CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL MIDI - PYRENEES

NFA 032 – Travaux pratiques

IPST-CNAM Licence

PROJET

Années 2017-2018
TOULOUSE
J. LAFORGUE

1. LE BESOIN

On se propose de faire la suite du projet réalisé dans le cadre du cours NFA 031.

Pour ceux qui n'aurait pas fait ce projet, il est impératif que vous preniez connaissance du sujet du projet NFA031 2017-2018 et de sa correction (le projet joint au sujet). Le projet se faisant en salle de TP et chez vous sous Eclipse, il est important que vous connaissiez l'usage de cet environnement de programmation.

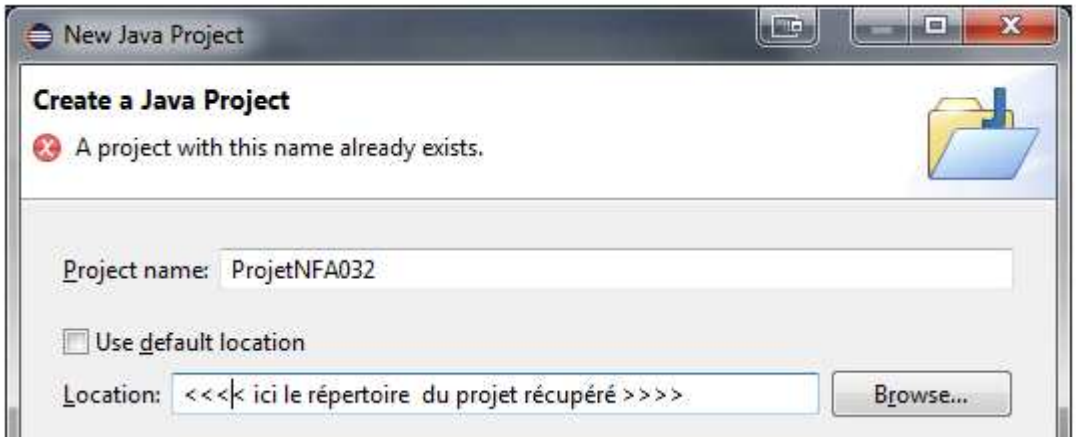
Pour commencer, vous devez partir du projet fourni avec le sujet qui est le projet de NFA031 dans lequel j'ai fait les modifications suivantes :

- la programmation de 2 classes supplémentaires de jeu : Morpion et Go
- **la classe Partie contient tous les attributs des parties d'Othello, de Morpion et de Go et le fichier des parties Parties.txt a évolué en conséquence**
- on ne crée plus 2 parties à la fin du jeu mais 1 seule contenant les noms des deux joueurs
- la partie n'est plus créée par PFJeu mais par la classe de jeu (ex: Othello)
- l'ihm de grille (CanvasIHM) n'est plus créée par PFJeu mais par le jeu (la grille est différente en fonction des jeux). Pour cela j'ai dû faire évoluer la classe Formulaire.
- mis en place d'une politique de sécurité pour gérer les fichiers et les sockets

Première chose à faire : enregistrer et décompresser le projet joint au sujet dans un répertoire de votre choix qui sera le répertoire définitif de votre projet (différent du workspace Eclipse)

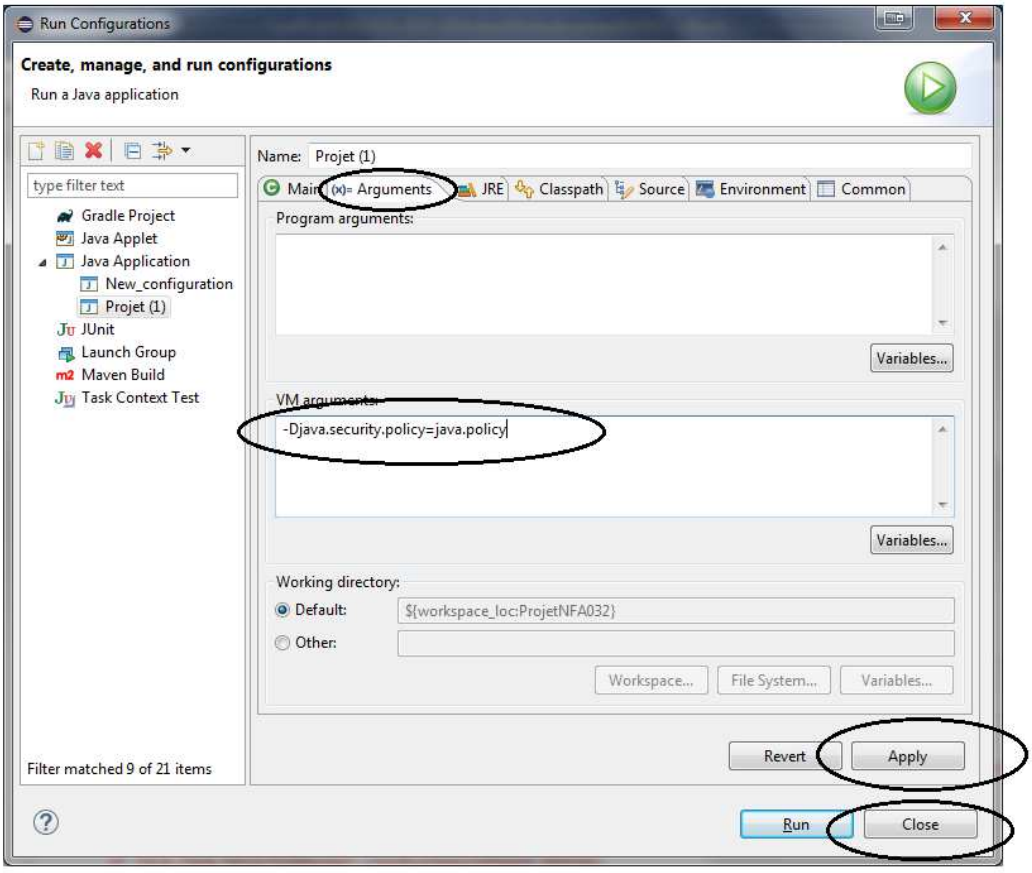
Puis , à faire sous Eclipse :

- créer un projet Eclipse : File>New>Java Project



- Mettre dans Localtion le répertoire "projet".
- Puis Finish

- Exécuter une première fois le projet ce qui va créer une configuration d'exécution.
- Il faut modifier la configuration d'exécution du projet afind d'ajouter un paramètre de la JVM pour configurer la politique de sécurité ou mettre en commentaire la 1ère ligne du main : System.setSecurityManager(new SecurityManager()).
- Mise à jour de la configuration d'exécution :
  - Run > Run Configurations…



Dans l'onglet "Arguments" ajouter -Djava.security.policy=java.policy, puis Apply, puis Close.

A terme le programme devra être capable de :

- [FUNC 1] importer les fichiers de données au format txt
- [FUNC 2] à iso-fonctionnalité implémenter les parties des joueurs sous la forme d'une arborescence d'héritage et gérer la collection des parties sous la forme d'une collection polymorphe
- [FUNC 3] gérer plusieurs jeux (Othello, Morpion, Go) avec l'utilisation d'une interface
- [FUNC 4] sauvegarder et charger la plate-forme de jeu dans un fichier binaire
- [FUNC 5] jouer en réseau avec l'utilisation des sockets
- [FUNC 6] ajouter un nouveau jeu : le jeu de Puissance4
- [FUNC 7] (optionnel) le traitement récursif qui élimine les pions en prises du jeu de Go.

Ces fonctionnalités sont détaillées dans les 3 étapes qui suivent correspondantes au 3 séances de TP.

Les séances de TP sont utilisées pour réaliser la fusion des codes du binôme, pour que je puisse faire un point d'avancement de vos développements et vous débloquer dans la conception et le codage de votre projet.

2. ETAPE 1

Dans cette étape on se propose de réaliser les fonctionnalités [FUNC 1] [FUNC 2] et [FUNC 3].

2.1. [FUNC 1] Importer les fichiers de données au format txt

Il s'agit d'ajouter dans **IHMPFJeu** deux zones de texte permettant de saisir le nom des fichiers txt et un bouton "Importer". Par défaut les zones de texte contiennent "Parties.txt" et "Joueurs.txt".
Si un fichier saisi n'existe pas alors afficher un texte d'erreur.
Si la zone de saisie d'un fichier est vide alors pas d'importation pour ce fichier.

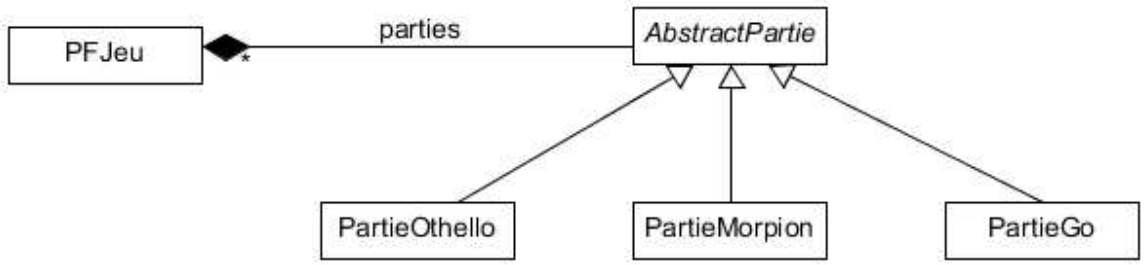
Dans la classe **IHMPFJeu**, il faut déplacer le code suivant dans la méthode **submit** qui est appelée quand on clique sur le bouton d'importation et modifier la méthode **pfjeu.initialiser()** afin qu'elle prenne en entrée les noms des deux fichiers récupérés de l'ihm :

```
// Initialisation de l'applicatif
//  On affiche le resultat de l'initialisation
//
pfjeu.initialiser();

// On initialiser la liste des joueurs
//
ArrayList<Joueur> joueurs = pfjeu.getJoueurs();
String[] tab = new String[joueurs.size()];
for(int i=0;i<tab.length;i++) tab[i]=joueurs.get(i).getIdent();
form.setListData("ADVERSAIRES",tab);
```

2.2. [FUNC 2] Héritage et collection polymorphe

La classe **Partie** telle qu'elle est codée dans le projet fourni n'est pas optimisée. Elle contient tous les attributs spécifiques à chacune des parties d'Othello, de Morpion et de Go.
L'objectif est donc de créer une arborescence d'héritage à partir d'une classe abstraite et qui permettra de gérer la collection des parties sous la forme d'une collection polymorphe.



Il faut mettre dans la classe **AbstractPartie** tous les attributs communs à tous les types de partie et mettre dans chacune des classes **PartieOthello**, **PartieMorpion**, **PartieGo** les attributs spécifiques respectivement à chacun des jeux.

Cette nouvelle façon d'implémenter les parties est à iso-fonctionnalités, c'est-à-dire que le programme fonctionne pour l'utilisateur de la même manière.

Les modifications à réaliser sont :

- l'écriture de ces nouvelles classes (Vous copiez le fichier actuel Partie.java en PartieOthello.java, PartieMorpion.java, PartieGo.java, vous renommez Partie.java en AbstractPartie.java, puis vous les modifiez)

- la collection parties de **PFJeu** devient une collection polymorphe sur **AbstractPartie**
- la méthode initialiser de **PFJeu** crée des objets des différentes classes de partie de jeu en fonction du nom du jeu lu dans le fichier texte
- dans les classes **Othello**, **Morpion** et **Go**, la méthode **getPartie** crée spécifiquement la partie.

2.3. Gérer plusieurs jeux

Il s'agit de rajouter dans l'ihm une liste de choix (*addListeChoix de Formulaire*) des jeux "Othello", "Morpion", "Go".
Quand on clique sur démarrer, en fonction du jeu sélectionné dans la liste de choix, on crée le jeu correspondant et on le démarre.

Pour cela, il faut créer et utiliser l' interface **Jeu** suivante :

```
public interface Jeu
{
  public void demarrer();
  public boolean getDemarree();
  public void setDemarree(boolean demarree);
  public AbstractPartie getPartie(int numeroPartie,
                                  String identJoueurCourant,
                                  String identAdversaire);
  public boolean selectionnerCase(int x,int y);
}
```

Les classes de jeu doivent implémenter l'interface **Jeu**.

3. ETAPE 2

Dans cette étape on se propose de réaliser la fonctionnalité [FUNC 4].

On crée dans l'IHM deux boutons : "Sauvegarder" et "Charger".

Sur le bouton "Sauvegarder", le programme crée un fichier binaire dans lequel on sauvegarde la plate-forme de jeu c'est-à-dire les deux collections *joueurs* et *parties*.
Le fichier créé est :

data/PFJeu.bin

Sur le bouton "Charger", le programme lit le fichier binaire "data/PFJeu.bin" permettant d'initialiser les deux collections *joueurs* et *parties*.

Il faut donc coder dans les classes **Joueur** et les classes **AbstractPartie**, **PartieOthello**, **PartieMorpion**, **PartieGo** les méthodes permettant d'écrire et lire les attributs de ces classes sous forme binaire.

On utilise les classe **DataInputStream** et **DataOutputStream** pour lire et écrire les éléments du fichier binaire.

4. ETAPE 3

Dans cette étape on se propose de réaliser la fonctionnalité [FUNC 5] et [FUNC 6].
Puis, si possible [FUNC 7] (optionnel).

4.1. Jouer en réseau avec l'utilisation des sockets

Le projet qui a été fourni avec le sujet est déjà configuré et pré-codé pour pouvoir jouer en réseau.
Pour vérifier que la communication fonctionne, j'ai créé un bouton "Tester". Vous choisissez la machine (adversaire) avec laquelle vous voulez communiquer.

Le principe est que chaque joueur exécute tout naturellement le projet chacun sur sa machine. Chacun des joueurs s'identifie avec le nom de sa machine et choisit le nom de la machine de son adversaire.
Ensuite quand un joueur joue un coup, ce coup est envoyé à l'adversaire.

Pour pouvoir mettre au point le projet sur un même machine, les deux scripts runLocal1.bat et runLocal2.bat permettent d'exécuter deux projets sur la même machine qui communiquent entre eux.
Si vous voulez exécuter sous Eclipse pour débuguer les programmes, il faudra créer une ou deux configurations d'exécution afin de préciser les paramètres d'exécution.

Le message à coder est **SELECTIONNER_CASE** ***pos_x pos_y*** qui est envoyé à l'adversaire quand le joueur joue un coup.
Pour écrire et lire ces 3 informations, on utilise les classes DataInputStream et DataOutputStream. Leurs types sont String, int, int (on utilise les primitives readUTF/writeUTF et readInt/writeInt).
Le respect de ce format du message est important afin que tous les groupes de TP soient compatibles.

Quand on teste en local, l'identifiant des joueurs qu'il faut utiliser est "localhost".

Normalement, on devrait coder des messages qui permettent de démarrer, arrêter la partie et d'autoriser ou non un joueur de jouer. Mais cela devient complexe à mettre au point.

4.2. Ajouter un nouveau jeu : le jeu de Puissance4

Règles du jeu Puissance4 :
Le but du jeu est d'aligner une suite de 4 pions de même couleur sur une grille comptant 6 rangées et 7 colonnes. Chaque joueur dispose de pions d'une couleur différente. Tour à tour les deux joueurs placent un pion dans la colonne de leur choix, le pion doit être à la position la plus basse possible dans la dite colonne à la suite de quoi c'est à l'adversaire de jouer. Le vainqueur est le joueur qui réalise le premier un alignement (horizontal, vertical ou diagonal) consécutif d'au moins quatre pions de sa couleur.

Un début de partie de Puissance 4.

Faire la classe **Puissance4** et **PartiePuissance4**.
Les intégrés dans **PFJeu** et **IHMPFJeu**.

On peut s'apercevoir que le jeu de puissance4 est comme le jeu de morpion sauf que la grille est moins grande, que la longueur de la ligne gagnante est de 4 au lieu de 5, que le premier qui aligne gagne, que l'on ne peut pas jouer n'importe où et que le jeu s'arrête dès qu'un joueur a fait une ligne de 4.

4.3. Eliminer les pions en prises du jeu de Go (optionnel)

Ce codage est optionnel.

Quand un coup de Go permet d'étouffer des pions de l'adversaire alors il faut les compter afin d'incrémenter le nombre de pions pris à l'adversaire et les effacer du tableau de jeu.

A la fin d'une partie de Go, il faudrait ensuite compter le nombre de point des territoires vivants. Cela est un algorithme encore plus complexe.

5. QUELQUES PRECISIONS

Les règles de codage que vous devez suivre sont :

- sauf pour les constantes, les attributs de toutes les classes doivent être privés
- toutes les classes commencent par une majuscule
- tous les attributs, méthodes, paramètres et variables commencent par une minuscule
- les noms de tous les répertoires de package sont en minuscule
- les packages créés commencent tous par "fr.cnam"
- une indentation stricte est respectée dans tout le code
- un minimum de commentaire est nécessaire afin de comprendre ce que fait le code. Surtout quand ce que fait le code n'est pas décrit précisément dans le sujet.

Conseils :

- Ecrivez peu de code à chaque fois afin d'avoir toujours un code qui se compile correctement
- Après chaque étape ou sous-étape, le programme s'exécutant correctement, faites une sauvegarde des sources. Cela permet de pouvoir revenir en arrière si besoin et permet aussi de pouvoir donner à l'enseignant une version du programme qui s'exécute correctement et qui remplit correctement une partie des fonctionnalités demandées dans le sujet.
- Dans un premier temps ne faites pas plus que le sujet ne le demande. Une fois que le programme respecte le sujet, vous faites une sauvegarde et vous pouvez apporter des améliorations qui sont toujours appréciées par le correcteur, à condition qu'elles soient bien commentées.

Dans le fichier Projet.java, vous renseigner les noms et prénoms du groupe.

6. PLANNING

Vous avez 3 séances de TP pour réaliser ce projet

Le projet sera ramassé 1/4 heure avant la fin de la dernière séance de TP (le temps de les ramasser tous).

Il est donc **impératif** que, au moins un membre du groupe de TP, soit présent lors de la dernière séance afin de donner son projet.

Je rappelle que, sauf situation exceptionnelle, il est obligatoire d'être présent durant les séances de TP.