

DEV2 – JAVL – Laboratoire Java**TD 3 – L'orienté objet***Partie 2 : Retour sur le Memory***Consignes**

Durant ce TD nous allons développer le jeu du Memory en appliquant les concepts de l'orienté objet. L'objectif est de comparer cette nouvelle implémentation avec l'implémentation réalisée dans le TD01. Le temps de travail estimé se compose de 2h de travail en classe et de 2h de travail à domicile.

Mise en place

- ▷ Vous allez utiliser le dépôt créé lors du 1^{er} TD.
- ▷ Vous allez coder dans le **même projet**, dans un package dédié (`g12345.dev2.td03`).
- ▷ Assurez-vous que la version du dépôt sur le PC est synchronisée avec le dépôt sur GITLAB.

1 Modélisation

Après avoir pris connaissance des règles du jeu¹, une première étape consiste à le modéliser. Pour ce faire nous allons découper le jeu en plusieurs classes en essayant de regrouper les tâches que le programme doit effectuer.

Information et action sur une carte. Un premier groupe concerne tout ce qui se rapporte à une carte. Cette classe que nous appellerons **Card** doit :

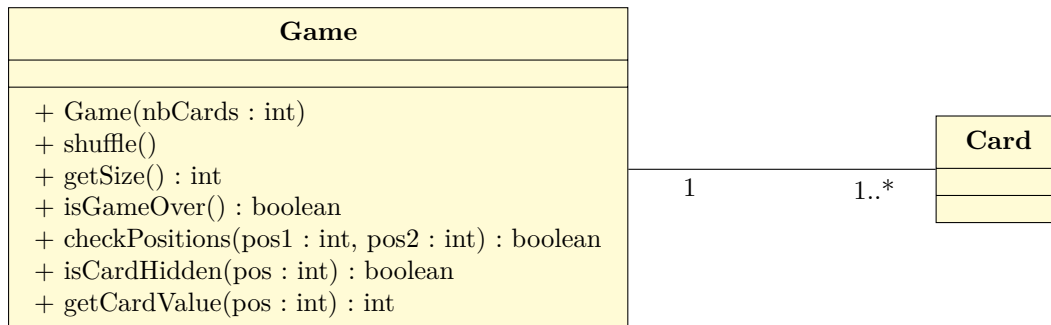
- ▷ connaître la valeur de la carte : 1,2,3,...;
- ▷ connaître l'état de la carte : caché ou révélé;
- ▷ changer l'état de la carte de caché à révélé;
- ▷ savoir si une carte est équivalente à une autre carte.

Card
- value : int - hidden : boolean
+ Card(value : int) + getValue() : int + isHidden() : boolean + reveal()

1. N'hésitez pas à parcourir les règles détaillées dans le TD01 si nécessaire.

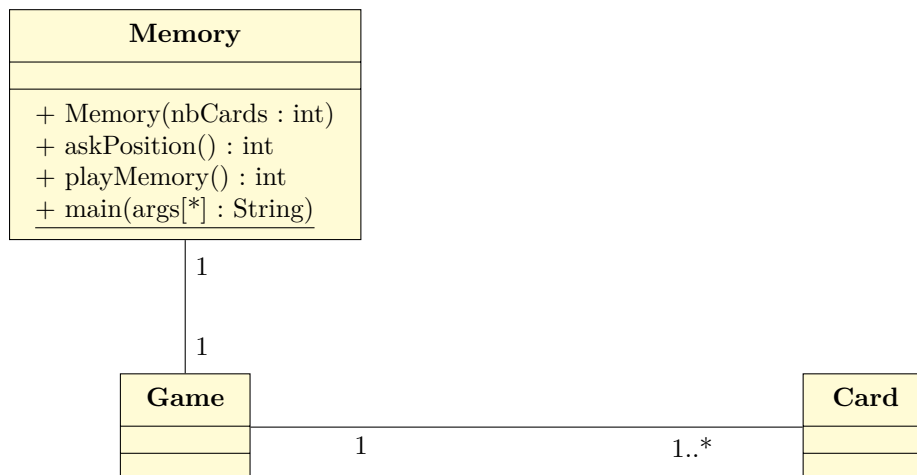
Information et action sur le jeu. Un second groupe concerne tout ce qui se rapporte au jeu, à ses règles et à ses composants (les cartes). Cette classe que nous appellerons **Game** doit :

- ▷ rassembler toutes les cartes du jeu ;
- ▷ pouvoir mélanger les cartes du jeu ;
- ▷ connaître le nombre de cartes du jeu ;
- ▷ savoir lorsque le jeu est terminé ;
- ▷ demander à deux cartes si elles sont équivalentes ;
- ▷ demander à une carte si elle est cachée ou révélée ;
- ▷ demander à une carte qu'elle est sa valeur.



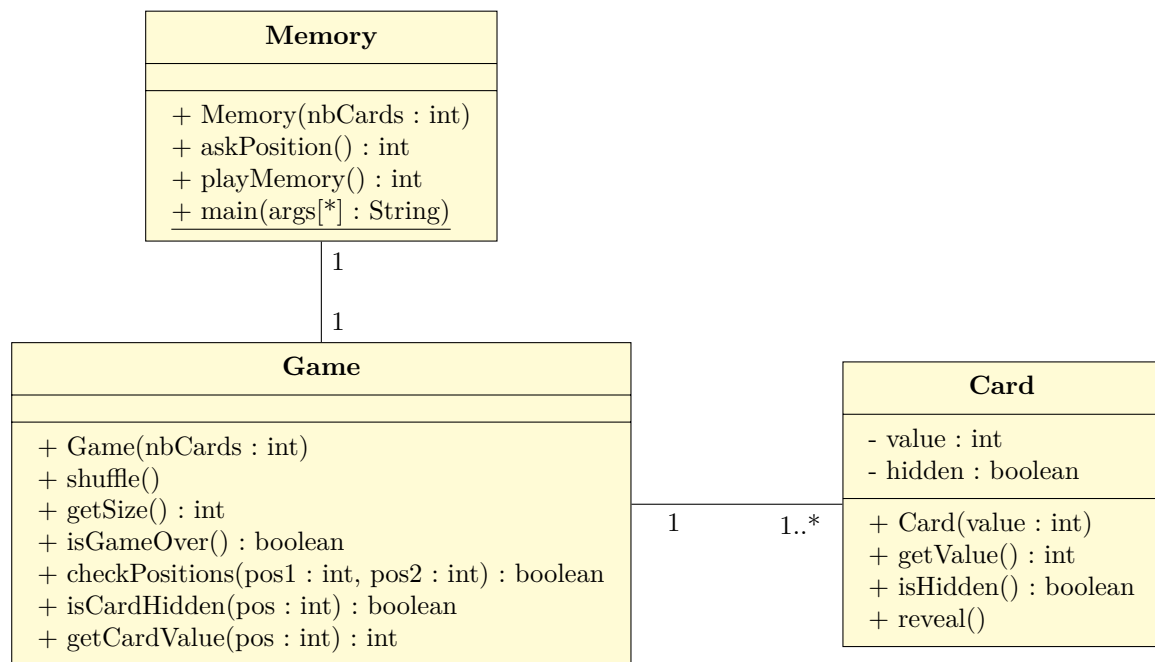
Contrôle et interaction. Un dernier groupe coordonne les actions du jeu avec les actions du joueurs. Cette classe que nous appellerons **Memory** doit :

- ▷ pouvoir démarrer une partie de **Memory** ;
- ▷ afficher des informations à l'utilisateur ;
- ▷ demander des actions à l'utilisateur ;
- ▷ interpréter les actions de l'utilisateur pour donner des ordres au jeu.



La division en responsabilité est évidemment discutable. Par exemple les responsabilités de contrôle du jeu et d'interaction avec l'utilisateur, peuvent être séparées afin d'apporter plus de clarté.

L'exercice de ce TD est d'implémenter le diagramme de classe complet suivant.



2 Implémentation

2.1 Le classe Card

Chaque carte du jeu est représentée par une instance de la classe **Card**.

Créez dans le package `<votreLogin>.dev2.td03` la classe **Card**.

Attributs

- ▷ `private boolean hidden` : prend la valeur vrai si la carte est cachée et faux dans le cas contraire ;
- ▷ `private int value` : représente la valeur de la carte, il s'agit d'un nombre entier strictement positif.

Constructeur Celui-ci prend en paramètre la valeur de la carte. Par défaut une carte est cachée à sa création. Si la valeur est négative, une exception est envoyée².

Méthodes

- ▷ Ajoutez les accesseurs des attributs ;
- ▷ `void reveal()` : cette méthode permet de révéler la carte, son attribut `hidden` doit être mis à faux ;
- ▷ `String toString()` : cette méthode retourne la chaîne de caractères "?" si la carte est cachée et retourne sa valeur si elle est révélée ;
- ▷ `boolean equals(Object obj)` et `int hashCode()` : ces méthodes permettent de comparer des objets entre-eux.

Écrivez les **tests unitaires** des différentes méthodes.

Vous devez écrire la *javadoc* de **toutes** les méthodes (hors méthodes de test).

2. Quelle exception envoyer dans ce cas parmi celles-ci : `IllegalArgumentException`, `NullPointerException` ou `IllegalStateException` ?

Faites un *commit* (message : "TD03 - Memory - Classe Card") avant de passer à l'étape suivante. Déposez votre travail sur le serveur GITLAB via la commande *push*.

2.2 Le classe Game

Le jeu est représenté par une instance de la classe **Game**.

Créez dans le package `<votreLogin>.dev2.td03` la classe **Game**.

Attributs

- ▷ `private Card[] cards` : tableau reprenant toutes les cartes du jeu en cours.

Constructeur Ce constructeur prend en paramètre le nombre de paires de cartes du jeu. Si le nombre de paires n'est pas entre 3 et 20 inclus, une exception est envoyée. Le constructeur crée un tableau de $2 * n$ objets **Card** et l'initialise avec les cartes des différentes valeurs. Chaque valeur de 1 à n apparaît deux fois.

Méthodes

- ▷ `void shuffle()` : cette méthode mélange les cartes. Pour ce faire, vous pouvez vous inspirer de la méthode fournie dans la classe **ArrayUtil** ;
- ▷ `int getSize()` : cette méthode retourne le nombre de cartes dans le jeu ;
- ▷ `boolean isCardHidden(int pos)` : cette méthode retourne vrai si la carte en position `pos` est cachée et faux dans le cas contraire ;
- ▷ `int getCardValue(int pos)` : cette méthode retourne la valeur de la carte en position `pos` ;
- ▷ `boolean checkPositions(int pos1, int pos2)` : cette méthode teste les deux cartes en position `pos1` et `pos2` et retourne vrai si elles correspondent et faux dans le cas contraire. Les cartes doivent être révélées en conséquence. La méthode doit lancer une exception si `pos1=pos2` ;
- ▷ `boolean isGameOver()` : cette méthode vérifie si le jeu est terminé, elle retourne vrai si toutes les cartes sont révélées et faux dans le cas contraire ;
- ▷ `String toString()` : cette méthode retourne une chaîne de caractères représentant toutes les cartes du jeu suivant leur statut ("?" ou valeur de la carte) ;
- ▷ `boolean equals(Object obj)` et `int hashCode()` : ces méthodes permettent de comparer des objets entre-eux.

Faites un *commit* (message : "TD03 - Memory - Classe Game") avant de passer à l'étape suivante. Déposez votre travail sur le serveur GITLAB via la commande *push*.

2.3 Le classe Memory

Le contrôle du jeu et les interactions avec l'utilisateur sont représentés par une instance de la classe **Memory**.

Créez dans le package `<votreLogin>.dev2.td03` la classe **Memory**.

Attributs

- ▷ `private Game game` : le jeu.

Constructeur Ce constructeur prend en paramètre le nombre de paires de cartes du jeu pour créer un jeu et initialiser l'attribut `game`.

Méthodes

- ▷ `int askPosition()` : cette méthode demande à l'utilisateur une position de carte, affiche la valeur de cette carte et retourne la position entrée par l'utilisateur. Si la position ne correspond pas à une carte encore en jeu, un message d'erreur est affiché et la méthode repose la question jusqu'à ce que ce soit correct ;
- ▷ `int playMemory()` : cette méthode appelle les différentes étapes du jeu et retourne le nombre de tours que le joueur a dû effectuer pour ramasser toutes les cartes. Les étapes du jeu consistent en :
 1. Mélanger les cartes.
 2. Répéter les étapes suivantes :
 - (a) Afficher les cartes.
 - (b) Demander une première position à l'utilisateur et afficher la valeur de la carte à cette position.
 - (c) Demander une seconde position à l'utilisateur et afficher la valeur de la carte à cette position.
 - (d) Jouer un coup avec ces 2 positions.
 - (e) Recommencer si la partie n'est pas finie.
- ▷ `static void main(String[] args)` : cette méthode demande à l'utilisateur avec combien de paires il veut jouer et crée le jeu correspondant. La partie est lancée et quand une partie est terminée, la méthode affiche le nombre de tours que le joueur a dû effectuer pour ramasser toutes les cartes.

Le développement du **Memory** est à présent terminé. N'oubliez pas de faire un dernier commit et de déposer votre travail sur le serveur GITLAB.

3 Impact de la visibilité

Afin de tester votre compréhension du concept de visibilité, essayez de répondre aux questions ci-dessous.

Dans la classe `Game`, la méthode `void shuffle()` peut avoir la visibilité :

1. `public`
2. `private`
3. `package`

Dans la classe `Card`, la méthode `void reveal()` peut avoir la visibilité :

1. `public`
2. `private`
3. `package`

Dans la classe `Card`, la visibilité du constructeur peut prendre les valeurs :

1. `public`
2. `private`
3. `package`

Dans la classe `Card`, la visibilité de la classe définie via `public class Card` peut-elle être changée en une visibilité `package` ?

Modifiez et compilez votre code afin de valider vos propositions.