Modélisation d'un problème : exemple sur les tours de Hanoi

L'exercice 2 du TP4 décrit avec exactitude la forme que doit prendre tout projet rendu et ce qu'on y attend, même si quelques points sont répétés dans ce document.

1/ Définir les classes nécessaires à sa résolution
Le jeu des tours de Hanoi consistant à déplacer des disques entre trois tours. On a envie d'utiliser des objets de types : Disc ; Tower ; Hanoi (le jeu)

2/ Diagramme UML
Définir ce qui caractérise les objets de chacune des classes précédemment définies (attributs), les fonctionnalités que l'on veut leur prêter (méthodes) et comment les initialiser (constructeur(s)). Toujours pas de code, juste comment on ferait fonctionner le tout par principe. Il peut manquer des méthodes, par exemple des méthodes "intermédiaires" qui ne seront pas utiles à l'utilisateur ou rajouter des fonctionnalités auxquelles on n'a pas pensées au départ quand on en sent le besoin, mais essayer d'avoir au-moins les méthodes principales dans le premier diagramme.

3/ Préparer votre environnement de travail
Vous pouvez commencer à définir l'environnement de travail (en suivant ce qui a été fait au TP4) c'est-à-dire un répertoire principal (TP5) contenant :

- les répertoires src, classes, docs, test,
- un fichier manifest pour définir la classe main,
- l'archive jar test-1.7.jar (à récupérer d'autres projets),
- un fichier README, qui doit absolument être rempli (cela fera partie de la notation) et contenir des informations qui me seront utiles pour manipuler votre projet : comment compiler, comment exécuter des tests, quels sont les éventuels problèmes connus dans votre code (irrésolus), comment exécuter vos exécutables… Toute information qui peut m'être utile,
- un makefile (optionnel) mais bien pratique pour éviter de retaper sans arrêt les mêmes commandes du TP4 (vous les avez vu en AP2, c'est une excellente occasion de les réutiliser désormais),

Mettre en œuvre les paquetages dans src et écrire les "squelettes" des classes puis…

4/ Écrire la documentation et les tests unitaires
Attention, la documentation et les tests doivent être écrits avant le code de chaque méthode ! On ne connaît que les signatures des méthodes à utiliser, ce qu'elles devraient faire mais on ne les a pas encore écrites. Si exceptions il y a, elles doivent être signalée dans la doc (@exception) ; on teste à part les exceptions.
Pour rappel, un test unitaire doit "prouver" avec une quasi certitude qu'une méthode A fait ce qu'on veut qu'elle fasse, dans tous les cas qu'elle peut rencontrer. Elle suppose que les autres méthodes utilisées (une méthode B par exemple) sont vraies. Ces méthodes utilisées seront "prouvées" quant à elles dans un autre test. Donc assurez vous que le test de B soit vrai avant de vérifier celui de A.

5/ Écrire les méthodes dans les classes
On peut réfléchir à coder nos méthodes. Quant une méthode est terminée, on vérifie qu'elle passe les tests. Si c'est le cas, on peut la valider, et écrire les méthodes suivantes.

6/ Rendre le travail
Cf exercice 2 du TP4.