Sujet 5 (Complémentaire) : Yam's / polymorphisme et classes abstraites	
Buts :	Hétitage, polymorphisme, classe abstraite, tableaux 1D, produire une documentation

Avant de commencer

Dans le répertoire `POO` créez un sous-répertoire `TPYams`. 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. Si vous avez opté pour un dépôt commun à tous les TPs, n'oubliez pas de le compléter systématiquement en créant un répertoire par sujet de TP. Dans le cas contraire initialisez un nouveau dépôt en suivant les directives d'initialisation de dépôt décrites ci-dessous.

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 $TPYams$
- 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 $POO - TPYams$

- 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 documentées. 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
```

L'objectif de ce sujet est de consolider vos acquis sur les tableaux et l'héritage. Il permet, en plus, d'aborder de manière simple les notions de polymorphisme et de classes abstraites.

Dans ce sujet, même si ceci n'est pas demandé explicitement, une **documentation complète** est TOUJOURS exigée pour chaque classe et/ou méthode définie.

Avant de commencer

Modélisation d'un jeu de Yam's

Règles du jeu (Yam's Boîte 1985)

Un joueur lance 5 dés lorsque vient son tour. Son but est de réaliser une figure (5 dés identiques, un triplet...). Pour réaliser cette figure, il a le droit à trois lancers de dés par tour. Lors de chacun de ces jets, il choisit les dés qu'il souhaite relancer. Il n'est donc pas obligé de relancer tous ses dés lors d'un nouveau jet, il n'est pas non plus obligé de relancer les mêmes entre deux jets différents.

Le gagnant est celui qui récolte le plus de points. Pour ceci il doit réaliser un ensemble de figures, chaque figure ne pouvant être réalisée qu'une seule fois. Chaque figure lui permet de récolter un nombre de points précis. La feuille de score est divisée en deux parties : la partie mineure et la partie majeure.

La partie mineure

Les figures de la partie mineure visent à obtenir le plus grand nombre de dès pour chacune des valeurs (de 1 à 6). Le score associé à une telle figure correspond à la somme des dés affichant cette valeur spécifique. Ces figures ont pour nom :

- Nombre de 1 ;
- Nombre de 2 ;
- Nombre de 3 ;
- Nombre de 4 ;
- Nombre de 5 ;
- Nombre de 6.

Si un joueur obtient un total supérieur à 63 points sur la partie mineure, il récolte alors un bonus de 35 points.

La partie majeure

La partie majeure est composée de 7 figures spécifiques, dont ci-dessous les descriptions respectives :

- Brelan : le jet contient 3 dés d'une même valeur ; score : $3 \times$ (valeur des dés identiques) ;
- Carré : le jet contient 4 dés d'une même valeur ; score : $4 \times$ (valeur des dés identiques) ;
- Full : le jet contient 3 dés d'une même valeur et 2 dés d'une même valeur ; score : 25 ;
- Petite Suite : le jet contient 4 dés qui se suivent (1234, ...); score : 30 ;
- Grande Suite : le jet contient 5 dés qui se suivent (12345, ...); score : 40 ;
- Yams : le jet contient 5 dés d'une même valeur ; score : 50 ;
- Chance : le jet doit être composé de dés permettant d'obtenir le plus grand nombre de points ; score : somme de la valeur des dés ;

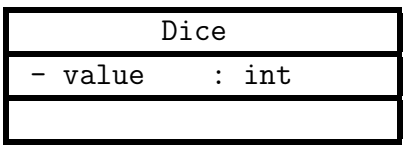
Présentation

L'objectif de ce sujet de TP est d'écrire une application simulant le jeu de Yams. Dans un premier temps cette simulation permettra de simuler le jeu d'un seul joueur. A cette fin, vous serez amenés à écrire et/ou utiliser un certain nombre de classes :

- `Dice` : cette classe représente un dé ;
- `DiceRoll` : cette classe représente un jet de dés ;
- `Combination` : cette classe représente une combinaison que le joueur doit réaliser avec les dés. Les classes qui en héritent permettent de spécifier concrètement les différentes figures possibles ;
- `Player` : cette classe représente le joueur ;
- `Game` : cette classe représente une partie en cours. Cette partie peut se dérouler pour un seul joueur ou entre plusieurs joueurs.

La classe Dice

La classe `Dice` correspond à la modélisation d'un dé et est représentée par la valeur de sa face située sur le dessus lorsqu'il s'arrête. Dans notre cas, on se limitera à la modélisation d'un dé à 6 faces, c'est-à-dire que les valeurs qu'il peut prendre sont comprises entre 1 et 6. Ci-dessous le diagramme de classe correspondant :



Dans le répertoire `TPYams`, créez une classe nommée `Dice`. Créez un script `TestDice.php` que vous compléterez et exécuterez à chaque ajout de nouvelle méthode dans la classe `Dice`.

Question 1.
Définissez la classe `Dice`, déclarez-y ses attributs d'instance en respectant leurs type et visibilité. Créez le script `TestDice.php`

Question 2.
Dans la classe `Dice`, définissez un constructeur qui prend en paramètre un entier. Cet entier servira à initialiser l'attribut d'instance de la classe. Si cet entier n'est pas compris entre 1 et 6, une exception de type `InvalidArgumentException` sera lancée.

Question 3.
Dans la classe `Dice`, on souhaite diposer d'un constructeur par défaut. Ce constructeur initialisera l'attribut d'instance avec une valeur aléatoire comprise entre 1 et 6. Modifiez le constructeur précédent en proposant une initalisation par défaut du paramètre à -1, valeur pour laquelle une tirage aléatoire devra être effectué.

Question 4.
Dans la classe `Dice`, définissez un accesseur sur son attribut d'instance.

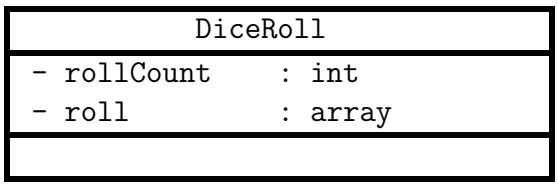
Question 5.
Dans la classe `Dice`, définissez une méthode d'instance `roll` qui ne retourne rien et ne prend rien en paramètre. Cette méthode a pour objetif de simuler l'action de jeter un dé. Elle met à jour l'attribut d'instance avec une valeur aléatoire comprise entre 1 et 6.

Question 6.
Dans la classe `Dice`, définissez la méthode magique `__toString`. Celle-ci retournera une chaîne composée uniquement de la valeur de l'attribut `value` (sans passage à la ligne).

Rappel : N'oubliez par que la **documentation complète** est **exigée** pour toute classe que vous développerez. Par ailleurs, toutes les méthodes d'une classe doivent être **testées** au fur et à mesure que vous les définissez.

La classe DiceRoll

La classe `DiceRoll` correspond à la modélisation d'un lancer de dés, c'est-à-dire les jets de dés du tour d'un joueur. Pour rappel, lors d'un tour, un joueur peut effectuer jusqu'à 3 lancers de dés pour obtenir une figure. Cette classe est caractérisée par le nombre de jets qu'a réalisé le joueur (initialisé à 1 par le constructeur), ainsi que le jet lui-même (tableau de dés). Ci-dessous le diagramme de classe correspondant :



Dans le répertoire `TPYams`, créez une classe nommée `DiceRoll`. Créez un script `TestDiceRoll.php` que vous compléterez et exécuterez à chaque ajout de nouvelle méthode dans la classe `DiceRoll`.

Question 7.
Définissez la classe `DiceRoll`, déclarez-y ses attributs d'instance en respectant leurs type et visibilité.

Question 8.
Dans la classe `DiceRoll`, définissez un constructeur qui prend en paramètre un entier représentant le nombre de dés à lancer. Il se chargera de simuler un jet (`roll`) complet. Si le paramètre est inférieur à 1, une exception de type `InvalidArgumentException` sera lancée.

Question 9.
Dans la classe `DiceRoll`, définissez un accesseur sur l'attribut `rollCount`.

Question 10.
Dans la classe `DiceRoll`, définissez la méthode d'instance `getCount` retournant le nombre de dés qui composent le jet (`roll`).

Question 11.
Dans la classe `DiceRoll`, définissez la méthode d'instance `getDice` qui prend en paramètre un entier nommé `index`. Elle retournera le dé d'indice `index` dans le tableau `roll`. Si le paramètre n'est pas un indice valide, une exception de type `OutOfRangeException` sera lancée.

Question 12.
Dans la classe `DiceRoll`, définissez la méthode d'instance `reRoll` qui prend en paramètre un tableau de booléens nommé `selection` et ne retourne rien et dont l'objectif est de simuler un nouveau lancer.

La méthode vérifiera en premier lieu que le tableau `selection` comporte le même nombre d'éléments que le tableau `jet`. Si ce n'est pas le cas, une exception de type `InvalidArgumentException` sera lancée.

Ensuite, la méthode relancera tous les dés du tableau `roll` pour lesquels le booléen correspondant du tableau `selection` est à vrai. Les autres dés ne seront pas relancés.

Enfin, si au moins un dé a été relancé, l'attribut `rollCount` sera incrémenté.

Question 13.
Dans la classe `DiceRoll`, définissez la méthode d'instance `computeFrequencies` qui ne prend rien en paramètre et retourne un tableau d'entiers. Ce tableau comportera 6 éléments dont le premier indice sera à 1. L'élément d'indice i du tableau aura pour valeur la nombre d'occurrences de la valeur i parmi les dés du lancer.

Question 14.
Dans la classe `DiceRoll`, définissez la méthode `__toString`. Celle-ci retournera une chaîne composée de deux lignes :

- la 1^e ligne comportera les valeurs des dés séparées par un espace ;
- la 2^e ligne comportera une lettre majuscule située en dessous de chaque valeur de dé. La première lettre sera A, la seconde B, etc...

Exemple :

```
4 5 2 6 1
A B C D E
```

La classe Combination

La classe `Combination` correspond à la modélisation d'une Figure. Elle est la classe mère qui sert de base à toutes les classes représentant des combinaisons spécifiques que le joueur doit réaliser avec des jets de dés. Cette classe est caractérisée par le nom de la figure, qui sera utilisé lors de l'affichage de la grille de scores du joueur, ainsi que le score obtenu par le joueur pour cette combinaison. Sa valeur initiale sera -1 ce qui signifie que le joueur n'a pas encore joué cette combinaison. Ci-dessous le diagramme de classe correspondant :

Combination
- name : string - score : int

Dans le répertoire `TPYams`, créez une classe nommée `Combination`. Créez un script `TestCombination.php` que vous compléterez et exécuterez à chaque ajout de nouvelle méthode dans la classe `Combination` et de ses classes dérivées.

Question 15.
Définissez la classe `Combination`, déclarez-y ses attributs d'instance en respectant leurs type et visibilité.

Question 16.
Dans la classe `Combination`, définissez un constructeur qui prend en paramètre une chaîne de caractères contenant le nom de la combinaison.

Question 17.
Dans la classe `Combination`, définissez les accesseurs sur ses deux attributs d'instance.

Question 18.
Dans la classe `Combination`, on souhaite disposer d'une méthode d'instance `computeScore` qui prend en paramètre un lancer de dés (`DiceRoll`) nommé `jetDeDes` et qui retourne le score associé. Ne disposant pas des informations nécessaires pour définir cette méthode dans la classe `Combination`, elle se limitera ici à retourner -2. Cette méthode sera ultérieurement redéfinie dans toutes les classe dérivées de la classe `Combination`.

Question 19.
Dans la classe `Combination` définissez la méthode d'instance `updateScore` qui prend en paramètre un lancer de dés (`DiceRoll`) nommé `jetDeDes` et ne retourne rien.

Cette méthode met à jour l'attribut d'instance `score` en faisant appel à la méthode `computeScore`, qui elle prendra en paramètre le `jetDeDes`.

Question 20.
Dans la classe `Combination`, copiez la définition de la méthode `__toString` suivante :

```
public function __toString() : string {
    return  sprintf("%15s : ", $this->nom) .
                  ($this->score > -1 ? $this->score : "-");
}
```

La classe LuckCombination

La classe `LuckCombination` est une classe qui hérite de la classe `Combination`. La figure `LuckCombination` n'impose aucune contrainte sur la composition du lancer. Le score de cette combinaison correspond à la somme des dés qui la compose. Cette classe ne contient aucune nouvelle caractéristique outre celles de la classe `Combination`.

Dans le répertoire `TPYams`, créez une classe nommée `LuckCombination`. Créez un script `TestLuckCombination.php` que vous compléterez et exécuterez à chaque ajout de nouvelle méthode dans la classe `LuckCombination`.

Question 21.
Définissez la classe `LuckCombination`, déclarez-y, si ceci est nécessaire, ses attributs d'instance en respectant leurs type et visibilité.

Question 22.
En supposant qu'aucun constructeur n'ait été défini dans la classe `LuckCombination`, est-il possible d'instancier un objet de la classe `LuckCombination` ? Si oui, donnez en la syntaxe, testez votre proposition, puis expliquez pourquoi. Sinon, justifiez votre réponse.

Question 23.
Dans la classe `LuckCombination`, définissez un constructeur par défaut qui initialisera le nom de la combinaison à *Chance*.

Question 24.
Dans la classe `LuckCombination`, redéfinissez la méthode `computeScore`. Cette méthode retournera la somme des dés du jet de dés passé en paramètre.

Question 25.
Créez un script `TestPolymorphisme.php`. Complétez ce script comme suit :
- instanciez un objet *chance* de type `LuckCombination` ;
- instanciez un objet $figure$ de type `Combination` ;
- affichez $figure$ et *chance* ;
- instanciez ensuite un objet $jetDeDes$ de type `DiceRoll` ;
- calculez et affichez le score de $figure$ associé à $jetDeDes$ (appel à la méthode `computeScore`) ;
- calculez et affichez le score de *chance* associé à $jetDeDes$ (appel à la méthode `computeScore`).

Que constatez-vous ? Justifiez votre réponse.

- définissez un tableau *combinaisons*. Ce tableau de 2 cases contiendra les objets $figure$ et *chance* ;
- en utilisant une boucle `foreach`, pour chaque case du tableau *combinaisons*, calculez et affichez le score associé à $jetDeDes$.

Que constatez-vous ? Justifiez votre réponse.

Définissez en mots simples la notion de polymorphisme.

La classe FullCombination

La classe `FullCombination` est une classe qui hérite de la classe `Combination`. Cette classe ne contient aucune nouvelle caractéristique outre celles de la classe `Combination`. Un $full$ est composé d'une paire (deux dés identiques) et d'un brelan (trois dés identiques), la valeur des dés de la paire devant être différente de la valeur des dés du brelan.
Exemples :

```
1 1 1 2 2
3 4 3 4 3
```

Question 26.
Créez la classe `FullCombination`, déclarez-y, si ceci est nécessaire, ses attributs d'instance en respectant leurs type et visibilité.

Question 27.
Dans la classe `FullCombination`, définissez le constructeur par défaut qui initialisera le nom de la combinaison à "$Full$".

Question 28.
Dans la classe `FullCombination`, redéfinissez la méthode `computeScore`. Cette méthode retournera 25 si le jet de dés passé en paramètre représente un full, ou 0 sinon.

La classe NumberCombination

La classe `NumberCombination` est une classe qui hérite de la classe `Combination`. Cette classe permet de représenter les 6 combinaisons de la partie mineure (Nombre de 1, Nombre de 2, ...), combinaisons correspondant à la somme des dés affichant une valeur spécifique.

Outre les caractéristique de la classe `Combination`, la classe `NumberCombination` est caractérisée par l'attribut d'instance privé `number` qui représente la valeur que les dés doivent afficher pour être comptabilisés.

Question 29.
Créez la classe `NumberCombination`, déclarez-y, si ceci est nécessaire, ses attributs d'instance en respectant leurs type et visibilité.

Question 30.

Dans la classe `NumberCombination`, définissez le constructeur qui prend en paramètre un entier. Ce constructeur initialisera l'attribut `number` avec la valeur du paramètre. Si ce dernier n'est pas compris entre 1 et 6, une exception de type `InvalidArgumentException` sera lancée. Le nom de la combinaison sera *Nombre de* suivi de la valeur de l'attribut `number`.

Question 31.
Dans la classe `NumberCombination`, redéfinissez la méthode `computeScore`. Cette méthode retournera $n \times nombre$, où n est le nombre de dés affichant la valeur de l'attribut `number`.

La classe StraightCombination

La classe `StraightCombination` est une classe qui hérite de la classe `Combination`. Cette classe permet de représenter une combinaison contenant une suite de dés dont les valeurs se suivent (grande suite et petite suite).

Outre les caractéristiques de la classe `Combination`, afin de pouvoir représenter simultanément la petite et la grande suite, la classe `StraightCombination` est caractérisée par les attributs d'instance privé `straightLenght` (longueur de la suite à détecter dans le jet de dés) et `scoreValue` (valeur que devra retourner la méthode `computeScore` si elle détecte une suite dans le jet de dés).

Question 32.
Créez la classe `StraightCombination`, déclarez-y, si ceci est nécessaire, ses attributs d'instance en respectant leurs type et visibilité.

Question 33.
Dans la classe `StraightCombination`, définissez le constructeur qui prend en paramètres une chaîne de caractères nommée *name* et deux entiers. Ce constructeur initialisera les attributs `straightLenght` et `scoreValue` avec les valeurs des paramètres entiers. Si l'un ou l'autre de ces derniers n'est pas strictement positif, une exception de type `InvalidArgumentException` sera lancée. Le nom de la combinaison sera la valeur du paramètre *name*.

Question 34.
Dans la classe `StraightCombination`, redéfinissez la méthode `computeScore`. Si le jet de dés comporte une suite de longueur supérieure ou égale à `straightLenght`, cette méthode retournera `scoreValue`. Sinon, la méthode retournera 0.

La classe SameCombination

La classe `SameCombination` est une classe qui hérite de la classe `Combination`. Cette classe représente les combinaisons de la partie majeure composées de plusieurs dés identiques (brelan, carré et, indirectement, *yams*).

Outre les caractéristiques de la classe `Combination`, la classe `SameCombination` est caractérisée par l'attribut d'instance privés `sameCount` qui représente le nombre de dés qui doivent être identiques pour que la combinaison soit validée.

Question 35.
Créez la classe `SameCombination`, déclarez-y, si ceci est nécessaire, ses attributs d'instance en respectant leurs type et visibilité.

Question 36.
Dans la classe `SameCombination`, définissez le constructeur qui prend en paramètres une chaîne de caractères nommée *name* et un entier. Ce constructeur initialisera l'attribut `sameCount` avec la valeur du paramètre entier. Si ce dernier n'est pas supérieur à 2, une exception de type `InvalidArgumentException` sera lancée. Le nom de la combinaison sera la valeur du paramètre *name*.

Question 37.
Dans la classe `SameCombination`, redéfinissez la méthode `computeScore`. Si le jet de dés comporte au moins `sameCount` dés identiques, cette méthode retournera `sameCount`$\times v$, où v est la valeur des dés identiques. Sinon, la méthode retournera 0.

La classe YamsCombination

La classe `YamsCombination` est une classe qui hérite de la classe `SameCombination`. Cette classe représente une combinaison comportant 5 dés identiques. Cette classe ne contient aucune nouvelle caractéristique outre celles issues de l'héritage.

Question 38.
Créez la classe `YamsCombination`, déclarez-y, si ceci est nécessaire, ses attributs d'instance en respectant leurs type et visibilité.

Question 39.
Dans la classe `YamsCombination`, définissez le constructeur par défaut. Le nom de la combinaison sera $Yams$.

Question 40.
Dans la classe `YamsCombination`, redéfinissez la méthode `computeScore`. Si le jet de dés comporte 5 dés identiques, cette méthode retournera 50. Sinon, la méthode retournera 0. (Pour implémenter cette méthode, vous n'êtes pas autorisés à copier le code qui se trouve dans la classe `SameCombination`.

Question 41.
Dans le script `TestPolymorphisme.php` :

- définissez un tableau 1D *testCombinaisons*. Ce tableau devra contenir au moins une figure de chaque type (7 combinaisons) ;
- dans une boucle $foreach$:
  - générez un jet de dé,
  - affichez le jet de dé,
  - pour chaque case du tableau *testCombinaisons*, calculez et affichez le score associé à ce jet.

Classe abstraite

Dans la classe `Combination` la méthode `computeScore` a été définie alors que nous ne disposions pas des informations nécessaires à sa définition. Cependant, afin de pouvoir bénéficier des avantages que présente le polymorphisme, cette méthode doit néanmoins être au minimum déclarée dans la classe `Combination`. On se propose pour ceci de la rendre abstraite.

Question 42.
Dans la classe `Combination`, rendez la méthode `computeScore` abstraite. A-t-on le droit de définir cette méthode dans la classe `Combination` ?

Exécutez le script `TestPolymorphisme.php`. Que constatez-vous ? Quel effet sur la classe `Combination` a cette action ? Peut-on instancier une objet de type `Combination` ? Expliquez pourquoi.

Mettez en commentaire les instanciations des objets de type `Combination` si ceci est nécessaire. Exécutez à nouveau le script `TestPolymorphisme.php`. Est-ce que le fait de rendre la méthode `computeScore` abstraite dans la classe `Combination` remet en cause le polymorphisme ? Justifiez votre réponse.

Question 43.
Dans la classe `LuckCombination` mettez en commentaire la méthode `computeScore`. Exécutez à nouveau le script `TestPolymorphisme.php`. Que constatez-vous ? Lorsque la classe mère contient une méthode abstraite, quel impact ceci a sur les classes dérivées ? Justifiez votre réponse.

Annulez la mise en commentaire de la méthode `computeScore` dans la classe `LuckCombination`.

La classe Player

La classe `Player` est caractérisée par son nom, son score, les 13 figures à réaliser, ainsi que le nombre de combinaisons qu'il lui reste encore à jouer. Ci-dessous le diagramme de classe correspondant :

Player
- name : string - score : int - combinations : array - emptyCombinations : int

Question 44.
Créez la classe `Player`, déclarez-y ses attributs d'instance en respectant leurs type et visibilité.

Question 45.
Dans la classe `Player`, définissez le constructeur qui prend en paramètre une chaîne de caractères. Ce constructeur initialisera l'attribut `name` avec cette chaîne. Il initialisera les attributs

`score` à 0 et *emptyCombinations* à 13. Le tableau `combinations` sera, quant à lui, initialisé avec les 13 combinaisons à réaliser, dans cet ordre :

- Somme des 1
- Somme des 2
- Somme des 3
- Somme des 4
- Somme des 5
- Somme des 6
- Brelan
- Carré
- Full
- Petite suite
- Grande suite
- Yams
- Chance

Question 46.
Dans la classe `Player`, définissez les accesseurs sur les attributs `name` et `score`.

Question 47.
Dans la classe `Player`, définissez la méthode d'instance `isFinished` qui ne prend rien en paramètre et retourne un booléen. Cette méthode retourne *true* lorsque la partie est terminée (valeur de l'attribut `$emptyCombinations$` inférieure ou égale à 0), $false$ sinon.

Question 48.
Dans la classe `Player`, définissez la méthode d'instance `getCombinationName` qui prend en paramètre un entier nommé *index* et retourne le nom de la combinaison d'indice *index* du tableau `combinations`. Si le paramètre n'est pas un indice valide, une exception de type `OutOfRangeException` sera lancée.

Question 49.
Dans la classe `Player`, définissez la méthode d'instance `playCombinaison` qui prend en paramètre un `DiceRoll` nommé *jetDeDes* et un entier nommé *index*. Cette méthode retourne un booléen. Si *index* est un indice valide pour le tableau de combinaisons, et si la combinaison d'indice *index* n'a pas encore été jouée (son score vaut -1), on effectuera les opérations suivantes :

- appel de la méthode *updateScore* de la combinaison d'indice *index* avec *jetDeDes* ;
- mise à jour du score du joueur ;

- décrémentation du nombre de combinaisons non jouées ;
- retour de la valeur *true*.

Sinon, si *index* n'est pas un indice valide pour le tableau de combinaisons, ou si la combinaison d'indice *index* a déjà été jouée, la méthode retournera $false$.

Question 50.
Dans la classe `Player`, définissez la méthode d'instance `determineBonus` qui ne prend rien en paramètre et retourne le bonus dont bénéficie le joueur. Si un joueur obtient un total supérieur à 63 points sur la partie mineure, la méthode retoune 35, elle retourne 0 sinon.

Question 51.
Dans la classe `Player`, définissez la méthode `__toString`. Celle-ci retournera une chaîne représentant l'état de la partie en respectant au maximum les contraintes suivantes :

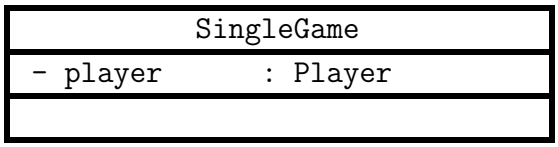
- la première ligne comporte le nom du joueur ;
- les lignes situées entre les `"----------"` correspondent aux combinaisons et comportent un numéro (1 à 13) suivi d'une chaîne représentant la combinaison (résultat de `__toString`) ;
- la dernière ligne comporte le score total du joueur.

Voici un exemple de rendu possible :

```
           JOUEUR : Bob
----------------------------------------
 1    Nombre de 1 : -
 2    Nombre de 2 : -
 3    Nombre de 3 : -
 4    Nombre de 4 : -
 5    Nombre de 5 : -
 6    Nombre de 6 : -
 7         Brelan : -
 8          Carré : -
 9           Full : -
10   Petite suite : -
11   Grande suite : -
12           Yams : -
13         Chance : -
----------------------------------------
            TOTAL : 0
```

La classe SingleGame

La classe `SingleGame` permettra de simuler le jeu du $Yam's$ pour un joueur. Elle est caractérisée par le joueur en question. Ci-dessous le diagramme de classe correspondant :



Question 52.
Créez la classe `SingleGame`, déclarez-y ses attributs d'instance en respectant leurs type et visibilité.

Question 53.
Dans la classe `SingleGame`, définissez le constructeur qui prend en paramètre une chaîne de caractères nommée *name*. Il attribuera au `player` le surnom *name*.

Question 54.
Dans la classe `SingleGame`, on souhaite disposer des méthodes d'instance privées `enterChoice` et `getNumber`.

La méthode `enterChoice` prend en paramètre les consignes de saisie, elle les affiche puis permet de saisir au clavier une chaîne de caractères dont les caractères vont être transformés en majuscules si ceci est possible, et dans laquelle les espaces vont être enlevés.

La méthode `getNumber` prend en paramètre une chaîne de caractères. Si cette chaîne est exclusivement composée de caractères numériques, la méthode retournera le nombre entier contenu dans la chaîne. En revanche, si ce n'est pas le cas, elle lancera une exception de type `InvalidArgumentException`.

Complétez la définition de la classe `SingleGame` avec la définition de ces deux méthodes.

Question 55.
Dans la classe `SingleGame`, définissez la méthode d'instance privée `identifyDiceToRaise` qui prend en paramètres une chaîne de caractères nommée *choice* ainsi qu'un entier représentant le nombre de dés composant un lancer nommé $rollSize$. Cette méthode retourne une tableau de booléens dont la taille est $rollSize$. Les cases dont les indices correspondent aux dés à relancer seront mises à $true$, les autres à $false$.

Afin de déterminer si un dé est à relancer, on parcourra la chaîne de caractères *choice*. Cette chaîne est supposée contenir des lettres majuscules comprises entre la lettre $'A'$ et la lettre $'E'$ pour un jet de taille 5, $'F'$ pour un jet de taille 6, $'G'$ pour un jet de taille 7, $\cdots$. Ainsi, si la lettre $'A'$ est contenue dans *choice*, la case 0 du tableau sera mise à $true$; si la lettre $'E'$ est contenue dans *choice*, la case 4 du tableau sera mise à $true$ à condition que la case 4 existe dans le tableau.

Lorsque la chaîne de caractère *choice* n'est pas au bon format, la méthode `identifyDiceToRaise` lancera une exception de type `InvalidArgumentException` avec le message *Saisie incorrecte : Caractère(s) invalide(s).*

Question 56.

Dans la classe `SingleGame`, définissez la méthode d'instance privée `simulateGameTurn` qui ne prend rien en paramètre et ne retourne rien. Cette méthode permet de simuler le déroulement d'un tour. Voici une trace d'exécution permettant d'illustrer le déroulement d'un tour :

```
           JOUEUR : Bob
----------------------------------------
 1    Nombre de 1 : -
 2    Nombre de 2 : -
 3    Nombre de 3 : -
 4    Nombre de 4 : -
 5    Nombre de 5 : -
 6    Nombre de 6 : -
 7         Brelan : -
 8          Carré : -
 9           Full : -
10   Petite suite : -
11   Grande suite : -
12           Yams : -
13         Chance : -
----------------------------------------
            TOTAL : 0
6 1 6 4 2
A B C D E
Saisir un numéro de figure non remplie (1-13) ou les lettres des dés à lancer sans
                                                                    espace (ABCDE) : z
Saisie incorrecte : Caractère(s) invalide(s)
6 1 6 4 2
A B C D E
Saisir un numéro de figure non remplie (1-13) ou les lettres des dés à lancer sans
                                                                    espace (ABCDE) : 24
Numéro de figure incorrect
6 1 6 4 2
A B C D E
Saisir un numéro de figure non remplie (1-13) ou les lettres des dés à lancer sans
                                                                    espace (ABCDE) : bed
6 5 6 2 5
A B C D E
Saisir un numéro de figure non remplie (1-13) ou les lettres des dés à lancer sans
                                                                    espace (ABCDE) : D
6 5 6 6 5
A B C D E
Saisir un numéro de figure non remplie (1-13) : 9
```

L'algorithme d'un tour du jeu du $Yam's$ repose sur le principe suivant : lors d'un tour, un joueur a le droit à trois lancers de dés au maximum. Lors de chacun de ces jets, il choisit les dés qu'il souhaite relancer. Il n'est donc pas obligé de relancer tous ses dés lors d'un nouveau jet, il n'est pas non plus obligé de relancer les mêmes entre deux jets différents. Il peut choisir de s'arrêter avant d'atteindre les trois jets autorisés. L'algorithme correspondant peut se résumer en plusieurs étapes :

Lancer 5 dés
Tant que nombre de jets effectués ≤ 3 ou joueur n'a pas associé le jet à une figure :
  Afficher le jet
  Si nombre de jets < 3
    saisir choix du joueur avec consignes *Saisir un numéro de figure non remplie (1-13) ou les lettres des dés à lancer sans espace (ABCDE) :*
  Sinon
    saisir choix du joueur avec consignes *Saisir un numéro de figure non remplie (1-13) :*
  Essayer de transformer choix en nombre,
    Si succès
      tenter d'associer choix à figure du joueur
      si possible, tour fini
      sinon, afficher *Numéro de figure incorrect*
    Si échec
      Si nombre de jets < 3
        Essayer de déterminer dés à relancer
          Si succès
            Relancer jet
          Si échec
            Afficher *Saisie incorrecte : Caractère(s) invalide(s)* (méthode `identifyDiceToRaise`)
      Sinon
        Afficher *Vous n'avez plus la possibilité de relancer*

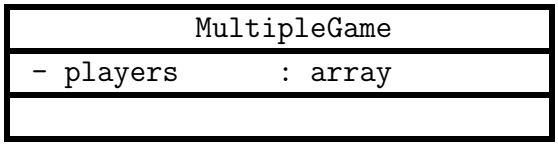
Dans cet algorithme, les étapes "Essayer de" correspondent à des structures de type $try \cdots catch$.
Définissez la méthode `simulateGameTurn`.

Question 57.
Dans la classe `SingleGame`, définissez la méthode d'instance `simulateGame` qui ne prend rien en paramètre et ne retourne rien. Cette méthode permet d'enchaîner les tours tant que la partie n'est pas terminée. Elle affiche ensuite le score de `player`. Attention, n'oubliez pas la gestion du bonus sur la partie mineure.

Question 58.
Validez le bon déroulement du jeu.

La classe MultipleGame (optionnelle)

La classe `MultipleGame` permettra de simuler le jeu du $Yam's$ entre plusieurs joueurs. Elle est caractérisée par les joueurs en question. Ci-dessous le diagramme de classe correspondant :



Question 59.
Créez la classe `MultipleGame`, déclarez-y ses attributs d'instance en respectant leurs type et visibilité.

Question 60.
Dans la classe `MultipleGame`, définissez le constructeur qui prend en paramètre un tableau de chaînes de caractères nommée *names*. Il attribuera aux `players` les surnoms contenus dans *names*.

Question 61.
En vous inspirant de la définition de la classe `SingleGame`, proposez une définition pertinente de la classe `MultipleGame`, celle-ci devant permettre de simuler une partie entre les `players`. Un classement devra être obtenu à la fin d'une partie.

Variantes du jeu de Yam's (optionnelle)

Il existe plusieurs variantes du jeu de Yam's :

- sec : le joueur n'a le droit qu'à un seul lancé de dés pour obtenir une figure ;
- désordre : le joueur a le droit à 3 lancers de dés par tour. Il sélectionne la figure à laquelle il veut associer le lancer ;
- ordre : le joueur a le droit à 3 lancers de dés au maximum par tour. Il doit remplir la feuille de score dans l'ordre ;
- ordre inversé : le joueur a le droit à 3 lancers de dés au maximum par tour. Il doit remplir la feuille de score dans l'ordre inverse.

Dans cette partie on s'intéressera à une feuille de score à quatre colonnes : sec, désordre, ordre et ordre inversé. A chaque tour, après son ou ses lancers, le joueur doit choisir la colonne qu'il souhaite remplir.

Proposez une solution à ce problème en implémentant une classe `MultiColumnMultipleGame`.