Histor** permet de conserver une copie de l'objet **Copiable** avant de le modifier (La fon `modifer()` de **PartieEchecs** lance l'appel de `executerProchainCoup()`).

La fonctionnalité `undo()` permet de restaurer l'objet à son état précédent.

Question 4 *(2 points) Proposez l'implémentation des fonctions* `do()` *et* `undo` *la classe* ***Historique***.

Question 5 *(3 points) Soit une partie d'échecs pour laquelle chaque joueur un coup. Donner le diagramme d'objets associé à ce scénario suivant le diagram classes ci-dessous.*



Soit une historisation plus riche permettant de mémoriser plusieurs évolutions de l'objet **Copiable** grâce à une structure en arbre. La racine de l'arbre représente l'objet dans son état initial (début de partie au jeu d'échecs), chaque branche de l'arbre représente une évolution possible (différentes positions de jeu aux échecs).

Selon ce nouveau fonctionnement, l'objet suivi est historisé systématiquement après chaque modification, afin de comparer la copie (par exemple du jeu) à l'arbre des copies.

À chaque mouvement en profondeur (fonction `do()`), la copie est ajoutée comme sous-arbre seulement si l'état de l'objet suivi est nouveau. La fonction `undo()` permet de restaurer l'objet suivi à partir de la copie mère de la copie courante.

Question 6 *(4 points) Proposez une nouvelle implémention des fonctions* `do()` *et* `undo()` *de la classe Historique, en vous basant sur le diagramme ci-dessous.*