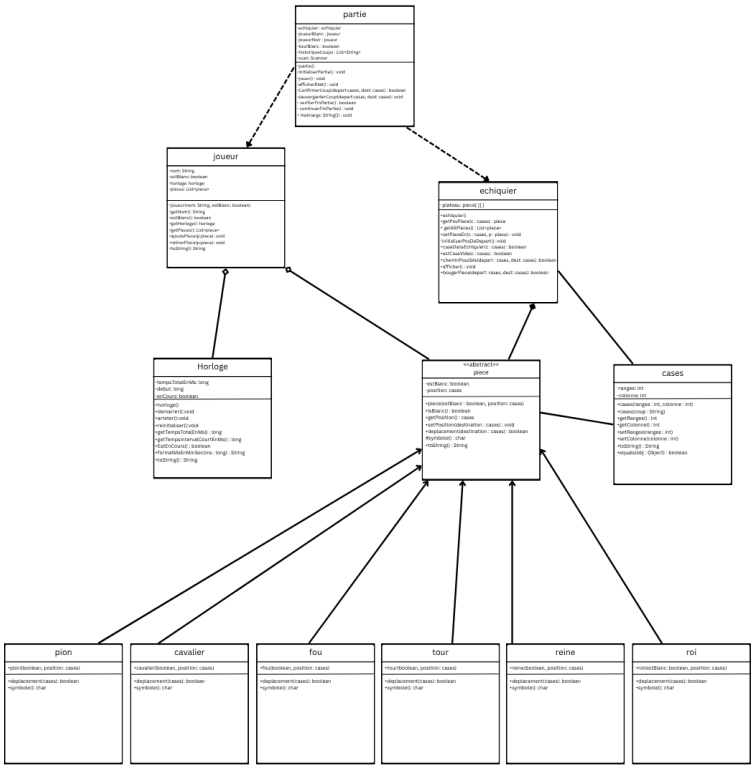
SAE JAVA - Jeu d’échec

PAR
BOUSARGHINI Mohamed
et
El MAHFOUDI Ali

Étape à effectuer pour exécuter le code:

1) Installer un JDK sur le site officiel d'Oracle (Si ce n'est pas déjà fait)
2) Dans le répertoire où les deux fichiers sont intallés, dézipper `BOUSARGHINI_ELMAHFOUDI.zip`:
   Sur Windows : clique droit sur `BOUSARGHINI_ELMAHFOUDI.zip` puis faire `extraire tout`
   Sur Linux: clique droit sur un endroit vide de l'explorateur de fichier dans le répertoire où le fichier a été installé `ouvrir dans le Terminal` puis taper `unzip BOUSARGHINI_ELMAHFOUDI.zip`
3) Aller dans le répertoire dézippé `BOUSARGHINI_ELMAHFOUDI`, clique droit sur un endroit vide de l'explorateur de fichier `ouvrir dans le Terminal` puis faire:
   dans un Terminal `javac *.java` puis `java partie`
4) Commencer la partie en entrant 2 noms.

DIAGRAMME UML(réalisé avant de faire le code):



l'image n'étant pas claire vous pouvez le visionner sur canva directement par ce lien: https://www.canva.com/design/DAGp3qEaiJo/WlMlwWebflsizoW-ZrL-8Q/edit
petite précision: les flèches des sous-classes de piece vers piece sont des flèches non pleine (impossible à mettre sur le diagramme)

Ce qui a été fait:

Le projet met en place une structure d'objet avec une classe abstraite, `pièce`, ayant une sous-classe pour chaque type de pièce d'échecs. Chaque pièce utilise une méthode `déplacement` qui correspond à ses règles de mouvement. Le jeu permettra à son tour le joueur blanc et noir de jouer en contrôlant leur mouvement, en s'assurant que le déplacement d'une pièce en particulier leur appartient. La capture d'une pièce adverse est correctement effectuée : si une pièce est sur la case de destination, elle est retirée du plateau et de la liste des pièces du joueur courant. L'échiquier s'affiche en tant que tableau lisible dans la console, avec des représentations claires du caractère de chaque pièce avec majuscule pour blanc, minuscule pour noir. Un historique des coups joués a été créé et affiché avec une notation simple telle que `c2 c4`, `e5xf6` ou `b2 b4` facilitant le suivi de la partie.

On ne détaille pas ce que font chaque méthode car déjà explicite grâce aux commentaires.

La classe `echiquier` a été implémentée pour représenter le plateau d'échecs. Elle est modélisée sous 8x8 cases avec un tableau à deux dimensions où chaque case peut être `null` ou contient une pièce. Au départ chaque pièce est positionnée selon le format de jeu d'échecs classique. Dans cette classe on a créé plusieurs méthodes pour gérer le plateau comme: obtenir une pièce à une certaine case, modifier une case, vérifier si une case est vide ou non, ou si une case donnée est bien située dans les limites du plateau. Une méthode a été créée afin d'afficher le plateau de manière compréhensible dans la console. On a créé d'autres méthodes afin de vérifier si un chemin est libre avec `cheminPossible`. On peut aussi récupérer toutes les pièces présentes sur le plateau avec `getAllPieces`. La méthode `bougerPiece` permet de faire un déplacement en mettant à jour la position de la pièce et en gérant les prises.

La classe `joueur` permet de représenter un joueur dans la partie. Cette classe gère plusieurs informations comme le nom, la couleur des pièces qu'il contrôle et son horloge personnelle. Cette classe permet aussi de modéliser les pièces contrôlées actuellement par le joueur. Cette méthode contient les getter, setter et toString de la propre classe.

La classe `partie` est la classe centrale de ce jeu d'échecs. On a créé une classe `initialiserPartie` qui permet de commencer la partie en demandant les noms des joueurs, créant les instances joueurs et positionnant les pièces des joueurs sur le plateau. On a la méthode `jouer` qui lance la boucle principale du programme, on permet aux joueurs de jouer chacun leur tour en demandant de saisir le coup voulu jusqu'à qu'il y ait une fin de partie. On y affiche aussi l'état du jeu avec leur pendule.

On a ensuite implémenter deux méthode `saisirCoup` et `verifierSaisie` qui permettent d'entrer le coup voulu et vérifier que cela est interprétable. On a aussi créé `confirmerCoup` qui valide si le coup respecte les règles du jeu et si elle est valide `sauvegarderCoup` applique le coup et la garde dans l'historique. Enfin, on a les méthodes `verifierFinPartie` et `continuerFinPartie` qui vérifier si, après chaque coup la partie doit se finir et affiche le résultat final.

La classe `cases` gère une case d'un échiquier qu'on peut interpréter par une rangée et une colonne comprises entre 1 et 8. Les constructeurs permettent de créer des cases avec les notations classiques d'échecs comme "`e4`" ou par rangée et colonne. On vérifie aussi si les cases sont valides et on a des exceptions dans le cas inverse. On a les `setter` et `getter` pour les rangées et colonnes utiles pour les autres méthodes ainsi qu'un `toString` qui représente la case par des positions compréhensibles tel que "a1" "h8". La méthode equals permet de vérifier deux cases donc comparer la colonne et la rangée.

Avec la classe `horloge` on mesure le temps cumulé passé pour chaque joueur en millisecondes avec un chronomètre. Le constructeur démarre le chrono à zéro. On a les méthodes `demarrer`, `arreter` et `reinitialiser` qui sont explicites pour gérer le chronomètre. On a la méthode `getTempsIntervalCourtEnMs` qui renvoie le temps passé depuis le dernier démarrage ou zéro si il est arrêté. `EstEnCours` indique l'état du chrono et on a `formatMsEnMinSec` convertisseur le temps en ms au format min:sec.

Enfin on a la classe abstraite `piece` représentant une pièce d'échec, avec un attribut pour la couleur et sa position `cases`. On inclut des méthodes pour accéder et modifier la position et vérifier la couleur. La méthode abstraite `déplacement` vérifie si le un déplacement vers une autre case est valide avec `symbole` qui retourne le caractère de la pièce. La `toString` affiche le symbole (lettre) en maj si la pièce est blanche et minuscule sinon. On a les sous-classes correspondant à chaque pièce dans d'autres fichiers à part qui héritent de `piece`. Chaque pièce possède sa propre manière de se déplacer et son symbole.

<u>Ce qui n'a pas été fait:</u>

On aurait pu faire plusieurs choses en plus mais on s'est limitées à l'essentiel.
Ce qui aurait pu être fait c'est de gérer les règles avancées du jeu d'échecs comme le roque la prise en passant etc… On aurait pu faire une interface graphique mais cela ne nous a pas été demandé. On aurait pu faire en sorte d'avoir un menu de démarrage. On pouvait aussi contrôler certaines erreurs comme la validation d'entrée dans les cas extrêmes.

On aurait pu aussi faire une classe test mais cela n'était pas demandé.

Ce qui fonctionne bien:

Le jeu global fonctionne bien. La capture d'une pièce adverse, le système de notation des coups le tour par tour, l'horloge l'affichage clair de l'échiquier fonctionne bien.

Ce qui ne fonctionne pas/partiellement:

La vérification complète d'un coup non autorisé comme traversée une autre pièce.
Le redémarrage de la partie. On doit faire ctrl+c pour finir la partie.
L'interface peut être difficile à comprendre pour un débutant.

Séparation du travail:
Mohamed: cases, horloge, pièces et toutes les sous-classes de pièces et échiquier
Ali: partie et joueur

En réalité, on a fait du mieux pour répartir équitablement le travail. On s'est entraidé sur chacune de nos classes car elles sont dépendantes des autres.
Donc on a autant travaillé sur ce projet.