UNIVERSITÉ
CAEN
NORMANDIE

Rapport

Les petits chevaux (JAVA)

19 mai 2019
Quentin Carre, Alexandre Roussel TD2.2
1ere DUT Informatique
M2103- Programmation Orientée Objet

Sommaire

I- Les fonctionnalités implémentées

Les fonctionnalités qui sont implémentées dans ce projet sont les fonctionnalités d'un jeu de petits chevaux classique. Parmi ces fonctionnalités, on retrouve entre autres :

- La possibilité de choisir le nombre de joueurs
- La possibilité de choisir sa couleur
- La possibilité de choisir son nom de joueur
- La détermination aléatoire du joueur qui commence
- Le tirage d'un dé de 6
- La possibilité de sortir un cheval de son écurie, de l'avancer sur le plateau et de manger les pions d'autres couleurs
- La vérification de la jouabilité des pions
- La vérification du joueur gagnant
- L'affichage d'un plateau basique qui permet de visualiser les écuries, le plateau et les échelles et les couleurs des pions (avec des lettres dénominatives).

II- Organisation du programme

Dans le cadre du projet des petits-chevaux en JAVA, un diagramme de classes nous a été fourni. Notre diagramme de classe reprend donc la plupart des éléments du diagramme de classe fourni.

Nous avons dans un premier temps analysé le diagramme fourni pour définir le rôle de chaque méthode à implémenter et pour regarder si certaines méthodes ou attributs doivent être modifiés pour faciliter leurs implémentations.

Nous avons donc fait plusieurs modifications qui nous ont amené à deux diagrammes de classes différents (annexe 1 et annexe 2) :

Nous avons défini une nouvelle classe, nous avons modifié un attribut et nous avons redéfini une méthode.

Nous avons donc ajouté une classe JoueurBot qui permet d'avoir des joueurs robots pour compléter les parties qui ont moins de 4 joueurs. Cette nouvelle classe redéfinit donc la méthode choisirPion. Malheureusement, nous n'avons pas réussi à implémenter une stratégie et cette classe de joueur ne joue que le premier pion détecté et disponible à bouger.

Nous avons également redéfini la méthode toString de la classe Pion pour qu'il puisse renvoyer l'ID du pion pour mieux les identifier lors des méthodes d'affichages comme le choix du pion.

Nous avons enfin modifié le type de l'attribut dé que nous avons redéfini en int (au lieu de random). Nous ne savions pas dans un premier temps que le type random existait. Nous avions donc utilisé un type int en début de projet avec auquel on appliquait la méthode math.random().

III- Organisation et répartition des tâches

Dans le cadre de ce projet en binôme, nous nous sommes repartis les tâches afin de créer et d'implémenter les fonctionnalités de ce jeu. Dans un premier temps, nous avons décidé de créer et d'implémenter les classes, les constructeurs et les méthodes d'accès. Nous avons décidé de répartir ces implémentations en fonctions des classes.

Nous nous sommes ensuite répartis les tâches en fonction des méthodes et des classes qui restaient à implémenter :

Quentin Carré :

Je me suis occupé de manière globale de l'affichage, de l'implémentation des classes cases et de l'implémentation de la méthodes jouerUnTour.

Je me suis donc occupé dans un premier temps d'implémenter les constructeurs et les méthodes d'accès de la classe Case et des classes qui héritent de celle-ci, et de la classe plateau. J'ai pu ensuite implémenter les premières méthodes qui ne dépendaient pas des autres méthodes tel que peutSArreter et peutPasser des classes de Case (et de celle qui hérite). Nous nous sommes par la suite confrontés à deux fonctions conséquentes qui sont : jouerUnTour et choisirPion. J'ai donc implémenté la fonction jouerUnTour. Après avoir effectué les tests pour vérifier le bon fonctionnement du programme, j'ai pu ajouter une nouvelle classe JoueurBot qui permet de compléter (en termes de joueur) les parties contenant moins de 4 joueurs. Cependant, l'implémentation de la stratégie de ces joueurs robots reste naïve.

Alexandre Roussel :

Je me suis tout d'abord occupé de l'écriture des classes Joueur, Pion et Partie avec l'initialisation de leurs méthodes et de leurs constructeurs. J'ai ensuite écrit la classe qui hérite de Joueur, JoueurHumain, avec notamment la fonction choisirPion qui était une des grosses méthodes du programme, cette méthode doit faire des conditions pour savoir quels pions le joueur peut bouger et ensuite demander au Joueur quel pion il souhaite jouer. J'ai également d'une partie de la classe Partie où j'ai écrit quelques méthodes et l'initialisation de la classe contenant le main PetitsChevaux.

Nous avons par la suite implémenté les classes exceptions que nous avons intégré dans chacune des méthodes où une erreur pourrait être levée. Nous avons également chacun de notre côté écrit la Java Doc au fur et à mesure des implémentations.

IV- Bilan qualitatif et problèmes rencontrés

Nous avons rencontré plusieurs problèmes lors de ce projet. L'une des premières difficultés a été la compréhension du digramme de classes fourni. Nous avons analysé dans un premier temps le digramme de classes pour déterminer le rôle précis de chaque méthode. Les 2 méthodes jouerUnTour et choisirPion ont été les méthodes qui nous ont le plus posé de problèmes. En effet, nous avons remarqué que dans le diagramme de classe fourni qu'il n'existait pas de méthode explicite qui permettait de vérifier qu'un mouvement de pion est valide. Nous en avons donc déduit que la méthode choisirPion permettait de choisir les pions déplaçables, et qu'elle devait par conséquent détecter les pions déplaçables en fonction du dé tiré.

La deuxième difficulté liée à ce projet que nous avons rencontré est la gestion des erreurs. Nous avons d'abord fait un programme fonctionnel qui vérifiait des conditions qui évitent le déclenchement des exceptions. Nous avons donc tenté de gérer des exceptions qui ne sont jamais levées. Nous avions par exemple l'exception PasDeJoueurException qui est une classe qui n'est pas sollicitée car lors de l'initialisation des joueurs, les méthodes forcent le joueur à choisir un nombre de joueurs supérieur à 0. Les exceptions ConflitDeCouleurException et CasePleineException nous ont également posé problème, car la méthode qui permet de vérifier si un pion est déplaçable empêche le joueur de jouer un pion qui pourrait lever ces exceptions.

La dernière difficulté que nous avons rencontrée a été la gestion des Scanners. Nous avons en effet utilisé des scanners qui prenaient des entiers. Mais ces mêmes scanners amènent à un plantage du programme si une entrée correspond à autre type qu'un entier. Nous avons tenté de corriger ce problème en les remplaçant par des scanners qui prennent des chaînes de caractères et qui les reconvertiraient en entier par la suite. Mais nous n'avons pas trouvé de solution qui permet d'effectuer cette action de manière optimale. Nous avons aussi remarqué qu'il n'était pas possible de fermer les scanners, car la fermeture d'un scanner ne permet pas d'ouvrir d'autres scanners.

Le projet de coder un jeu de petits chevaux nous a déjà été demandé auparavant avec le langage C. On a donc noté de grandes différences par rapport au langage C. Il a été, en effet, plus simple d'implémenter des méthodes en langage Java qu'en langage C, car la construction d'un projet en Java nécessite une conception préalable avec un diagramme de classe. Le diagramme de classe nous été fourni ce qui nous a permis de connaître les classes et les fonctions à implémenter. Contrairement au JAVA, le langage C nous demandait de faire des fonctions parfois plus difficiles à implémenter.

V- Mode d’emploi

Lors du lancement du jeu, le joueur a la possibilité de choisir entre quitter ou commencer pour lancer la partie.

```
-----------------------------
|                           |
|  Jeux des petits chevaux  |
|                           |
-----------------------------

1 - Commencer

2 - Quitter

Votre choix :
```

Je joueur pourra alors indiquer le nombre de joueurs, leurs noms et leurs couleurs. Les couleurs prises sont retirées à fur et à mesure.

```
Entrer le nombre de joueurs : 4
Entrer votre nom du joueur 1 : Joueur 1
1- Jaune
2- Rouge
3- Bleu
4- Vert
Choix de la couleur pour le joueur 1 : 1
```

Une fois le jeu lancer, le joueur pourra voir qui joue, le dé qu’il a tiré, et s’il peut déplacer un pion ou non. L’affichage d’un pion déplaçable est marqué par l’affichage d’une liste avec la possibilité de faire un choix. Ensuite, le programme affichera le plateau avec toutes les écuries, le plateau et les échelles et leurs couleurs. L’affichage du plateau se met à jour à chaque tour.

```
1 joue !
1 joue et fait un 6
Quel pion voulez-vous bouger ? :
1 - pion 1
2 - pion 2
3 - pion 3
4 - pion 4
Votre choix : 1
ecurie Jaune : [J][J][J][J]
ecurie Bleu  : [B][B][B]
ecurie Rouge : [R][R][R][R]
ecurie Vert  : [V][V][V][V]
plateau : [-][-][-][-][-][-][-][-][-][-][-][-][-][B][-][-][-][-]
echelle Jaune : [-][-][-][-][-][-]
echelle Bleu  : [-][-][-][-][-][-]
echelle Rouge : [-][-][-][-][-][-]
echelle Vert  : [-][-][-][-][-][-]
```

Une fois qu’un pion a terminé un tour, il est retiré du jeu et n’est donc plus visible.

```
le pion a terminé sa course!
```

Les pions mangés sont également affichés si un joueur mange un ou plusieurs pions.

```
le pion 1 a été mangé
```

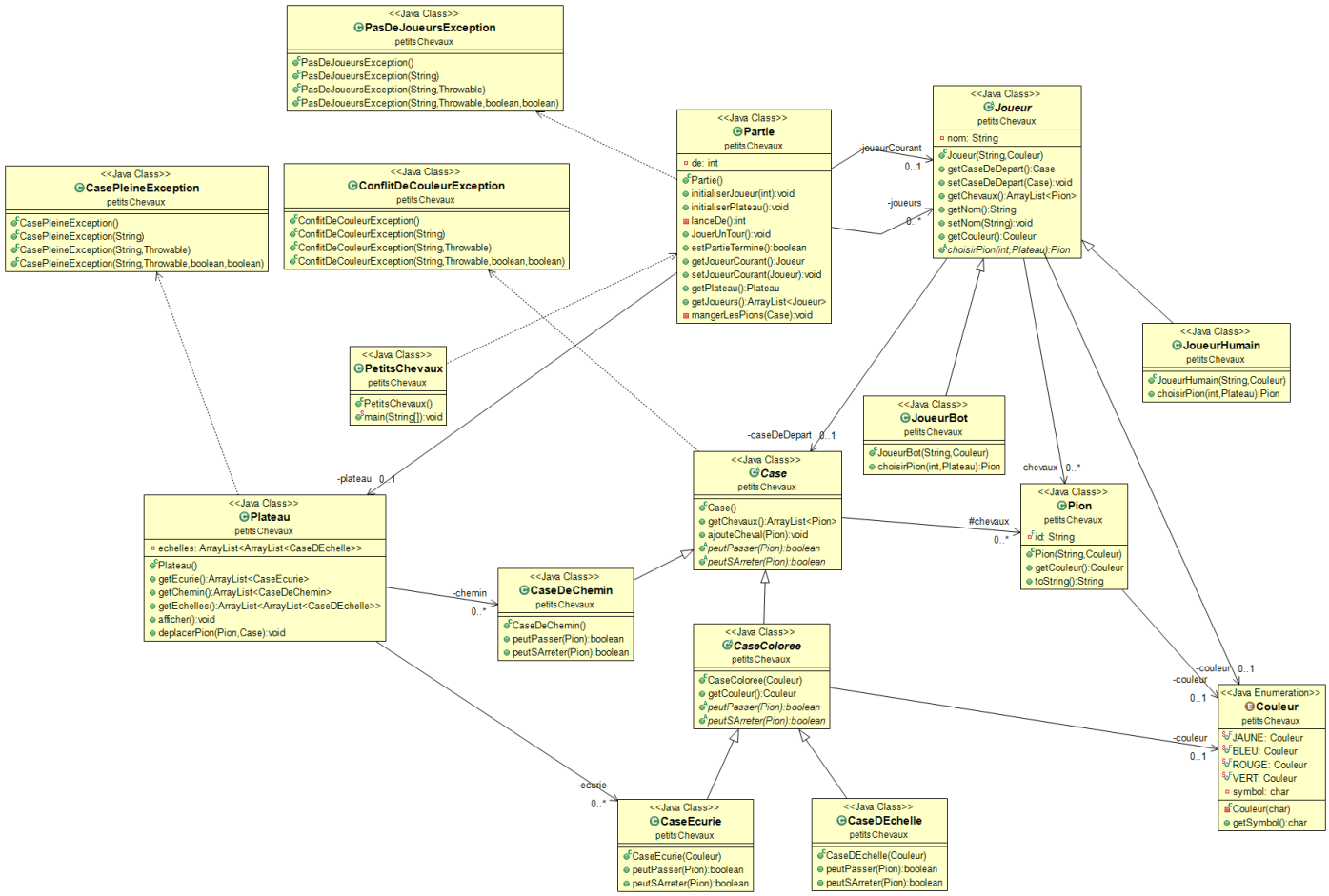
La fin de partie est annoncée par le joueur gagnant.

```
le Bot 1 a gagné !
```

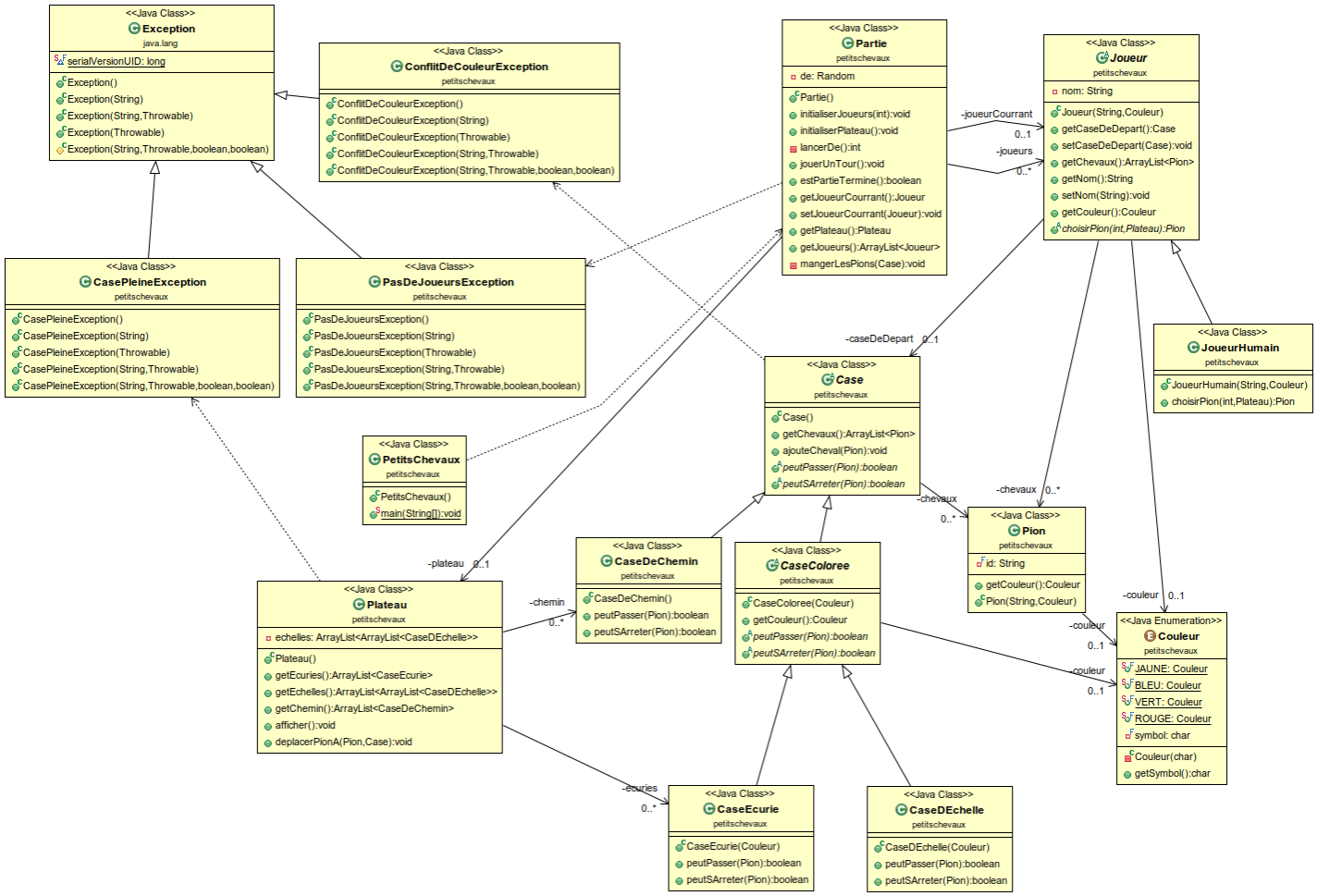
VI- Conclusion

Ce projet a été une occasion de mettre en pratique de manière plus concrètes les connaissances que nous avons apprise en Java et donc POO. Il nous a permis d'une part de renforcer notre expérience en termes de programmation en mettant en pratique un langage sur une durée plus conséquente et à l'échelle d'un projet. Mais, d'une autre part, ce projet nous a permis d'apprendre à gérer les différentes problématiques en ce qui concerne la compréhension d'un diagramme de classes et l'application de certaines méthodes, et la recherche de méthodes plus efficaces pour améliorer la lisibilité et la performance du code. Enfin, ce projet a bien sûr renforcé notre expérience en termes de travail de groupe et de la gestion de la répartition du travail.

Annexe 1


<<Java Class>>
PasDeJoueursException
petitsChevaux
PasDeJoueursException()
PasDeJoueursException(String)
PasDeJoueursException(String,Throwable)
PasDeJoueursException(String,Throwable,boolean,boolean)
<<Java Class>>
Partie
petitsChevaux
de: int
Partie()
initialiserJoueur(int):void
initialiserPlateau():void
lanceDe():int
JouerUnTour():void
estPartieTermine():boolean
getJoueurCourant():Joueur
setJoueurCourant(Joueur):void
getPlateau():Plateau
getJoueurs():ArrayList<Joueur>
mangerLesPions(Case):void
-joueurCourant
0..1
-joueurs
0..*
<<Java Class>>
Joueur
petitsChevaux
nom: String
Joueur(String,Couleur)
getCaseDeDepart():Case
setCaseDeDepart(Case):void
getChevaux():ArrayList<Pion>
getNom():String
setNom(String):void
getCouleur():Couleur
choisirPion(int,Plateau):Pion
<<Java Class>>
CasePleineException
petitsChevaux
CasePleineException()
CasePleineException(String)
CasePleineException(String,Throwable)
CasePleineException(String,Throwable,boolean,boolean)
<<Java Class>>
ConflitDeCouleurException
petitsChevaux
ConflitDeCouleurException()
ConflitDeCouleurException(String)
ConflitDeCouleurException(String,Throwable)
ConflitDeCouleurException(String,Throwable,boolean,boolean)
<<Java Class>>
JoueurHumain
petitsChevaux
JoueurHumain(String,Couleur)
choisirPion(int,Plateau):Pion
<<Java Class>>
PetitsChevaux
petitsChevaux
PetitsChevaux()
main(String[]):void
<<Java Class>>
JoueurBot
petitsChevaux
JoueurBot(String,Couleur)
choisirPion(int,Plateau):Pion
-caseDeDepart
0..1
-chevaux
0..*
-plateau
0..1
<<Java Class>>
Case
petitsChevaux
Case()
getChevaux():ArrayList<Pion>
ajouteCheval(Pion):void
peutPasser(Pion):boolean
peutSArreter(Pion):boolean
<<Java Class>>
Pion
petitsChevaux
id: String
Pion(String,Couleur)
getCouleur():Couleur
toString():String
#chevaux
0..*
<<Java Class>>
Plateau
petitsChevaux
echelles: ArrayList<ArrayList<CaseDEchelle>>
Plateau()
getEcurie():ArrayList<CaseEcurie>
getChemin():ArrayList<CaseDeChemin>
getEchelles():ArrayList<ArrayList<CaseDEchelle>>
afficher():void
deplacerPion(Pion,Case):void
-chemin
0..*
<<Java Class>>
CaseDeChemin
petitsChevaux
CaseDeChemin()
peutPasser(Pion):boolean
peutSArreter(Pion):boolean
<<Java Class>>
CaseColoree
petitsChevaux
CaseColoree(Couleur)
getCouleur():Couleur
peutPasser(Pion):boolean
peutSArreter(Pion):boolean
-couleur
0..1
-couleur
0..1
<<Java Enumeration>>
Couleur
petitsChevaux
JAUNE: Couleur
BLEU: Couleur
ROUGE: Couleur
VERT: Couleur
symbol: char
Couleur(char)
getSymbol():char
-ecurie
0..*
<<Java Class>>
CaseEcurie
petitsChevaux
CaseEcurie(Couleur)
peutPasser(Pion):boolean
peutSArreter(Pion):boolean
<<Java Class>>
CaseDEchelle
petitsChevaux
CaseDEchelle(Couleur)
peutPasser(Pion):boolean
peutSArreter(Pion):boolean

Annexe 2


<<Java Class>>
Exception
java.lang
serialVersionUID: long
Exception()
Exception(String)
Exception(String,Throwable)
Exception(Throwable)
Exception(String,Throwable,boolean,boolean)
<<Java Class>>
ConflitDeCouleurException
petitschevaux
ConflitDeCouleurException()
ConflitDeCouleurException(String)
ConflitDeCouleurException(Throwable)
ConflitDeCouleurException(String,Throwable)
ConflitDeCouleurException(String,Throwable,boolean,boolean)
<<Java Class>>
Partie
petitschevaux
de: Random
Partie()
initialiserJoueurs(int):void
initialiserPlateau():void
lancerDe():int
jouerUnTour():void
estPartieTermine():boolean
getJoueurCourrant():Joueur
setJoueurCourrant(Joueur):void
getPlateau():Plateau
getJoueurs():ArrayList<Joueur>
mangerLesPions(Case):void
-joueurCourrant
0..1
-joueurs
0..*
<<Java Class>>
Joueur
petitschevaux
nom: String
Joueur(String,Couleur)
getCaseDeDepart():Case
setCaseDeDepart(Case):void
getChevaux():ArrayList<Pion>
getNom():String
setNom(String):void
getCouleur():Couleur
choisirPion(int,Plateau):Pion
<<Java Class>>
CasePleineException
petitschevaux
CasePleineException()
CasePleineException(String)
CasePleineException(Throwable)
CasePleineException(String,Throwable)
CasePleineException(String,Throwable,boolean,boolean)
<<Java Class>>
PasDeJoueursException
petitschevaux
PasDeJoueursException()
PasDeJoueursException(String)
PasDeJoueursException(Throwable)
PasDeJoueursException(String,Throwable)
PasDeJoueursException(String,Throwable,boolean,boolean)
-caseDeDepart
0..1
<<Java Class>>
JoueurHumain
petitschevaux
JoueurHumain(String,Couleur)
choisirPion(int,Plateau):Pion
<<Java Class>>
Case
petitschevaux
Case()
getChevaux():ArrayList<Pion>
ajouteCheval(Pion):void
peutPasser(Pion):boolean
peutSArreter(Pion):boolean
<<Java Class>>
PetitsChevaux
petitschevaux
PetitsChevaux()
main(String[]):void
-chevaux
0..*
-chevaux
0..*
<<Java Class>>
Pion
petitschevaux
id: String
getCouleur():Couleur
Pion(String,Couleur)
-plateau
0..1
<<Java Class>>
CaseDeChemin
petitschevaux
CaseDeChemin()
peutPasser(Pion):boolean
peutSArreter(Pion):boolean
<<Java Class>>
CaseColoree
petitschevaux
CaseColoree(Couleur)
getCouleur():Couleur
peutPasser(Pion):boolean
peutSArreter(Pion):boolean
-couleur
0..1
<<Java Class>>
Plateau
petitschevaux
echelles: ArrayList<ArrayList<CaseDEchelle>>
Plateau()
getEcuries():ArrayList<CaseEcurie>
getEchelles():ArrayList<ArrayList<CaseDEchelle>>
getChemin():ArrayList<CaseDeChemin>
afficher():void
deplacerPionA(Pion,Case):void
-chemin
0..*
-couleur
0..1
<<Java Enumeration>>
Couleur
petitschevaux
JAUNE: Couleur
BLEU: Couleur
VERT: Couleur
ROUGE: Couleur
symbol: char
Couleur(char)
getSymbol():char
-ecuries
0..*
<<Java Class>>
CaseEcurie
petitschevaux
CaseEcurie(Couleur)
peutPasser(Pion):boolean
peutSArreter(Pion):boolean
<<Java Class>>
CaseDEchelle
petitschevaux
CaseDEchelle(Couleur)
peutPasser(Pion):boolean
peutSArreter(Pion):boolean