Sujet complémentaire : Jeu de cartes - Bataille
Buts : Implémentation de plusieurs classes, Gestion des exceptions.

Avant de commencer

Dans le répertoire `POO` créez un sous-répertoire `TPCartes`. Vous y stockerez l'ensemble des solutions développées tout au long de ce TP. Votre enseignant devra y retrouver la trace de tout le travail effectué, ne remplacez donc pas vos solutions par d'autres, privilégiez, en cas de nécessité, une mise en commentaire. Pour une question de lisibilité, veillez à reporter les numéros des questions dans votre code. De la même façon les noms des auteurs devront y apparaître en haut de chaque script.

Création du dépôt Git

Votre travail sera impérativement déposé sur un dépôt Git.

Consignes

Vous veillerez à respecter scrupuleusement les consignes suivantes :

- Vous effectuerez un commit après chaque exercice.
- A la fin de chaque séance, vous effectuerez un commit. Si ce commit contient du code incomplet ou ne fonctionnant pas, mentionnez-le dans le message de commit. Vous pousserez ensuite votre travail vers le dépôt distant.
- Ce dépôt sera utilisé par votre enseignant(e) de TP pour évaluer votre travail. Assurez-vous donc régulièrement que tous les fichiers que vous souhaitez lui rendre sont bien présents dans le dépôt.
- Le dépôt lui-même sera évalué : soignez l'écriture de vos messages.

Initialisation du dépôt

- Ouvrez un terminal et utilisez la commande *cd* pour vous placer dans le répertoire $TPCartes$
- Initialisez le dépôt *Git* du projet.
- Si nécessaire, éditez le fichier *.gitignore*
- Effectuez un premier commit
- Connectez-vous à l'application *Gitlab* et créez un dépôt nommé $POO - TPCartes$
- Assignez le rôle *Reporter* à votre enseignant(e) de TP.
- Poussez le dépôt local vers le dépôt distant.

Liens utiles

- Aide-mémoire Git : *https://iut-info.univ-reims.fr/users/nourrit/git_aide_memoire.pdf*
- Gitlab : *https://iut-info.univ-reims.fr/gitlab*

Les bons usages

D'après les recommandations PSR-2, un script PHP débute par la balise <?*php* mais ne se termine par aucune balise. Vous éviterez de fermer vos scripts avec la balise ? >.

Afin que le typage de vos fonctions et de vos méthodes soit vérifié de manière stricte nous procéderons dorénavant à une typage strict. Vous ajouterez pour cela la directive suivante en tant que première instruction de vos scripts :

```
declare(strict_types = 1);
```

Vos classes doivent être clairement documéntées. Pour générer une documentation associée à votre classe vous le ferez à partir d'un terminal. Vous vous placerez pour ceci au niveau du répertoire contenant les fichiers à documenter. Voici le ligne de commande permettant d'obtenir la documentation de la classe `MaClasse` :

```
phpdoc -t Documentation --visibility public -f MaClasse.php
```

Introduction

L'objectif de ce sujet est de modéliser le jeu de la bataille entre 2 joueurs. La solution proposée sera générique : le code, en partie, pourra être réutilisé pour modéliser un autre jeu de cartes. L'implémentation proposée devra être correctement **documentée** et **commentée**.

1. Règles du jeu

La bataille est un jeu de cartes qui se joue à deux ou plus. On distribue l'ensemble d'un jeu de 52 cartes (figure 1) aux joueurs. Les joueurs ne regardent pas les cartes de leur main. À chaque tour, les joueurs retournent la carte du haut de leur main. Celui qui a la carte de plus haut ordre fait le pli qu'il place sous son tas. L'ordre des cartes est celui du jeu de bridge, soit dans l'ordre décroissant : as, roi, dame, valet, 10, 9, 8, 7, 6, 5, 4, 3, 2.

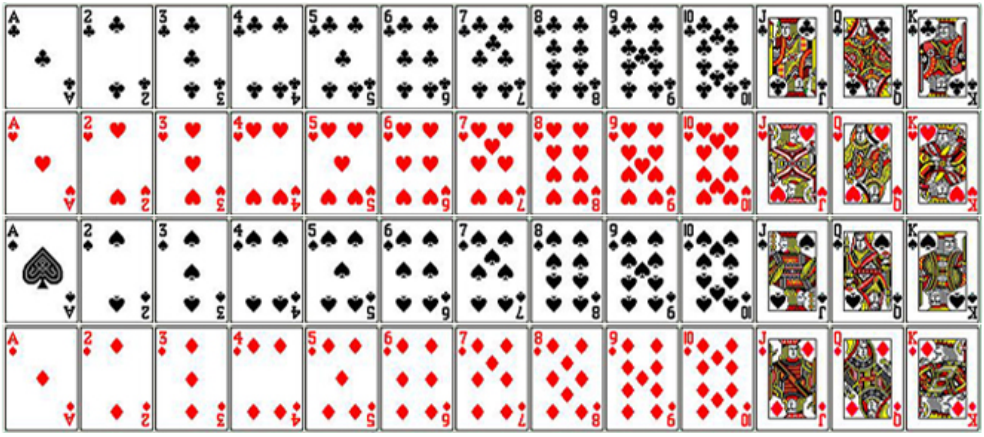


Figure 1: Jeu de cartes classique (52 cartes).

En cas d'égalité de deux cartes (même ordre) les joueurs se livrent à la bataille. Les joueurs posent alors une carte non retournée sur le pli puis retournent une nouvelle carte pour décider du vainqueur du pli (figure 2).

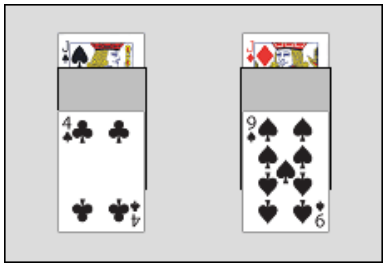


Figure 2: Une bataille

Le joueur qui termine le jeu avec l'ensemble des cartes en main gagne la partie.

2. Découpage en classes

La programmation est la phase terminale d'un long processus. L'étape fondamentale consiste à analyser le problème : le jeu de la bataille.

La bataille est un jeu de cartes qui se joue à deux ou à plusieurs. On distribue l'ensemble d'un jeu de 52 cartes aux joueurs. Les joueurs ne regardent pas les cartes de leur main. [···] *Celui qui a la carte de plus haut ordre fait le pli qu'il place sous son tas.*[···]

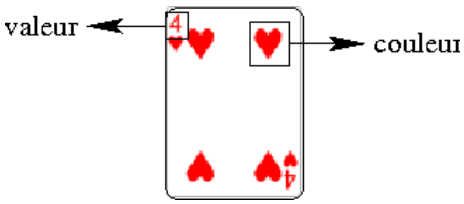
Outre les règles propres au jeu de la bataille, quelques éléments émergent de la lecture des règles :

- le jeu se joue avec des `cartes`,
- on manipule un *jeu de cartes, tas, pli* ou *main* qui sont des `tas` de cartes,
- le jeu se déroule entre des `joueurs`,
- le jeu de la `bataille` définit les comportements ou interactions entre les `cartes`, les `tas` et les `joueurs`.

Le découpage en classes qui en résulte est le suivant :

- une classe `Card` ;
- une classe `Pile` qui modélise les comportements liés à un tas de cartes ;
- une classe `Player` qui modélise les actions liées à un joueur. Un joueur possède un tas correspondant à sa *main* ;
- une classe `Battle` qui modélise le jeu entre deux joueurs.

3. La classe *Card*



Une Carte `Card` est caractérisée par sa couleur `color`, sa valeur `value` et son ordre `order`. Ci-dessous le diagramme de classe correspondant :

Card
- value : string - color : string - order : int

Les fonctionnalités nécessaires à la manipulation d'une Carte sont : la construction, les accesseurs, les comparaisons (effectuées sur le critère de l'ordre uniquement) et l'affichage en mode console.

Question 1. Dans le répertoire *TPCartes*, créez une classe nommée `Card`. Éditez la classe `Card` et complétez-la au fur et à mesure. En parallèle, toujours dans le répertoire *TPCartes*, créez un script nommé `TestsCarte.php` que vous compléterez et exécuterez à chaque ajout de nouvelle méthode dans la classe `Card`.

Ajoutez les attributs d'instance dans la classe `Card`

Question 2. On souhaite pouvoir instancier des objets de la classe `Card` de la manière suivante :

```
$vide = new Card;
$asDeCoeur = new Card("As" , "Coeur" , 13);
```

Le premier appel au constructeur initalisera la valeur et la couleur à `""`, et l'ordre à 0. Le second appel au constructeur les initalisera respectivement avec les valeurs de ses paramètres. Définissez le constructeur dans la classe `Card`.

Question 3. Définissez les accesseurs sur les 3 attributs d'instance de la classe `Card`.

Question 4. On souhaite pouvoir afficher une ou plusieurs intances de la classe `Card`. Pour ceci on aura besoin :

- de la méthode d'instance `__toString`
- de la **fonction** `printSideBySide` qui prend en paramètre les deux cartes et ne retourne rien. Elle se charge d'afficher deux cartes côte à côte.
- de la **fonction** `printSideBySideFaceDown` qui ne prend rien en paramètre et ne retourne rien. Elle se charge d'afficher deux cartes côte à côte face cachée.

La définition de ces trois modalités est disponible en ligne à l'adresse :
https://iut-info.univ-reims.fr/users/romaniuk/restricted/S2BasesPOO/ressources/Carte.txt

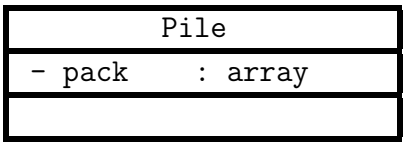
Récupérez ces définitions, documentez-les et testez-les.

Question 5. On souhaite pouvoir comparer deux cartes. Définissez les méthodes `isWeaker` , `isStronger` et `isEqual` dans la classe `Card`. Ces méthodes se limiteront à la comparaison (stricte) de l'`ordre` de l'instance `$this` avec celui de la Carte passée en paramètre de ces trois méthodes.

Question 6. Compte tenu de l'implémentation qui a été proposée pour la classe `Card`, est-il possible, de l'extérieur de la classe, de modifier un objet de type `Card` ? Quelle en est la conséquence sur les instances de cette classe.

4. La classe *Pile*

Un tas (`Pile`) de cartes est caractérisé par les `Cartes` qu'il contient (`pile`). Ci-dessous le diagramme de classe correspondant :



Dans cette classe vous serez amené à utiliser des fonctions prédéfinies en PHP pour manipuler les *array*. Vous êtes invités à utiliser la documentation officielle de PHP afin de les identifier (extraction, fusion, ...).

Question 7. Dans le répertoire *TPCartes*, créez la classe `Pile`. En parallèle, toujours dans le répertoire *TPCartes*, créez un script nommé `TestPile.php` que vous compléterez et exécuterez à chaque ajout de nouvelle méthode dans la classe `Pile`.

Ajoutez les attributs d'instance dans la classe `Pile`

Question 8. La définition du constructeur de la classe `Pile` est accessible à l'adresse :
https://iut-info.univ-reims.fr/users/romaniuk/restricted/S2BasesPOO/ressources/Tas.txt

Voici la documentation correspondante :

Constructeur permettant de charger un jeu de cartes à partir d'un fichier dont le nom est passé en paramètre. Si le nom n'est pas précisé à l'appel le tas initialisé sera vide. Dans le cas contraire, les cartes décrites dans le fichier sont stockées dans le tas[1].

Exemple d'appel : `$jeuCartes = new Pile ("bataille.ini");`

Récupérez la définition de ce constructeur dans la classe `Pile`. Récupérez les fichiers "bataille.ini" et "batailleSimplifie.ini" et enregistrez les dans le répertoire TPCartes. Testez le constructeur.

Question 9. Définissez une méthode d'instance `getCardsNumber` qui retourne le nombre de cartes dont est constitué un paquet

Question 10. On souhaite disposer de la méthode d'instance `getCard` dans la classe `Pile`. Cette méthode prend en paramètre la valeur d'un indice i et retourne la i^e carte du paquet. Définissez la méthode `getCard`.

Question 11. On souhaite disposer de la méthode d'instance `addCard` dans la classe `Pile`. Cette méthode permet d'ajouter une carte prise en paramètre à la fin du paquet. Définissez la méthode `addCard`.

Question 12. On souhaite disposer de la méthode d'instance `popCard` dans la classe `Pile`. Cette méthode permet de retirer la première carte du paquet (la carte que le joueur va poser

[1] les seuls fichiers disponibles pour le moment sont "bataille.ini" et "batailleSimplifie.ini" à l'adresse *https://iut-info.univ-reims.fr/users/romaniuk/restricted/S2BasesPOO/ressources*

sur la table). Cette méthode retourne la carte enlevée si le paquet contient des cartes, lance une exception de type `OutOfRangeException` sinon. Définissez la méthode `popCard`.

Question 13. On souhaite disposer de la méthode d'instance `showLast` dans la classe `Pile`. Cette méthode permet de consulter la dernière carte du paquet (la dernière carte posée par un joueur sur la table). Cette méthode retourne la carte consultée si le paquet n'est pas vide, lance une exception de type `OutOfRangeException` sinon. Définissez la méthode `ShowLast`.

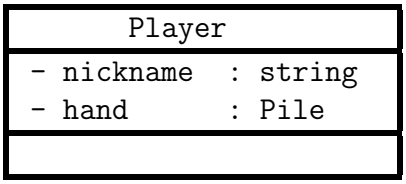
Question 14. On souhaite disposer de la méthode d'instance `drawCard` dans la classe `Pile`. Cette méthode permet de retirer aléatoirement une carte du tas (retirer une carte au hasard dans le jeu). Cette méthode retourne la carte enlevée si le tas contient des cartes, lance une exception de type `OutOfRangeException` sinon. Définissez la méthode `drawCard`.

Question 15. On souhaite disposer de la méthode d'instance `addPile` dans la classe `Pile`. Cette méthode permet d'ajouter à la fin du tas un autre tas passé en paramètre. Elle va être utilisée pour effectuer la prise à la fin d'un tour, aspect sur lequel on reviendra ultérieurement. Définissez la méthode `addPile`.

Question 16. Définissez la méthode d'instance `__toString` dans la classe `Pile`. Elle se limitera à l'affichage des cartes contenues dans le paquet.

5. La classe *Joueur*

Un jouer `Player` est caractérisé par son surnom `nickname` et sa main `hand`. Ci-dessous le diagramme de classe correspondant :



Question 17. Dans le répertoire *TPCartes*, créez une classe nommée `Player`. En parallèle, créez un script nommé `TestJoueur.php` que vous compléterez et exécuterez à chaque ajout de nouvelle méthode dans la classe `Player`.

Ajoutez les attributs d'instance dans la classe `Player`

Question 18. On souhaite disposer d'un constructeur qui prendra en paramètre le surnom du joueur dans la classe `Player`. Ce constructeur initialisera la main du joueur à un tas vide. Définissez ce constructeur.

Question 19. Définissez un accesseur sur le surnom dans la classe `Player`.

Question 20. Définissez la méthode `getHandCardsCount` dans la classe `Player`. Cette méthode retournera le nombre de cartes dont dispose le joueur dans sa main.

Question 21. On souhaite disposer d'une méthode d'instance `playCard` dans la classe `Player`. Cette méthode permettra au joueur de jouer la première carte de sa main (elle retournera la carte jouée). Cette méthode transférera une exception de type `OutOfRangeException`. Définissez la méthode `playCard`.

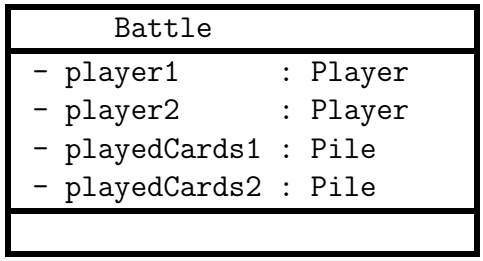
Question 22. On souhaite disposer d'une méthode d'instance `clearTable` dans la classe `Player`. Cette méthode prendra un tas (`Pile`) en paramètre. Elle se chargera d'ajouter ce pli à la fin de sa main et simulera ainsi la prise. Définissez la méthode `clearTable`.

Question 23. On souhaite disposer d'une méthode d'instance `newGame` dans la classe `Player`. Cette méthode prendra un tas en paramètre et s'en servira pour attribuer un nouveau jeu au joueur (celui-ci remplacera l'ancien). Définissez la méthode `newGame`.

Question 24. Définissez la méthode `__toString` dans la classe `Player`. Elle se limitera à assimiler le joueur à son surnom ainsi que le nombre de cartes qu'il a en main.

6. La classe Battle

La classe `Battle` sert à modéliser le jeu de la bataille entre **deux** joueurs. Elle est caractérisée par ses deux joueurs et les jeux (sur table) de chacun d'eux. Ces jeux sont des tas qui à la fin du jeu constituent le pli que récupère le joueur qui a remporté le tour. Ci-dessous le diagramme de classe correspondant :



Question 25. Dans le répertoire *TPCartes*, créez une classe nommée `Battle`. En parallèle, créez un script nommé `TestBataille.php` que vous compléterez et exécuterez à chaque ajout de nouvelle méthode dans la classe `Battle`.

Ajoutez les attributs d'instance dans la classe `Battle`

Question 26. Dans la classe `Battle` on souhaite disposer d'un constructeur avec trois arguments. Ces arguments, tous de type `string`, correspondent aux surnoms des deux joueurs ainsi qu'au nom du fichier contenant la description d'un jeu de carte. Ce constructeur assurera le chargement du jeu de cartes propre à la bataille (fichier "bataille.ini", ou "batailleSimplifie.ini") dans un tas, instanciera les joueurs et assurera la distribution du jeu de cartes à l'aide de la méthode d'instance de la classe `Pile drawCard`.

Voici un exemple d'affichage console obtenu lors de la bataille entre les joueurs.

```
Tour Simple
   Gaston              Tommy
 --------            --------
|        |          |        |
|     8  |          |     8  |
|        |          |        |
|        |          |        |
|   Pique|          | Carreau|
 --------            --------
Bataille
   Gaston              Tommy
 --------            --------
|        |          |        |
|        |          |        |
|        |          |        |
|        |          |        |
|        |          |        |
 --------            --------
 --------            --------
|        |          |        |
|     6  |          |   Roi  |
|        |          |        |
|        |          |        |
|  Trefle|          | Carreau|
 --------            --------
---------------> Prise : Tommy

  Joueur 1
            Gaston                        Cartes en main : 20

  Joueur 2
             Tommy                        Cartes en main : 32
```

Question 27. Dans la classe `Battle`, on souhaite simuler le déroulement d'un tour simple en implémentant la méthode d'instance `singleRound`. Lors de cette simulation chaque joueur joue à son tour une carte sur la table si cela est possible. Cette méthode d'instance retourne `true` si les cartes ont pu être jouées, `false` sinon. Elle affichera les 2 cartes posées en utilisant la fonction `printSideBySide`. Elle sera amenée à attraper des exceptions de type `OutOfRangeException`.

Question 28. Dans la classe `Battle`, on souhaite simuler le déroulement d'une bataille (qui suit un tour simple) en implémentant la méthode d'instance `battleRound`. Lors de cette simulation chaque joueur posera à son tour deux cartes sur la table si ceci est possible, la première côté dos, la seconde côté face. Cette méthode d'instance retourne `true` si les cartes ont pu être jouées, `false` sinon. Elle affichera les 2 premières cartes posées côté dos (fonction `printSideBySideFaceDown`) puis les 2 suivantes côté face (fonction `printSideBySide`). Elle sera amenée à attraper des exceptions de type `OutOfRangeException`.

Question 29. La méthode d'instance `decideBattle` permet de déterminer si le tour suivant sera une bataille. Elle compare pour cela les deux dernières cartes posées sur la table (les dernières jouées par chacun des joueurs). Elle retourne `true` si la bataille doit avoir lieu, `false` sinon. Définissez la méthode `decideBattle` dans la classe `Battle`.

Question 30. La méthode d'instance `clearTable` permet d'attribuer le pli au joueur ayant remporté un tour. Elle se charge de déterminer le joueur en question et de réinitialiser les jeux des deux joueurs une fois la prise effectuée. Définissez la méthode `clearTable`.

Question 31. La méthode d'instance `determineWinner` permet de désigner le joueur gagnant une fois la partie terminée. Elle retourne le surnom de ce joueur. Définissez la méthode `determineWinner` dans la classe `Battle`.

Question 32. Définissez la méthode d'instance `__toString` dans la classe `Battle`. Cette méthode permettra d'afficher l'état des joueurs (surnoms et nombre de cartes en main) à la fin de chaque prise.

7. Simulation du jeu

Question 33. Dans le répertoire *TPCartes*, créez un script nommée `JeuBataille.php`. Proposez-y une simulation du jeu de la bataille.