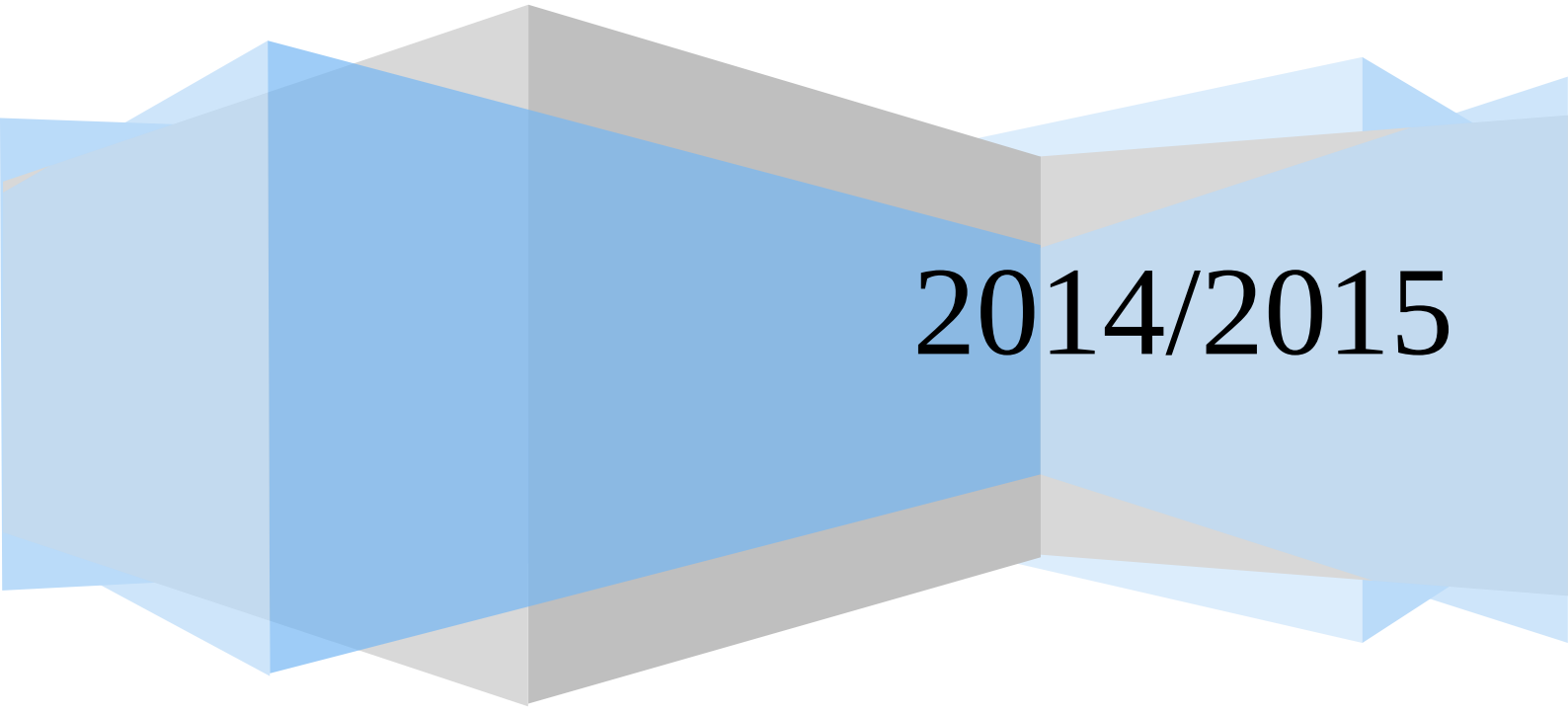
Université de Béjaia – Abderrahmane Mira

TEST de TP – Programmation Avancée

Tours de Hanoï (Version récursive et Version Dérécursivée)

Mounir Hamani



2014/2015

Objectif :

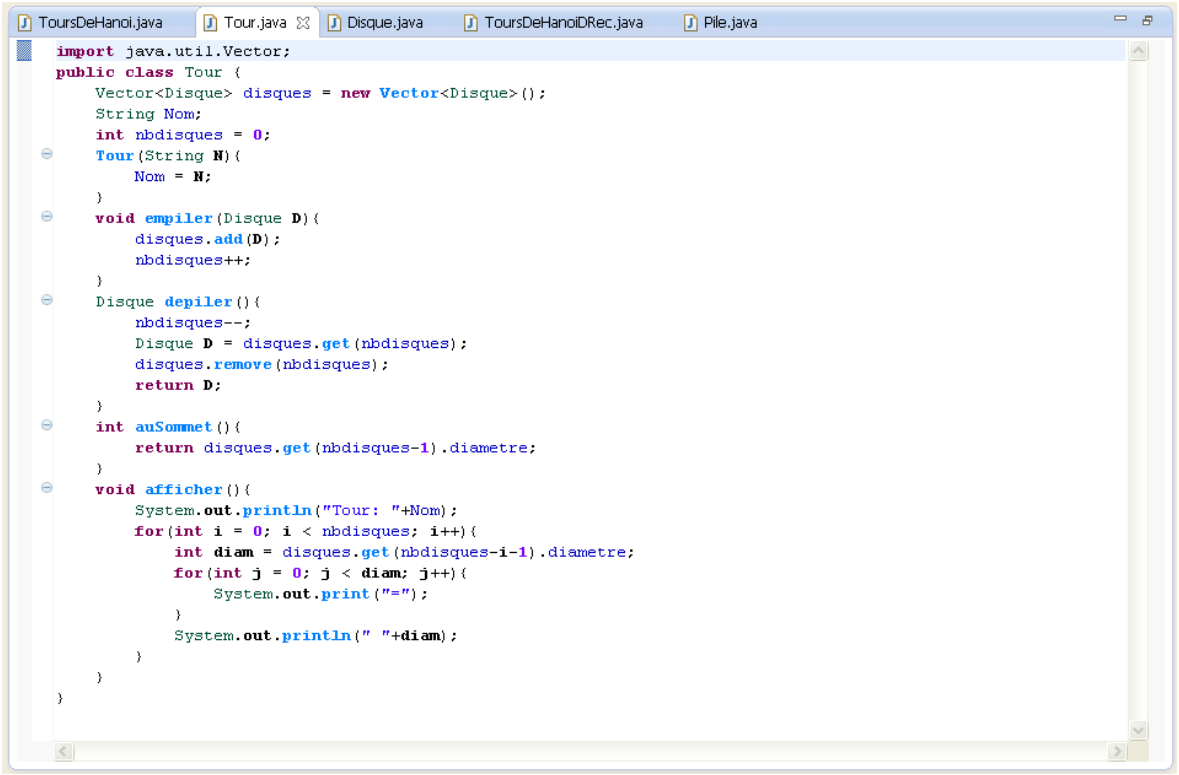
Dérécursiver l'algorithme de résolution des Tours de Hanoï.

La métrique de performance à calculer est la : Complexité.

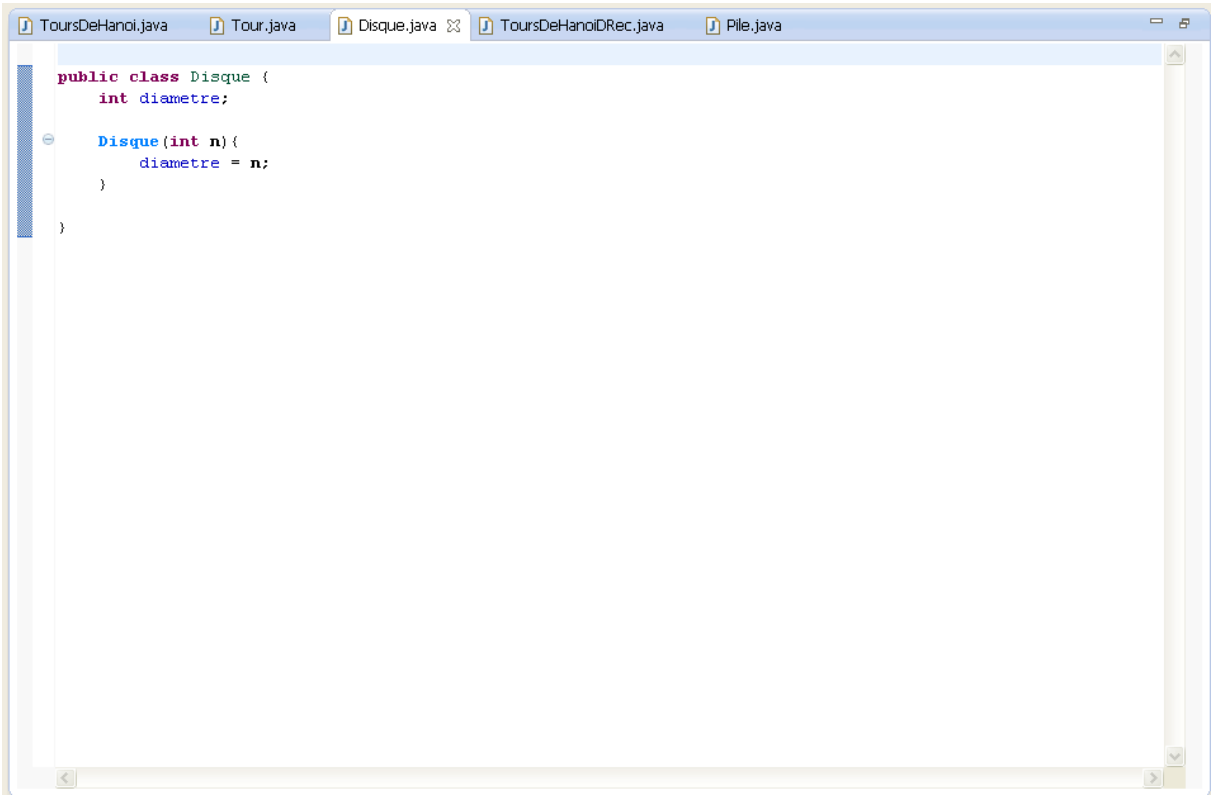
La complexité de l'algorithme de résolution des Tours de Hanoï possède une complexité de (2^n-1) déplacements. Dans ce problème, il n'existe pas de meilleur ou pire des cas, cette complexité dépend exclusivement du nombre de disques considérés. Aussi, la version dérécursivée possède la même complexité (2^n-1) que la version récursive.

Implémentation :

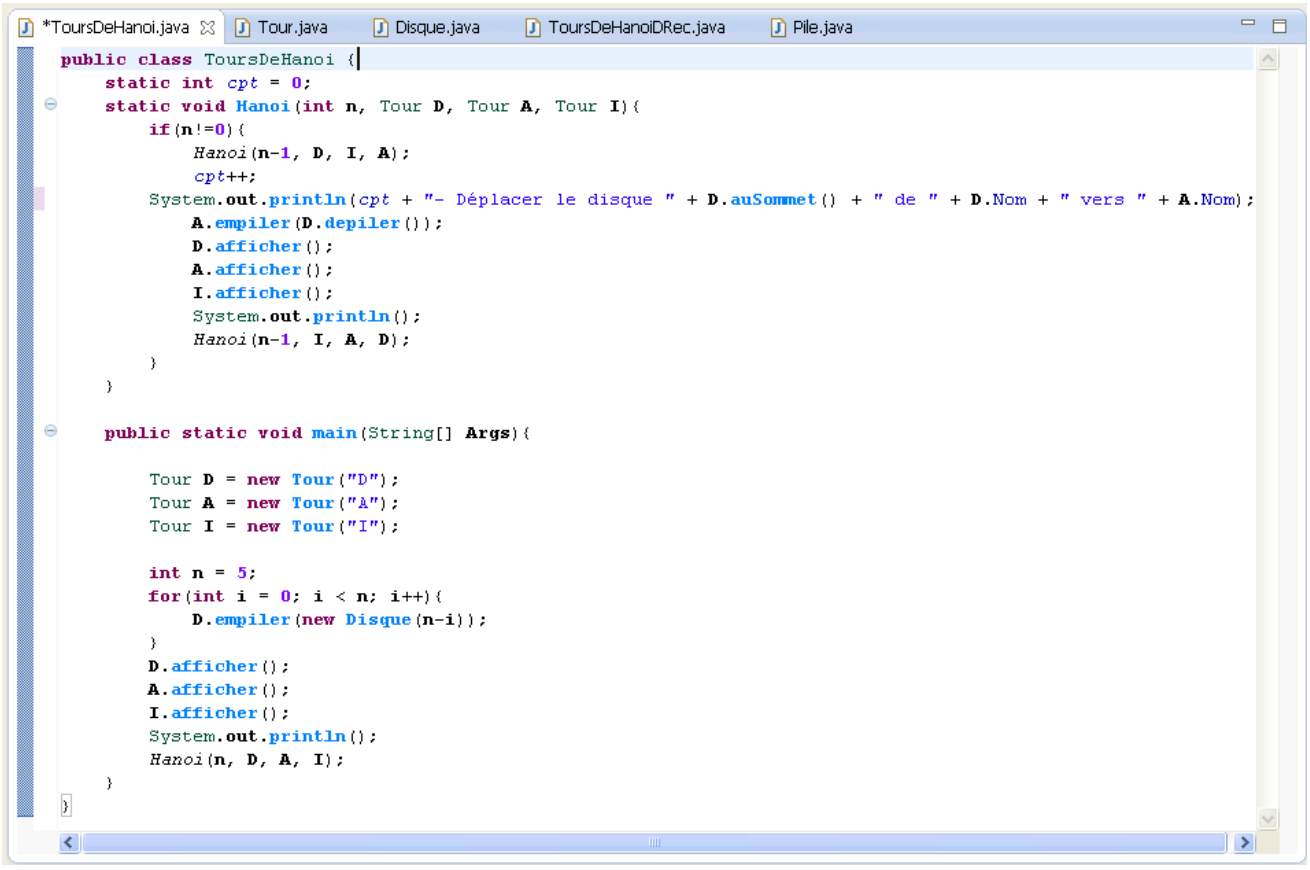
Pour implémenter la solution, j'ai choisi l'approche Orientée Objet, j'ai donc défini les classes Tour, Disque ainsi que ToursDeHanoi qui contiendra le programme principal de la version récursive.



Prise d'écran de la classe Tour



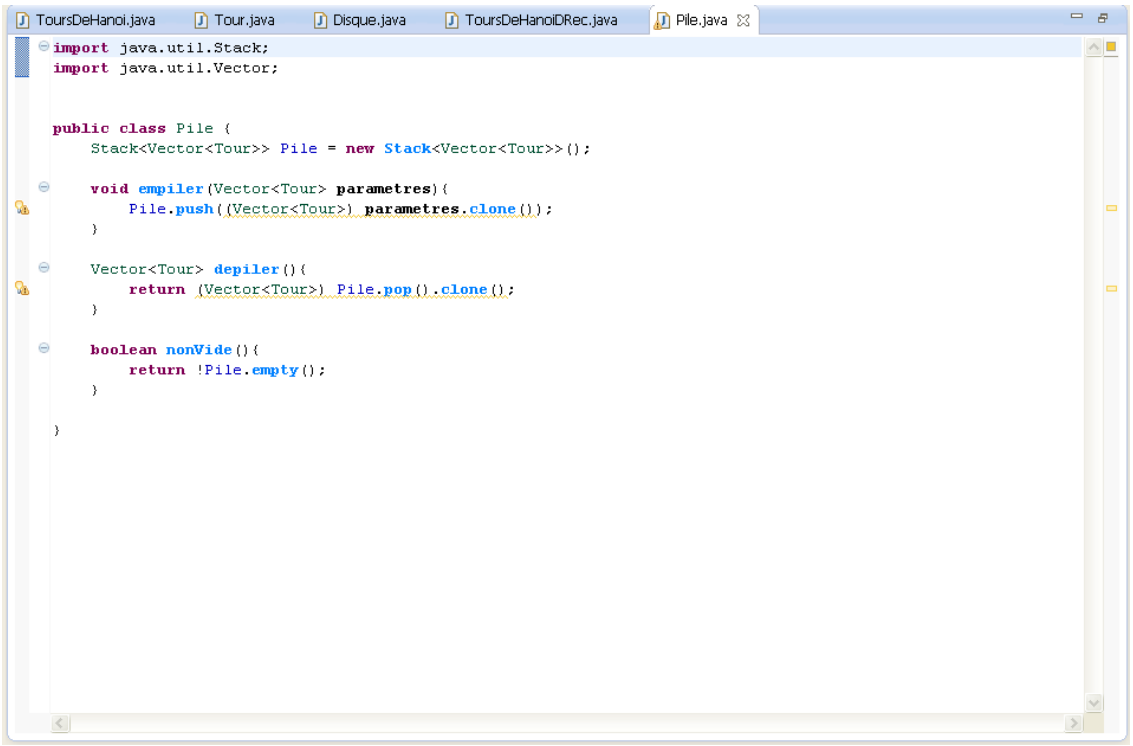
Prise d'écran de la classe Disque



Prise d'écran de la classe ToursDeHanoi (version récursive)

Dérécursivation :

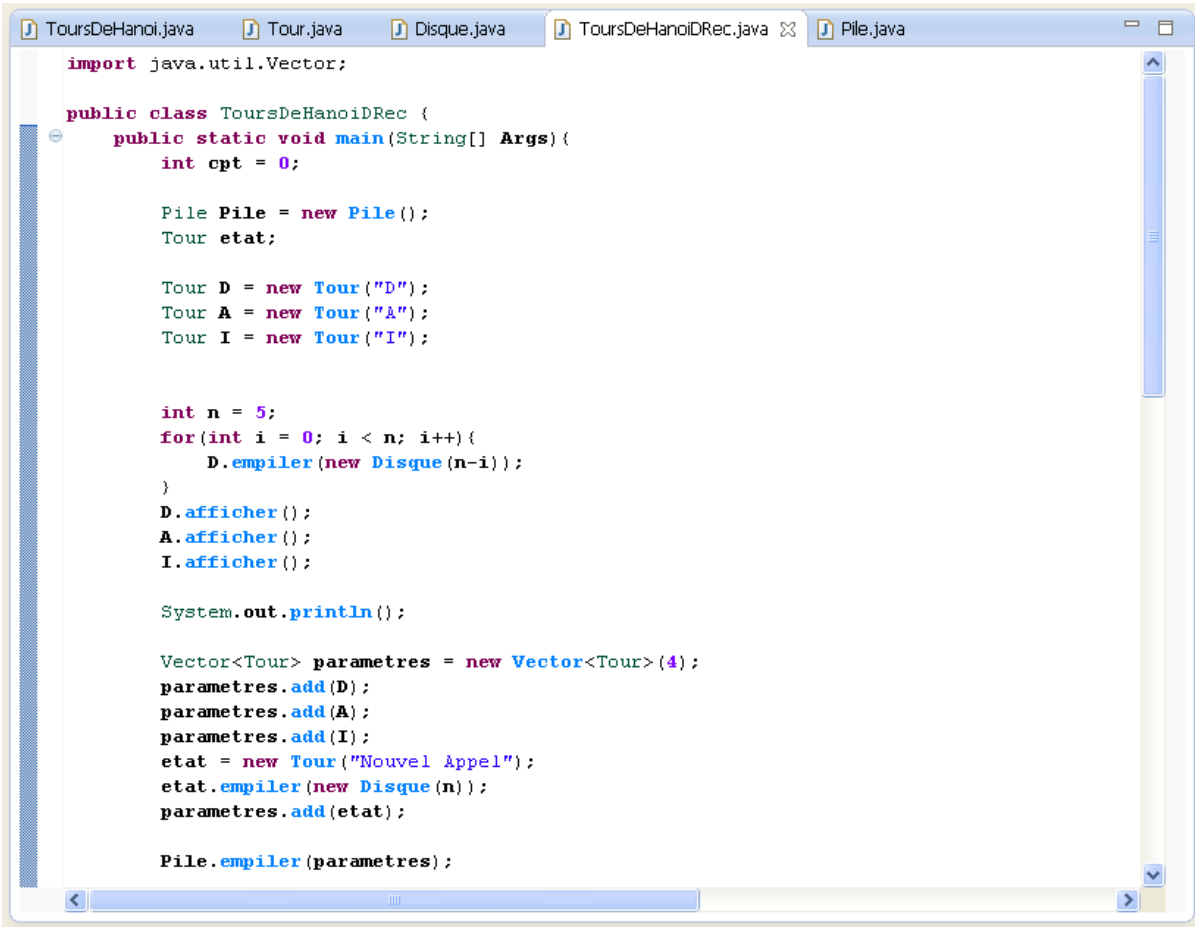
Concernant la version dérécursivée, j'ai réutilisé les classes « Tour » et « Disque », auxquelles j'ai ajouté la classé « Pile » qui permettra d'implémenter la pile d'appels.



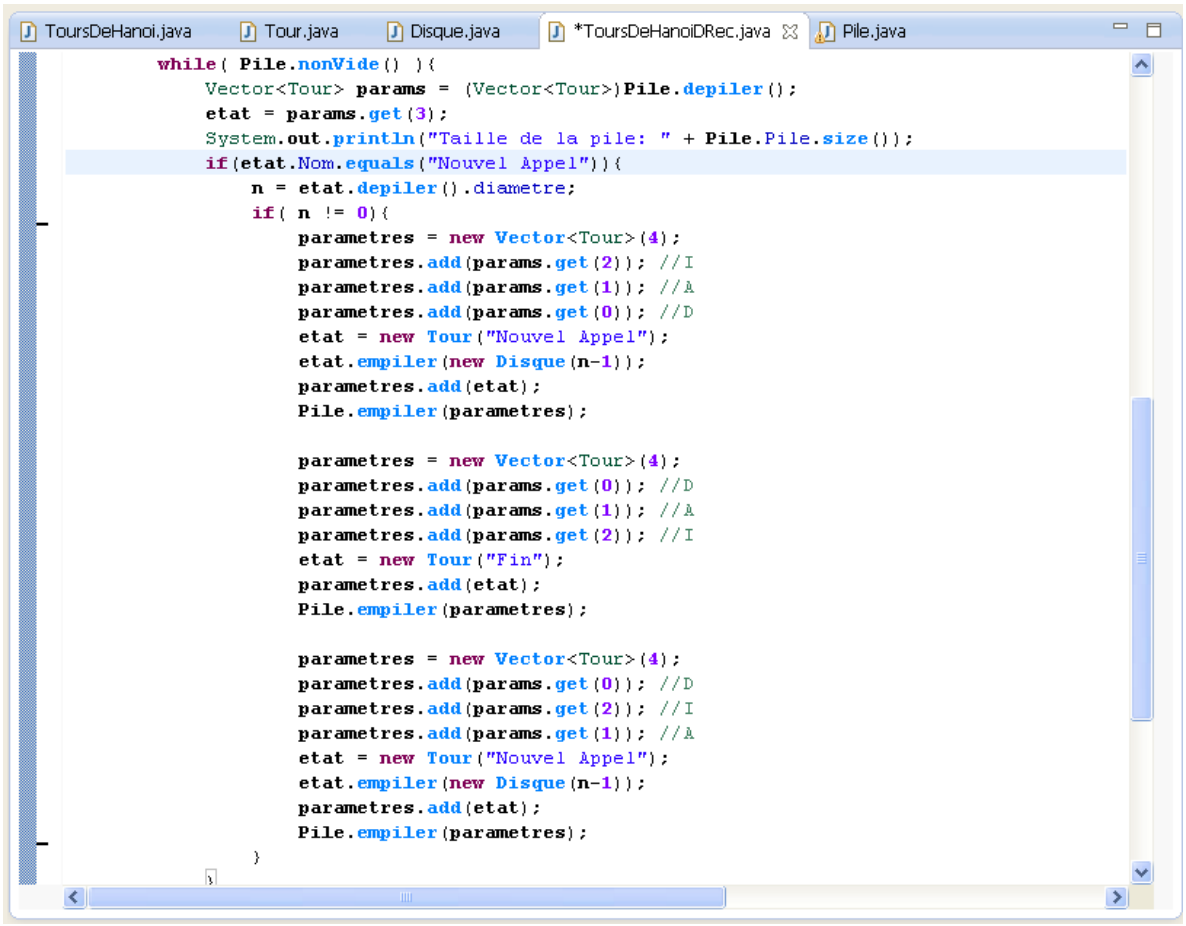
Prise d'écran de la classe Pile

Dans la figure suivante, vous pouvez voir la partie « initialisation » de l'algorithme (dérécursivé).

Vous remarquerez que pour implémenter l'état, j'utilise une instance de « Tour », dans la quelle je mets un « Disque » dont le « diamètre » représentera la valeur du paramètre « n ». Selon les cas, cet « etat » prendra pour valeur de son champ « Nom » ou bien « Nouvel Appel » ou bien « Fin », ce qui nous permettra par la suite d'effectuer un test afin de savoir s'il faut exécuter la partie créant de nouveaux appels (après un test sur la valeur de « n ») ou celle exécutant ce qui s'apparente à « F(U) » (les déplacements).



Prise d'écran de la première partie de ToursDeHanoiDRec (version dérécursivée)



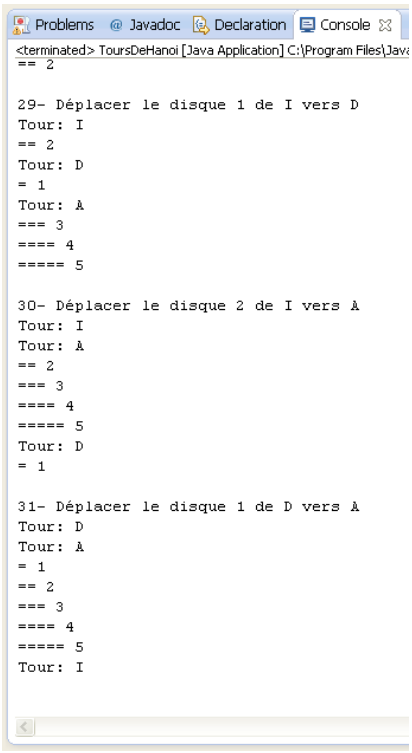
Ci-dessus, prise d'écran de la partie contenant le début de la boucle (dépiler, affectation de l'état stocké à l'état courant et test sur la valeur de l'état).

Ci-dessus, la prise d'écran de la partie contenant le début de la boucle, autrement dit, les opérations « dépiler » et affectation de l'état déstocké à l'état courant. La valeur de cet état est ensuite testée afin de déterminer s'il s'agit d'un nouvel appel ou s'il faut passer à la partie suivante et exécuter l'équivalent de « F(U) », les déplacements et affichages.

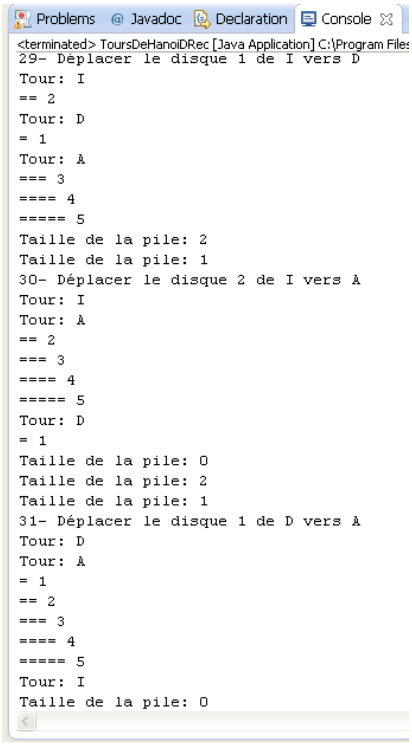
Ci-dessus, capture d'écran de la dernière partie de la boucle « while » et du programme dérécursivé.

La partie que vous pouvez observer dans le bloc « else » ci-dessus, est exécutée si la valeur de l'état est différente de « Nouvel Appel » (c'est-à-dire que l'état est égale à « Fin »).

Exemple de déroulement :



Déroulement récursif avec 5 Disques



Déroulement dérécursivé avec 5 Disques

Comme vous pouvez le voir ci-dessus, les deux déroulements réalisent le même nombre de déplacements avec 5 Disques, en l'occurrence 31 = (2^5-1). Les deux versions produisent toujours les mêmes résultats même en testant avec d'autres valeurs de « n » (nombre de Disques).

Vous remarquerez également que dans la version dérécursivée, j'ai choisi d'afficher l'état (de remplissage) de la pile, cela afin de vérifier et valider qu'elle est bien vide à la fin du déroulement.

P.S. : Vous pouvez récupérer le code source sur le site www.Dirassatic.com afin de le tester et vérifier les résultats de l'exécution.