Guide utilisateur du squelette

Encadrement : Armelle Bonenfant, Inès de Courchelle, Faustine Maffre,
Vincent Mussot, Ileana Ober

16 décembre 2014

Ce document décrit le mode de fonctionnement et la conception du programme squelette fourni pour la partie Java du *Projet de programmation*.

1 Premier lancement

Avant toute première exécution du programme, vous devez d'abord importer le code téléchargé comme un projet *Eclipse*. Pour cela extrayez le code et copiez-le dans le dossier représentant le *workspace Eclipse*. Ensuite utilisez la fonction *Import* du menu *File* (étape 1) en choisissant l'option *Existing Project into Workspace* du menu *General* (étape 2). Vous devez naviguer jusqu'au dossier que vous venez de copier (étape 3).

Le projet est désormais visible dans la fenêtre *Package explorer* (étape 4). Le code se trouve dans le dossier *v1415vx*. Celui-ci contient les classes de test (*TestServeur*, *TestIHM*, *TestPersonnage* et *TestPotion*) et 6 paquetages touchant chacun à un but différent (modèle du système, interface graphique, serveur, contrôleur, ...). La structuration du code respecte le patron de conception Modèle-Vue-Contrôleur et elle sera détaillée dans la section suivante du document.

Si aucun problème n'a été détecté au moment de l'importation ou à la compilation, le code est prêt à être exécuté. Vous devez suivre les étapes :

1. Exécution du fichier *TestServeur.java* (étape 5). Celui-ci va créer sur votre machine un serveur RMI auxquels différents éléments se connecteront.
2. Exécution du fichier *TestIHM.java* (étape 6). Une arène de jeu sera affichée à l'écran ainsi qu'une fenêtre utilisée par les éléments du jeu pour communiquer leur actions. Celle-ci est connectée au serveur du jeu.
3. Exécution du fichier *TestPersonnage.java* (étape 7) (ou *TestPotion.java*). Un élément du jeu est créé, ainsi que sa représentation graphique, les deux étant gérés par un contrôleur appelé *console* par la suite. La console se connecte au serveur et l'élément est ensuite affiché sur l'arène du jeu. Pour connecter plusieurs éléments sur l'arène du jeu, exécutez ce code le nombre de fois voulu. Comme leurs noms l'indique, *TestPersonnageAlea.java* place aléatoirement le personnage dans l'arène, alors que *TestPersonnageCentre.java* le place au centre. La position initiale de l'élément est donnée comme paramètre passé dans le constructeur de la classe *Console*.

2 Conception : diagramme de classes et explications

La Figure 1 présente le diagramme de classes associé au squelette. Il est structuré en 6 paquetages comme suit :

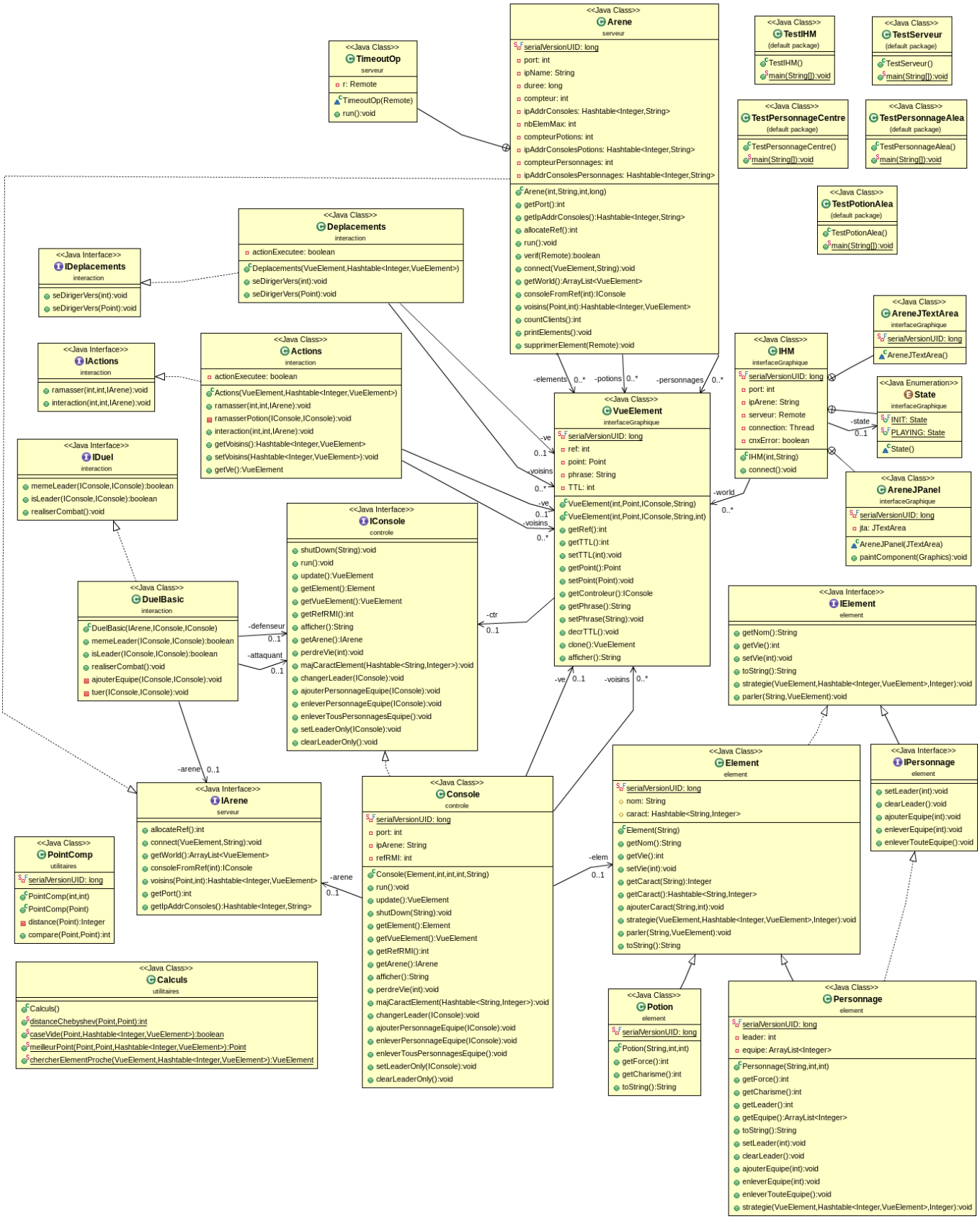
1. *controle* qui fait la liaison entre le modèle du système (personnages et potions), l'interface graphique et le serveur :

- l'interface **IConsole** donne la signature du contrôleur. Cette interface est utilisée pour la communication RMI avec le serveur.
- la classe **Console** implémente l'interface *IConsole*. Elle représente un thread sur le serveur qui peut interagir avec les autres éléments. Un objet de type *Element* et un objet de type *VueElement* sont associés à chaque contrôleur.

2. *element* qui contient la définition des *personnages* et *potions* :
   - l'interface **IElement** donne la signature commune de tous les personnages et potions.
   - la classe **Element** implémente l'interface *IElement*.
   - l'interface **IPersonnage** définit quelques méthodes spécifiques aux personnages.
   - la classe **Personnage** implémente *IPersonnage* et représente un personnage avec sa force, son charisme, son leader et son équipe.
   - la classe **Potion** implémente *IElement* et représente une potion (qui donne un bonus de force et/ou de charisme).
3. *interaction* qui définit les combats auxquels deux personnages peuvent participer :
   - l'interface **IDuel** donne la signature nécessaire pour l'interaction entre deux personnages.
   - la classe **DuelBasic** implémente l'interface *IDuel* en proposant une version basique du combat.
   - l'interface **IActions** donne la signature des actions d'interaction devant passer par le réseau.
   - la classe **Actions** implémente l'interface *IActions*.
   - l'interface **IDeplacements** donne la signature des actions de déplacement devant passer par le réseau.
   - la classe **Deplacements** implémente l'interface *IDeplacements*.
4. *interfaceGraphique* qui définit la représentation graphique de l'arène de jeu et des éléments :
   - la classe **IHM** représente l'arène de jeu. Elle se connecte au serveur pour le synchronisation des éléments qu'elle contient.
   - la classe **VueElement** définit la représentation graphique d'un élément. La classe possède une référence vers le contrôleur auquel elle appartient. Celle-ci est nécessaire pour la mise à jour des informations de l'élément après différentes interactions.
5. *serveur* qui contient le serveur RMI auquel les joueurs se connectent :
   - l'interface **IArene** donne la signature d'un serveur. L'interface est utilisée par plusieurs classes pour mettre en œuvre la communication RMI.
   - la classe **Arene** implémente l'interface *IArene*. Cette classe est responsable de déclencher les interactions entre éléments.
6. *utilitaires* qui contient des calculs nécessités par les autres classes :
   - la classe **Calculs** propose des calculs spécifiques à la *Console* (par exemple, distance entre deux vues des éléments, la meilleure case pour se déplacer, etc.).
   - la classe **PointComp** définit les moyens pour comparer deux points graphiques. Chaque vue d'un élément contient un point qui donne sa position sur l'arène.

Dans le diagramme ci-présent, le paquetage d'une classe est indiqué en dessous de son nom. Certains attributs sont modélisés par des *associations* (ligne continue avec flèche). L'implémentation des interfaces est représentée par une relation de *réalisation* (ligne discontinue avec flèche pleine).

La documentation complète se trouve dans la *javadoc* contenue dans le projet.



3

FIGURE 1 – Diagramme de classes complet.

3 Etapes de lancement en images

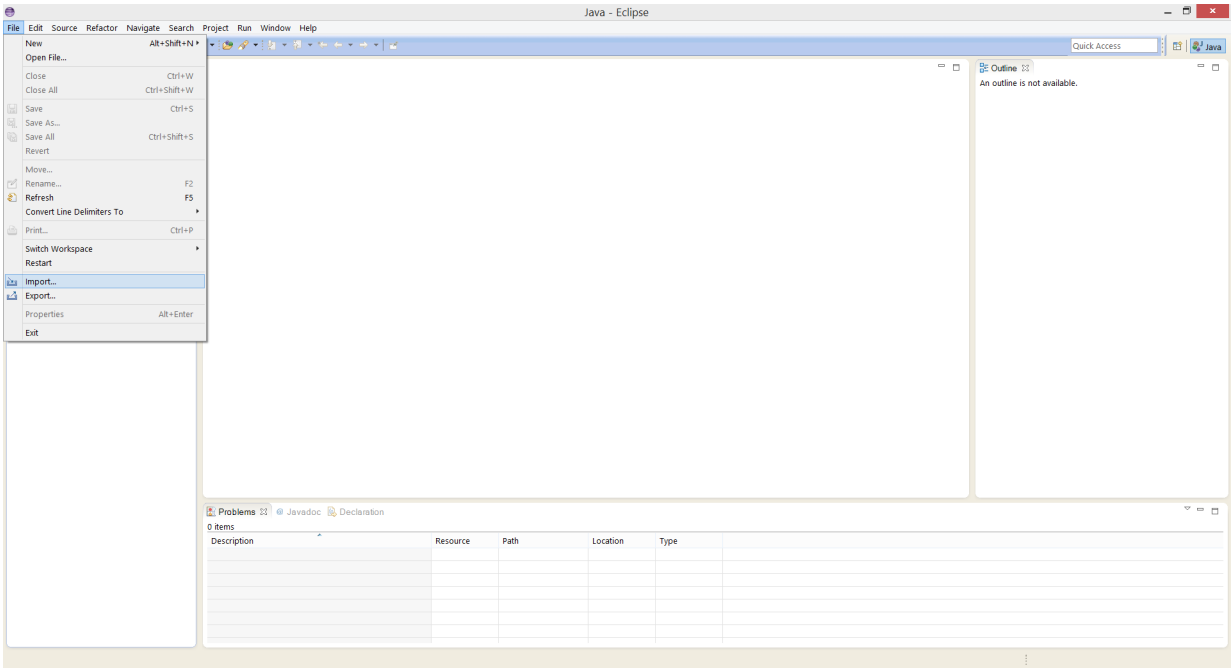


FIGURE 2 – Etape 1.

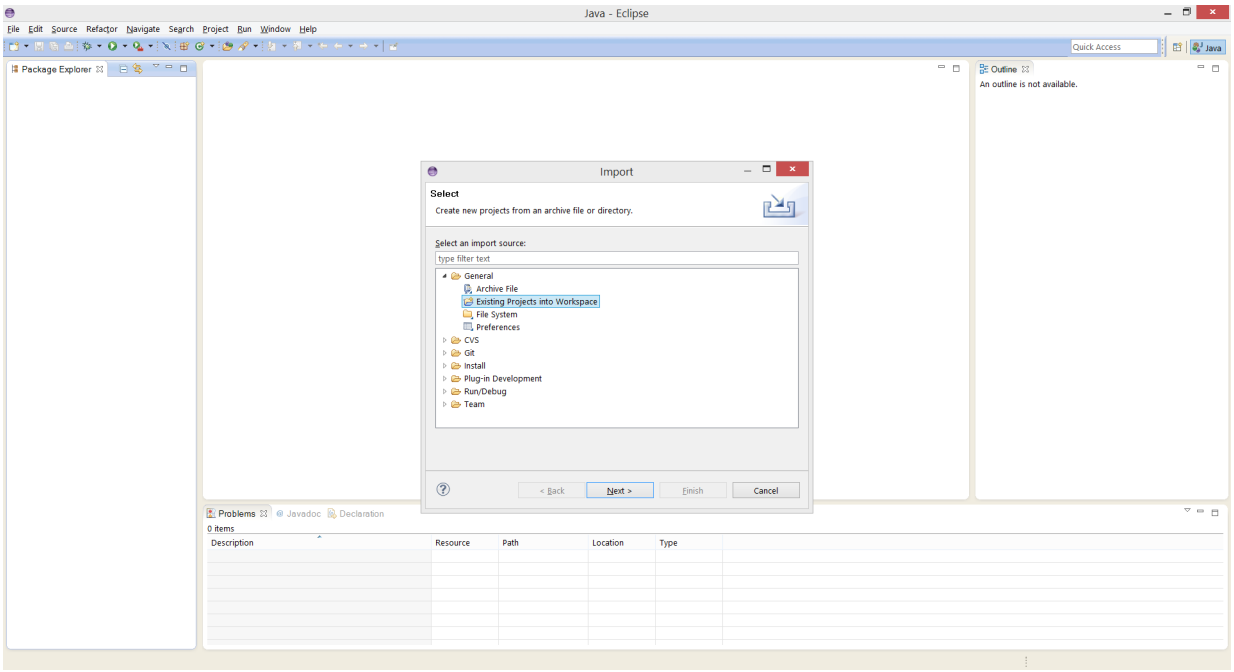


FIGURE 3 – Etape 2.

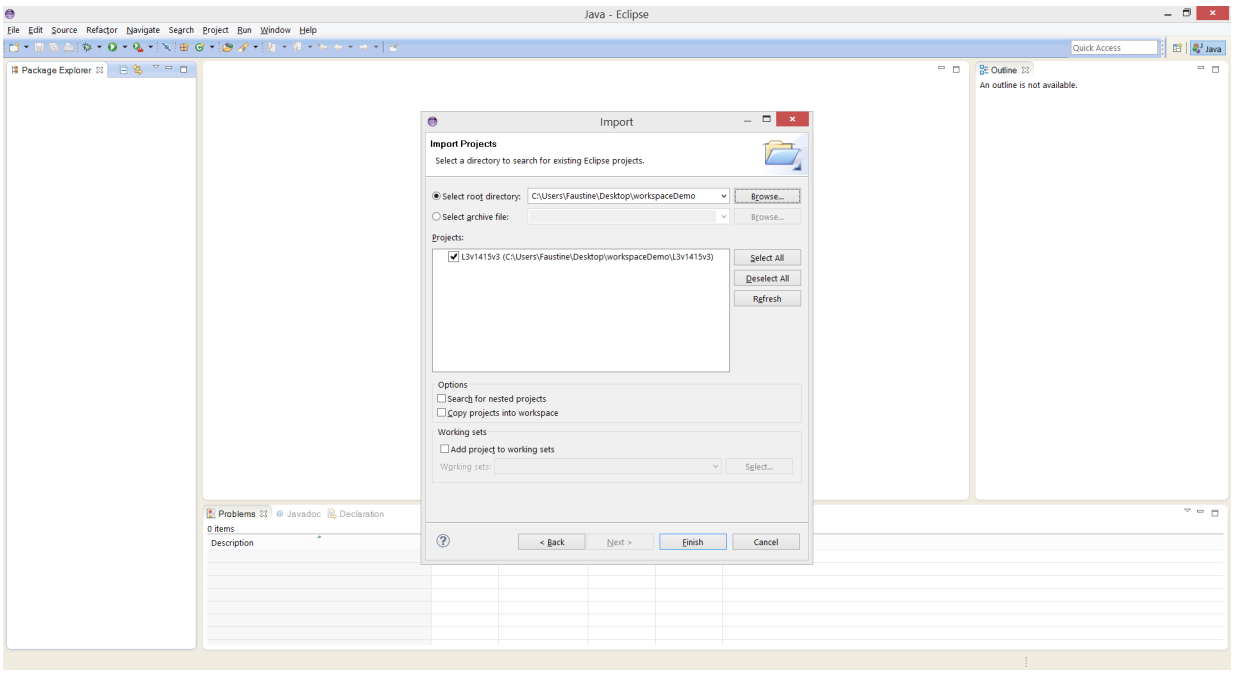


FIGURE 4 – Etape 3.

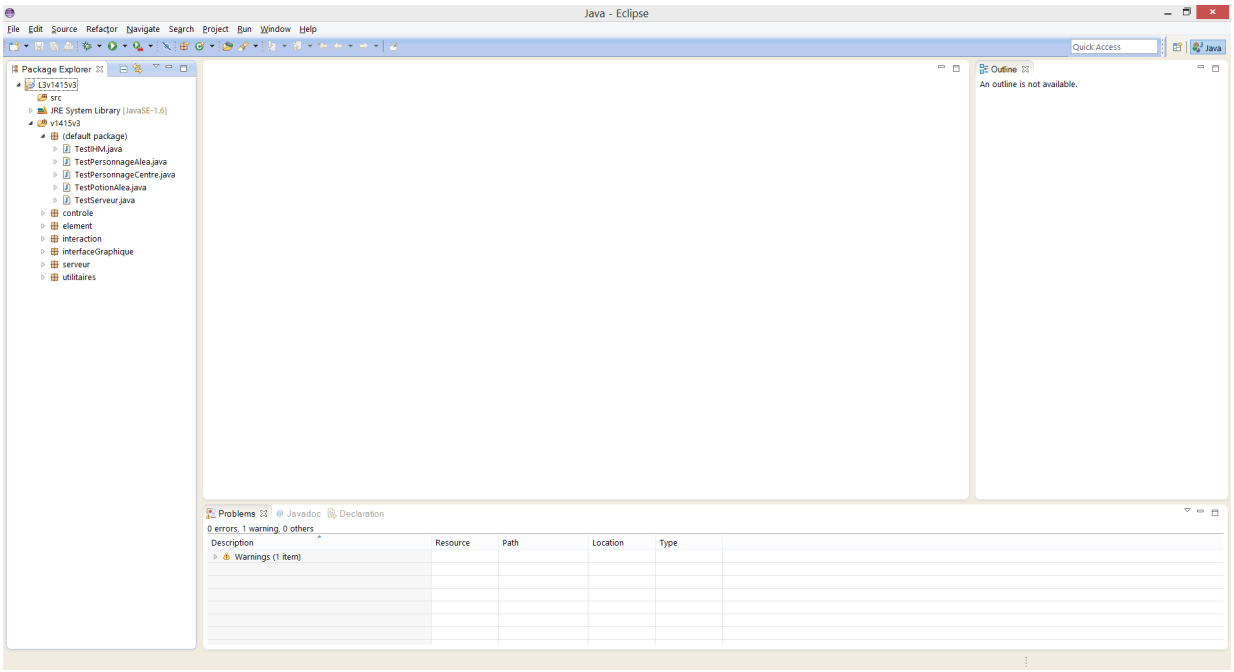


FIGURE 5 – Etape 4.

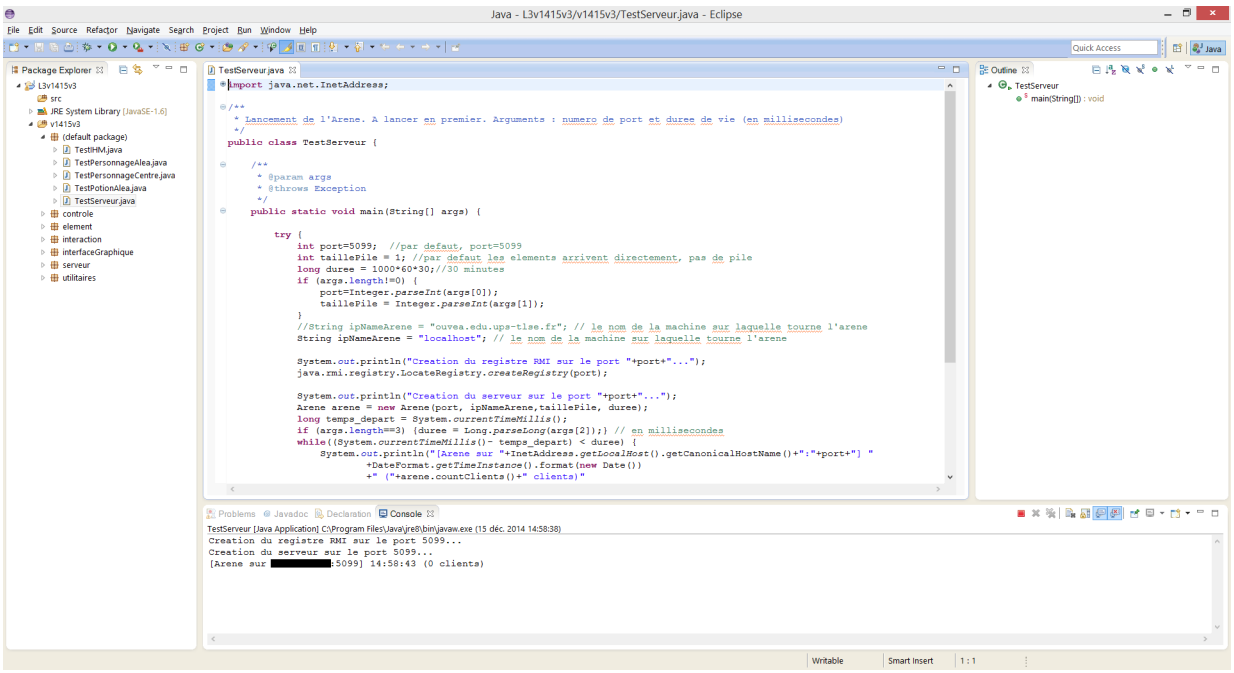


FIGURE 6 – Etape 5.

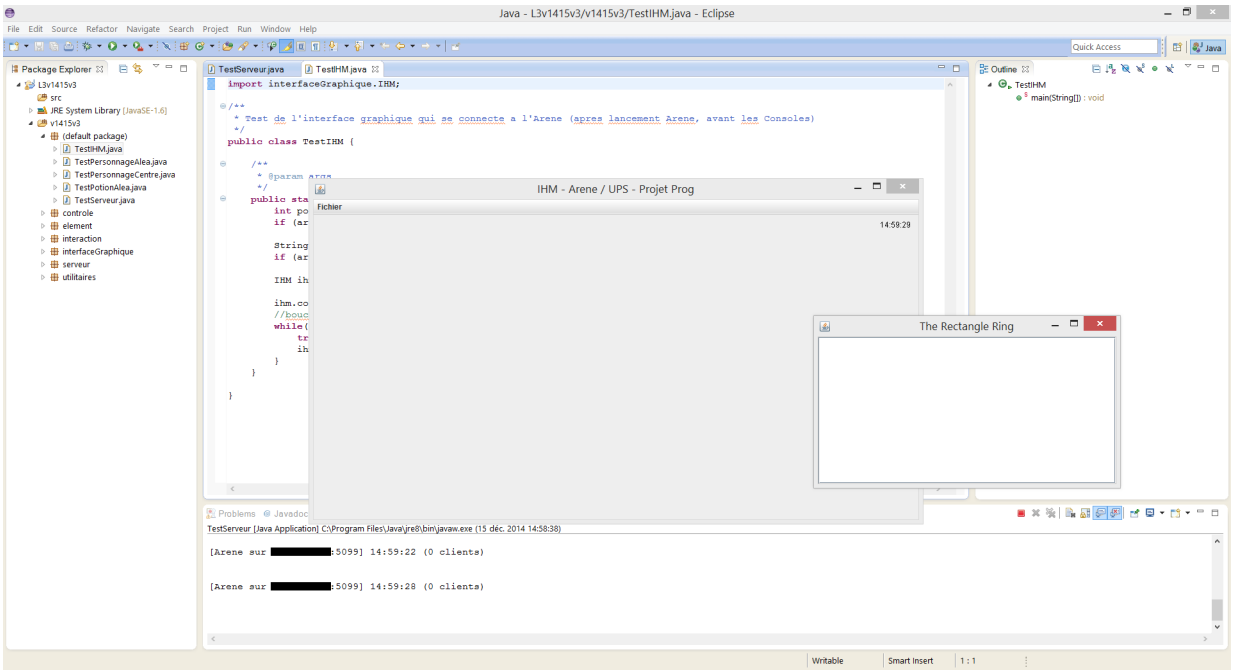


FIGURE 7 – Etape 6.

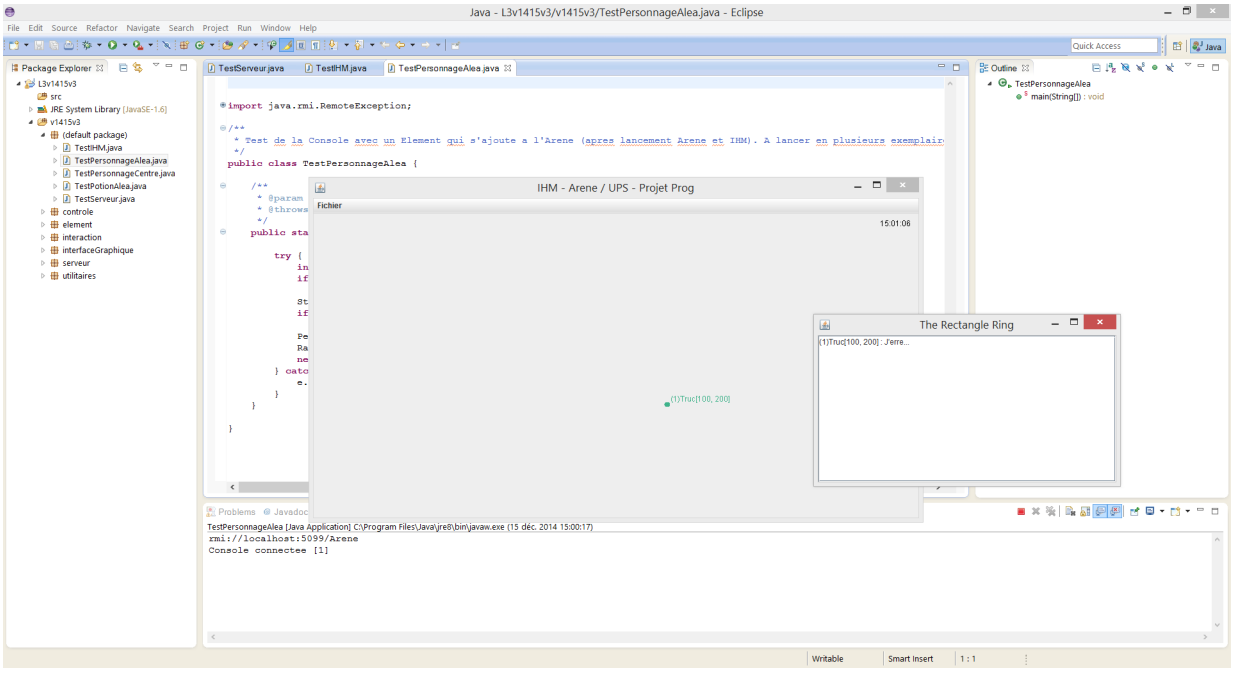


FIGURE 8 – Etape 7.