Projet JAVA L2 2013 - Jeu d'Othello

# Notation

Pour rappel , les éléments suivantes font parti intégrante de la notation :

* Convention de nommage
* Commentaires
* Javadoc
* Cas limites
* Indentation, Convention d'écriture [Java](http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html)
* Conception, Découpage en petites fonction, Factorisation du code, structure algorithmique
* Professionnalisme (Délai de rendu, origine du code, etc...)
* Suivi des spécifications (noms des éléments demandés respectés)

# Forme :

* A chaque début de cours la partie précédente réalisée devra être rendue
* Le projet final sera réalisé en groupe de 2 personnes.
* Le code source devra être livré sous la forme d'un fichier zip. Le fichier zip devra porter le nom du binôme
* Le projet devra être rendu au maximum 2 semaines après la fin du dernier cours de Java L3

# Enoncé:

Le but du projet est de manipuler le langage Java, ainsi que de démontrer les avantages de la programmation par interface. Dans un premier temps nous réaliserons de façon itérative sous la forme de TP, un jeu de Morpion.

Le projet consistera à ajouter une implémentation permettant de basculer d'un jeu de Morpion vers un jeu d'Othello. Les règles sont détaillées sur le site wikipedia,: http://fr.wikipedia.org/wiki/Othello\_%28jeu%29

Afin de simplifier au maximum, nous considèrerons que les noirs sont représentés par des X et les blancs par des O.

Partie 1 : Model

Dans cette première partie du TP nous allons coder les classes correspondantes au modèle manipulé par l'application.

Coordonnee

Créez une classe Coordonnee dans un package fr.epsi.game.data.

Implémentez les méthodes et constructeurs suivants :

/\*\*

\* Constructeur pour une coordonnee a partir de sa representation text (A5, C6)

\* **@param** coord La coordonnee

\*/

**public** Coordonnee(String coord){}

/\*\*

\* Constructeur pour une coordonnee

\* **@param** ligne le numero de la ligne

\* **@param** colonne le numero de la colonne

\*/

**public** Coordonnee(**int** ligne, **int** colonne) {}

/\*\*

\* Retourne l'indice de la ligne

\* **@return** the ligne

\*/

**public** **int** getLigne() {}

/\*\*

\* Retourne l'indice de la colonne

\* **@return** the colonne

\*/

**public** **int** getColonne() {}

/\*\*

\* Redefini le toString pour afficher une coordonnee sous la forme A0, B5, D1

\*/

**public** String toString() {}

BiDimension

Créez une classe générique BiDimension<T> dans un package fr.epsi.game.data. Une implémentation possible du stockage interne est la suivante : Object[][] tableau;

Implémentez les méthodes et constructeurs suivants :

/\*\*

\* Constructeur pour un tableau a 2 dimensions

\* **@param** nbLignes

\* **@param** nbColonnes

\*/

**public** BiDimension(**int** nbLignes, **int** nbColonnes) {}

/\*\*

\* Positionne la valeur <obj> dans la case de coordonnees (ligne, colonne)

\* **@param** ligne la ligne (commence à 0)

\* **@param** colonne la colonne (commence à 0)

\* **@param** obj la valeur

\*/

**public** **void** set(**int** ligne, **int** colonne, T obj) {}

/\*\*

\* Retourne la valeur contenu dans la case de coordonnees (ligne, colonne)

\* **@param** ligne la ligne (commence à 0)

\* **@param** colonne la colonne (commence à 0)

\* **@return** La valeur

\*/

@SuppressWarnings("unchecked")

**public** T get(**int** ligne, **int** colonne) {}

/\*\*

\* Positionne la valeur <obj> dans la case de coordonnees (coord)

\* **@param** coord la coordonnee

\* **@param** obj la valeur

\*/

**public** **void** set(Coordonnee coord, T obj) {}

/\*\*

\* Retourne la valeur contenu dans la case de coordonnees (coord)

\* **@param** coord la coordonnee

\* **@return** La valeur

\*/

**public** T get(Coordonnee coord) {}

/\*\*

\* Retourne le nombre de ligne du tableau a 2 dimensions

\* **@return** nb de lignes

\*/

**public** **int** getNbLignes() {}

/\*\*

\* Retourne le nombre de colonnes du tableau a 2 dimensions

\* **@return** nb de colonnes

\*/

**public** **int** getNbColonnes() {}

/\*\*

\* Recherche un element dans le tableau et renvoi ses coordonnes s'il est trouve, null sinon

\* **@param** obj L'objet en question

\* **@return** Les coordonnees de l'objet

\*/

**public** Coordonnee search(T obj) {}

GameModel

Dans le package fr.epsi.game.data, créez les 2 énumérations suivantes :

* ButtonState, contenant les valeurs 0 et X
* Status contenant les valeurs suivantes :
  + GAGNE,
  + PERDU,
  + MON\_TOUR,
  + SON\_TOUR,
  + ATTENTE\_PARTENAIRE

Dans le package fr.epsi.game créez l'interface suivante :   
**public** **interface** GameModel {

/\*\*

\* Recupere l'etat d'un bouton a la coordonnee passee en parametre

\* **@param** coord La coordonnee

\* **@return** Le statut du bouton

\*/

ButtonState get(Coordonnee coord);

/\*\*

\* Recupere l'etat d'un bouton a la coordonnee passee en parametre

\* **@param** int La coordonnee de la ligne

\* **@param** int La coordonnee de la colonne

\* **@return** Le statut du bouton

\*/

ButtonState get(**int** ligne, **int** colonne);

/\*\*

\* Retourne le statut du jeu

\* **@return** the status

\*/

Status getStatus();

/\*\*

\* Change le statut du jeu et notifie les listener associes

\* **@param** status the status to set

\*/

**void** setStatus(Status status);

/\*\*

\* Change l'etat d'un bouton et notifie les listeners associes

\* **@param** coord La coordonnee du bouton

\* **@param** state Le nouvel etat du bouton

\*/

**void** setButtonState(Coordonnee coord, ButtonState state);

/\*\*

\* Retourne les dimensions de la grilles sous forme de coordonnees

\* **@return** Les coordonnees representant la grille

\*/

Coordonnee getDimensionGrille();

/\*\*

\* Renseigne le model sur le signe de l'utilisateur (X ou O) et le status du jeu

\* **@param** state

\* **@param** status

\*/

**void** setMonStatus(ButtonState state);

}

Dans le package fr.epsi.game.impl, Créez une classe GameModelImpl qui implémentes les méthodes de GameModel. Pour la représentation interne des ButtonState réutilisez la classe générique Bi Dimension que vous avez précédemment codé.

GameModel

Cette classe aura pour constructeur

/\*\*

\* Constructeur d'un model de nbLignes x nbColonnes

\* **@param** nbLignes nb de lignes de la grille

\* **@param** nbColonnes nb de colonnes de la grille

\*/

**public** GameModelImpl(**int** nbLignes, **int** nbColonnes)

L'interface parle de notifier des Listener. Dans cette première implémentation vous afficherez simplement un message sur la console Java.

Partie 2 : IHM

Cette partie va permettre la création de l'IHM.

Dans le package fr.epsi.game.ui, Créez une classe FenetreClient qui dans son constucteur prend comme paramètre un GameModel. Le modèle sera utile pour récupérer les dimensions de la Grille a afficher.

Utilisez un GridLayout pour représenté la grille. Celle ci sera composé de JButton. Afin de faciliter la récupération des JButton par la suite créer un attribut de type BiDimension<JButton>. Cet objet permettra d'accéder à un bouton depuis sa coordonnée.

Partie 3 : Controller et Listener

Dans le package fr.epsi.game.listener, créez les 2 interfaces suivantes

/\*\*

\* Listener utilise pour le changement d'etat d'un bouton

\* **@author** spt001

\*

\*/

**public** **interface** ButtonStateListener {

/\*\*

\* Methode appelee pour le changement d'etat d'un bouton

\* **@param** coord La coordonnee d'un bouton

\* **@param** state Le nouvel etat

\*/

**public** **void** buttonStateChange(Coordonnee coord, ButtonState state);

}

et

/\*\*

\* Listener appele lors d'un changement de statut du jeu

\* **@author** spt001

\*

\*/

**public** **interface** GameStatusListener {

/\*\*

\* Methode declenchee lorsque le statut du jeu change

\* **@param** status le statut du jeu

\*/

**public** **void** statusChanged(Status status);

}

GameModel, gestion d'évènements

Ajoutez a votre interface GameModel les 3 méthodes suivantes :

Les listener sont gérés sous forme de liste de telle façon que pour une action donnée plusieurs listeners puissent être informés.

/\*\*

\* Ajoute un listener notifie uniquement sur les actions de l'utilisateur gere par ce modele

\* **@param** listener

\*/

**void** addOtherButtonStateListener(ButtonStateListener listener);

/\*\*

\* Ajoute un listener notifie sur toutes les actions de jeu (adversaire et utilisateur local)

\* Cette methode est en general appele par l'IHM

\* **@param** listener

\*/

**void** addButtonStateListener(ButtonStateListener listener);

/\*\*

\* Ajoute un listener notifie lors d'un changement d'etat dans le jeu

\* **@param** listener Le listener a ajouter

\*/

**void** addGameStatusListener(GameStatusListener listener);

Implémentez ces méthodes dans votre GameModelImpl et remplacer les System.out par les notifications qui correspondent.

Modifiez la classe FenetreClient afin qu'elle implémente les Listeners ButtonStateListener et GameStatusListener.

Pour une notification par ButtonStateListener, on modifiera le libellé du bouton. Pour un GameStatusListener, on modifiera le titre de la fenêtre et pour certaines actions importante (fin de partie) on affichera une boite de dialogue à l'aide de la méthode JOptionPane. showMessageDialog.

GameController

Dans le package fr.epsi.game, créez l'interfaces suivante:

/\*\*

\* Interface d'un jeu tour a tour de type Morpion ou Othello

\* Chaque joueur joue a son tour.

\* **@author** spt001

\*

\*/

**public** **interface** GameController {

/\*\*

\* Cette methode est appelee par la vue lorsque l'utilisateur joue une case

\* **@param** coord La coordonnee de la case jouee

\* **@throws** RuleException En cas d'infraction aux regles du jeu

\*/

**public** **void** jeJoue(Coordonnee coord) **throws** RuleException;

/\*\*

\* Cette methode est appelee par le protocole reseau lorsque l'adversaire joue une case

\* **@param** coord La coordonee de la case jouee

\* **@throws** RuleException En cas d'infraction aux regles du jeu

\*/

**public** **void** ilJoue(Coordonnee coord) **throws** RuleException;

/\*\*

\* Renvoie le modele associe a ce controller

\* **@return** Le modele

\*/

**public** GameModel getModel();

}

Dans le package fr.epsi.game.impl, créez la classe MorpionControllerImpl qui implémente GameController:

Cette classe est responsable de la logique du jeu. C'est ici qu'on effectuera les contrôles de fin de jeu. MorpionController sera appelé par l'IHM FenetreClient et par la protocole réseau. Le controller ne mettra pas a jour directement l'IHM il passera par le modèle qui lui notifiera l'IHM.

FenetreClient

Ajouter dans la classe FenetreClient un listener sur les JButton afin que sur chaque appuie on notifie la classe MorpionController en appelant la méthode jeJoue.

Partie 4 : Jeu d'othello

Créez une classe OthelloControllerImpl qui implémente GameController. Cette classe contiendra la logique du jeu d'othello.

Si vous avez implémenté les autres parties correctement, le reste du code reste inchangé excepté la classe de test.

Annexes : Tests

Une classe de test pourrait ressembler a ceci :

/\*\*

\* Launch the application.

\*/

**public** **static** **void** main(String[] args) {

EventQueue.*invokeLater*(**new** Runnable() {

**public** **void** run() {

**try** {

MorpionController state1 = **new** MorpionController(ButtonState.*X*);

FenetreClient frame1 = **new** FenetreClient(state1);

frame1.setVisible(**true**);

MorpionController state2 = **new** MorpionController(ButtonState.*O*);

FenetreClient frame2 = **new** FenetreClient(state2);

frame2.setVisible(**true**);

NetworkFake.*newGame*(state1, state2);

} **catch** (Exception e) {

e.printStackTrace();

}

}

});

}

Pour simuler le réseau créez dans le package fr.epsi.game.net la classe suivante :

**public** **class** NetworkFake {

**private** **final** **static** Logger *LOGGER* = Logger.*getLogger*("NetworkFake");

**public** **static** **void** newGame(**final** MorpionController joueur1, **final** MorpionController joueur2) {

joueur1.getModel().addOtherButtonStateListener(**new** ButtonStateListener() {

@Override

**public** **void** buttonStateChange(**final** Coordonnee coord, ButtonState state) {

SwingUtilities.*invokeLater*(**new** Runnable(){

**public** **void** run() {

**try** {

joueur2.ilJoue(coord);

} **catch** (RuleException ex){

*LOGGER*.log(Level.*SEVERE*, "Violation des regles", ex);

}

}

});

}

});

joueur2.getModel().addOtherButtonStateListener(**new** ButtonStateListener() {

@Override

**public** **void** buttonStateChange(**final** Coordonnee coord, ButtonState state) {

SwingUtilities.*invokeLater*(**new** Runnable(){

**public** **void** run() {

**try** {

joueur1.ilJoue(coord);

} **catch** (RuleException ex){

*LOGGER*.log(Level.*SEVERE*, "Violation des regles", ex);

}

}

});

}

});

joueur2.getModel().setStatus(Status.*SON\_TOUR*);

joueur1.getModel().setStatus(Status.*MON\_TOUR*);

}

}