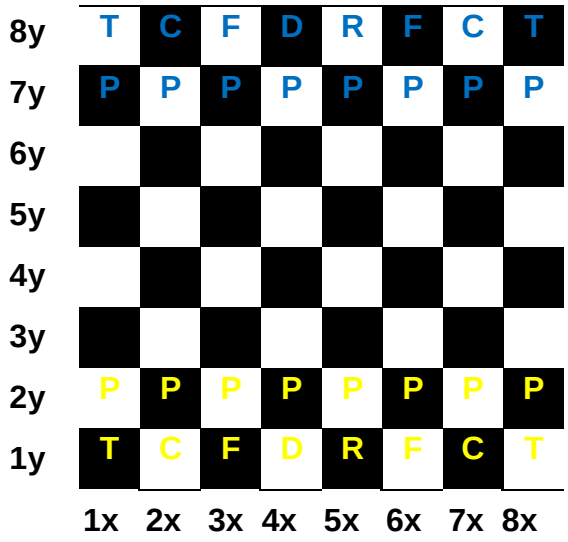
Exercices :

Jeu d'échecs



Les pièces blanches sont en bas.

Les pièces noires sont en haut.

La dame se situe sur sa couleur.

Les cases paires (x+y) sont noires.

Classe PieceEchecs

Les échecs fournissent un exemple d'application de l'héritage. Cet exercice aborde une partie du sujet.

Définissez une classe **PieceEchecs** avec :

2 attributs int privés définissant les coordonnées x, y (de 1 à 8) de la case sur laquelle la pièce se trouve (afficher une erreur si coordonnées incorrectes).

1 attribut *couleur* privé définissant la couleur de la pièce avec la convention

- utiliser une classe avec des constantes pour définir les couleurs :

```php
<?php
class Couleur {
    // As of PHP 7.1.0
    public const BLANC = 'blanc';
    public const NOIRE = 'noire';
}
echo Couleur::BLANC, // blanc;
echo Couleur::NOIRE, // noire;
?>
```

Vous sauvegarderez cette classe dans un dossier « util »

1 constructeur permettant l'initialisation des différents attributs.

1 méthode getCouleur () qui retourne une *couleur* (blanche ou noire) suivant que la pièce est blanche ou noire.

1 méthode getCouleurCase () qui retourne une couleur suivant que la case sur laquelle se trouve la pièce est blanche ou noire.

Testez dans le navigateur : Ecrire le programme principal permettant de tester les fonctionnalités décrites. Définir un jeu de test. Créez des setters dans la classe PieceEchecs pour permettre la modification des données membres. (Les setters doivent garantir les contraintes de valeurs pour les coordonnées et la couleur.)
Créez la méthode estDansLEchiquier(int,int) qui retourne vrai si la position passée en paramètre fait partie de l'échiquier.
Définir une méthode abstraite peutAllerA(int x,int y) dans la classe PieceEchecs renvoyant une valeur booléenne (la classe PieceEchecs devient donc une classe abstraite).

Classes Cavalier et Fou

Définissez 2 classes Cavalier et Fou héritant de PieceEchecs. Chacune de ces

classes implémentera :

- Un constructeur faisant simplement appel au constructeur de la superclasse par l'instruction : parrent ::__construct() ;
(https://www.php.net/manual/fr/language.oop5.decon.php).

- Définir la méthode peutAllerA(int x, int y) qui renvoie une valeur booléenne indiquant si la pièce en question peut aller à (x, y), compte tenu de sa position actuelle.

- Une méthode __toString() qui retourne une chaine de caractère de ce type :

Je suis un Cavalier, je me trouve sur la case (4,4) et je suis de couleur blanc.

Vous pourrez pour cela utiliser la méthode `get_class()`

(https://www.php.net/manual/fr/function.get-class.php).

Testez dans le navigateur.

Tableau de PieceEchecs

Dans votre programme principal, créez un tableau de 10 pièces d'échec. Chaque pièce sera soit un Fou soit un Cavalier que vous placerez sur l'échiquier.

Après l'initialisation du tableau, l'application déterminera pour chaque pièce si elle peut ou non aller à la case (5,5).

Remarque : Tel que les classes Cavalier et Fou ont été définies, chacune dispose d'une méthode peutAllerA.

Dans votre application, vous allez appeler la méthode peutAllerA à un élément du tableau, la bonne méthode associée à la classe sera alors exécutée.

Si on voulait simuler un véritable échiquier de 64 cases, quelle serait la bonne architecture logicielle à implémenter ? Il faudrait pouvoir écrire une méthode peutManger(x,y) ….

Facultatif :

Classe Roi

Créez une classe Roi dérivant aussi de PieceEchecs, avec les mêmes membres que Fou et Cavalier.

Testez dans le navigateur

Classe Pion

Créez une classe Pion dérivant aussi de PieceEchecs, avec les mêmes

membres. Testez dans le navigateur