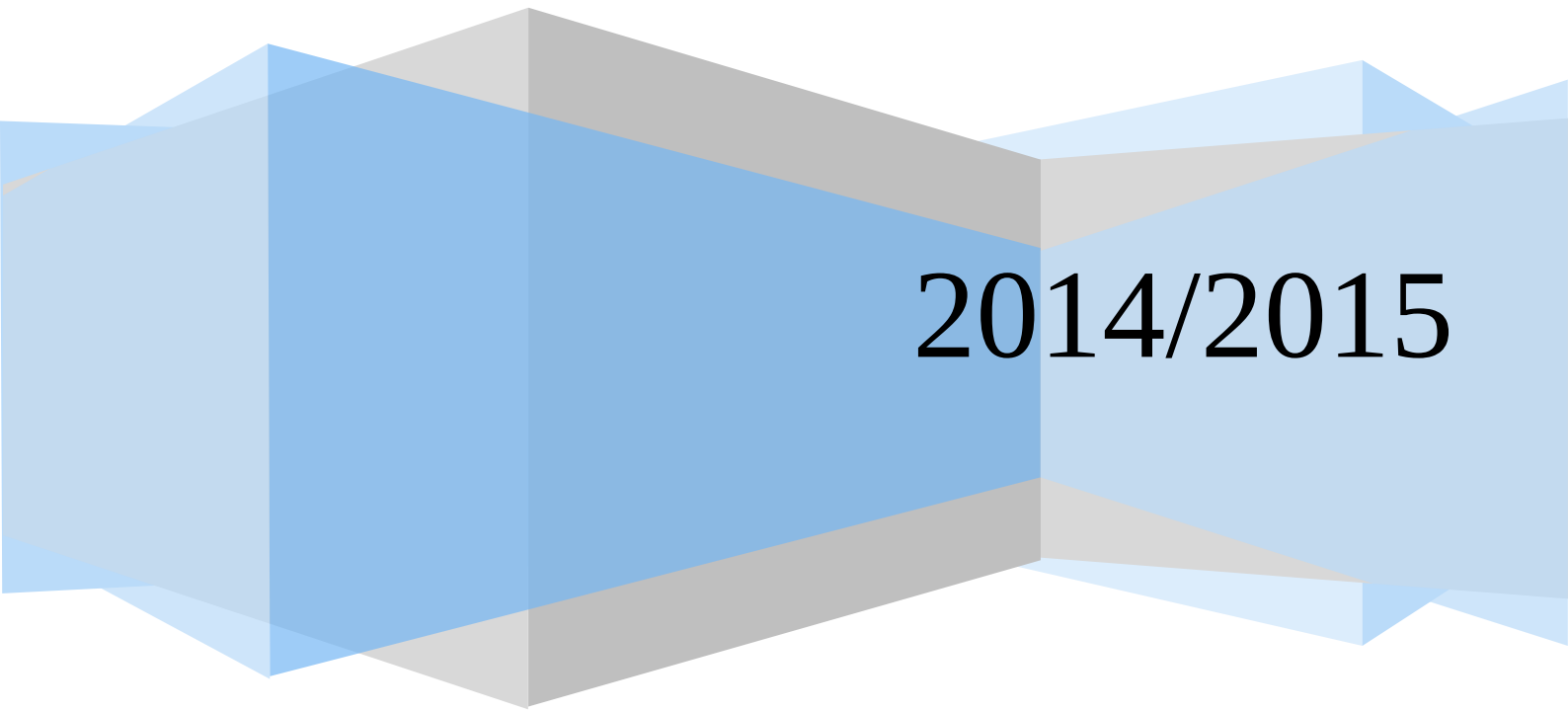


Université de Béjaia – Abderrahmane Mira

TEST de TP – Programmation Avancée

Tours de Hanoï (Version récursive et Version
Dérécursivée)

Mounir Hamani



2014/2015

Objectif :

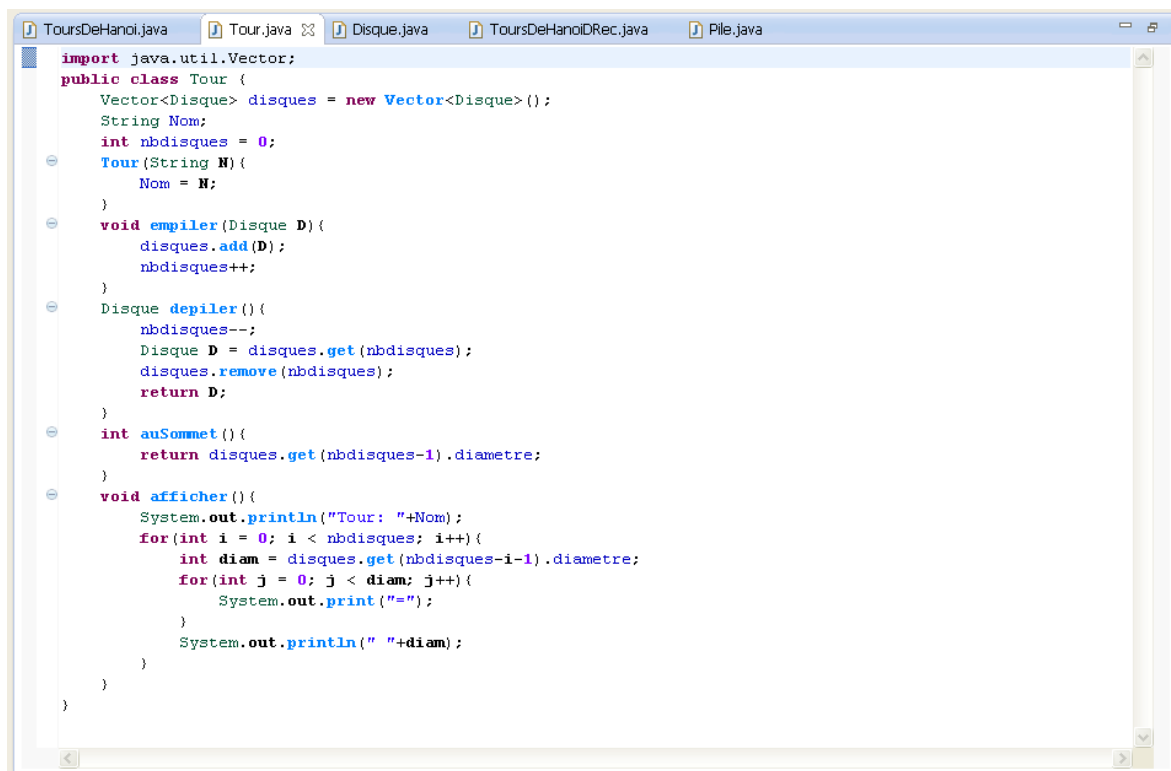
Dérécursiver l'algorithme de résolution des Tours de Hanoï.

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 dérécurivée possède la même complexité $(2^n - 1)$ que la version récursive.

Implémentation :

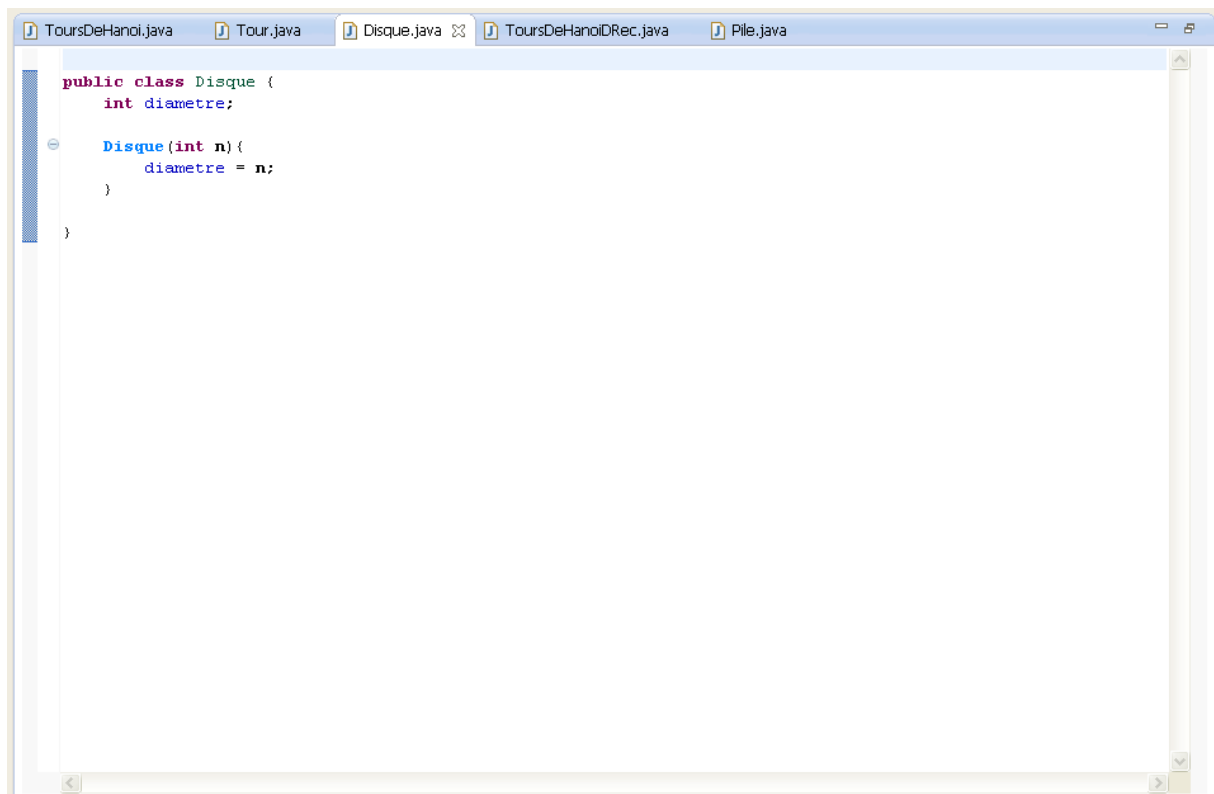
Pour implémenter la solution, j'ai choisi l'approche Orientée Objet, j'ai donc défini les classes Tour, Disque ainsi que ToursDeHanoi qui contiendra le programme principal de la version récursive.



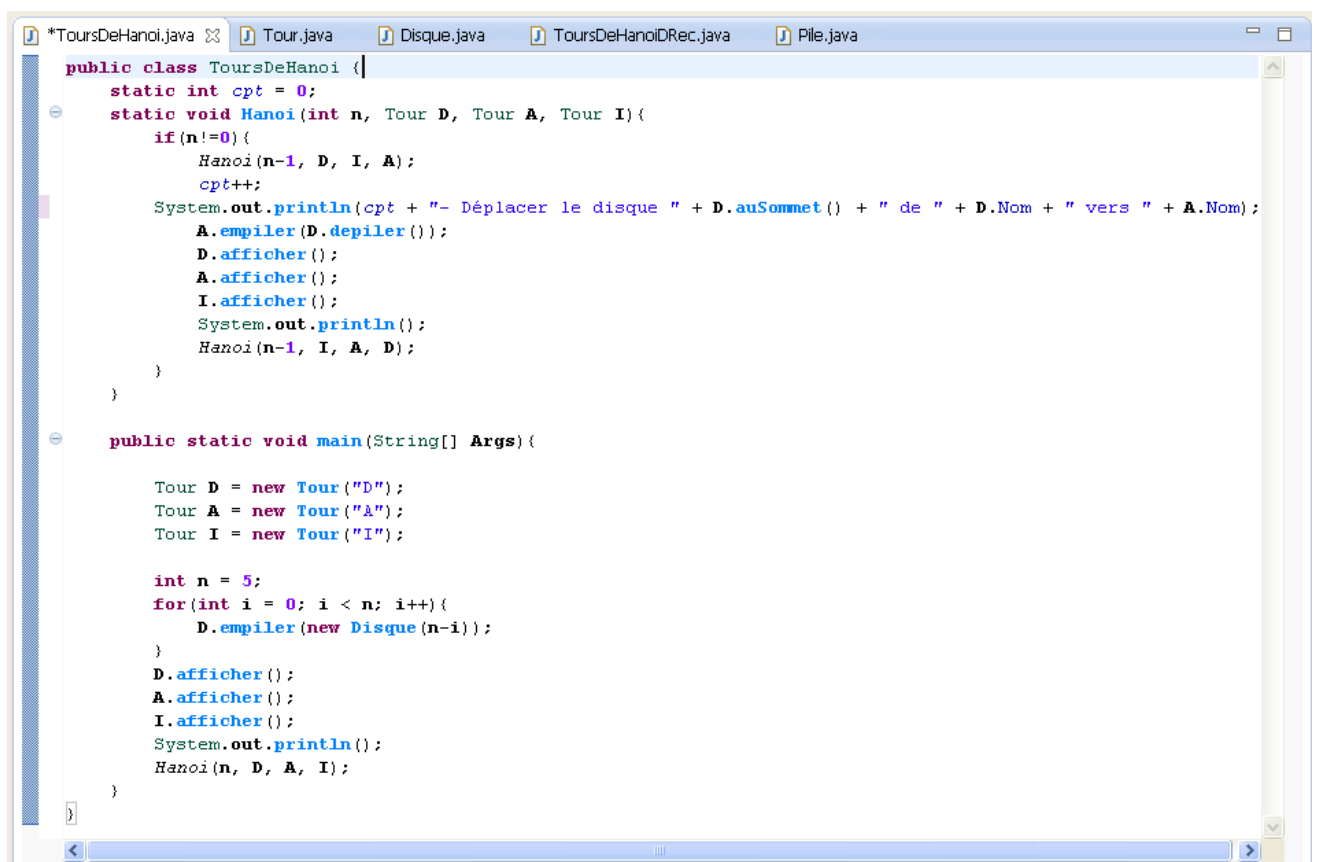
```
import java.util.Vector;

public class Tour {
    Vector<Disque> disques = new Vector<Disque>();
    String Nom;
    int nbdisques = 0;
    Tour(String N) {
        Nom = N;
    }
    void empiler(Disque D) {
        disques.add(D);
        nbdisques++;
    }
    Disque depiler() {
        nbdisques--;
        Disque D = disques.get(nbdisques);
        disques.remove(nbdisques);
        return D;
    }
    int auSommet() {
        return disques.get(nbdisques-1).diametre;
    }
    void afficher() {
        System.out.println("Tour: " + Nom);
        for (int i = 0; i < nbdisques; i++) {
            int diam = disques.get(nbdisques-i-1).diametre;
            for (int j = 0; j < diam; j++) {
                System.out.print("=");
            }
            System.out.println(" " + diam);
        }
    }
}
```

Prise d'écran de la classe Tour



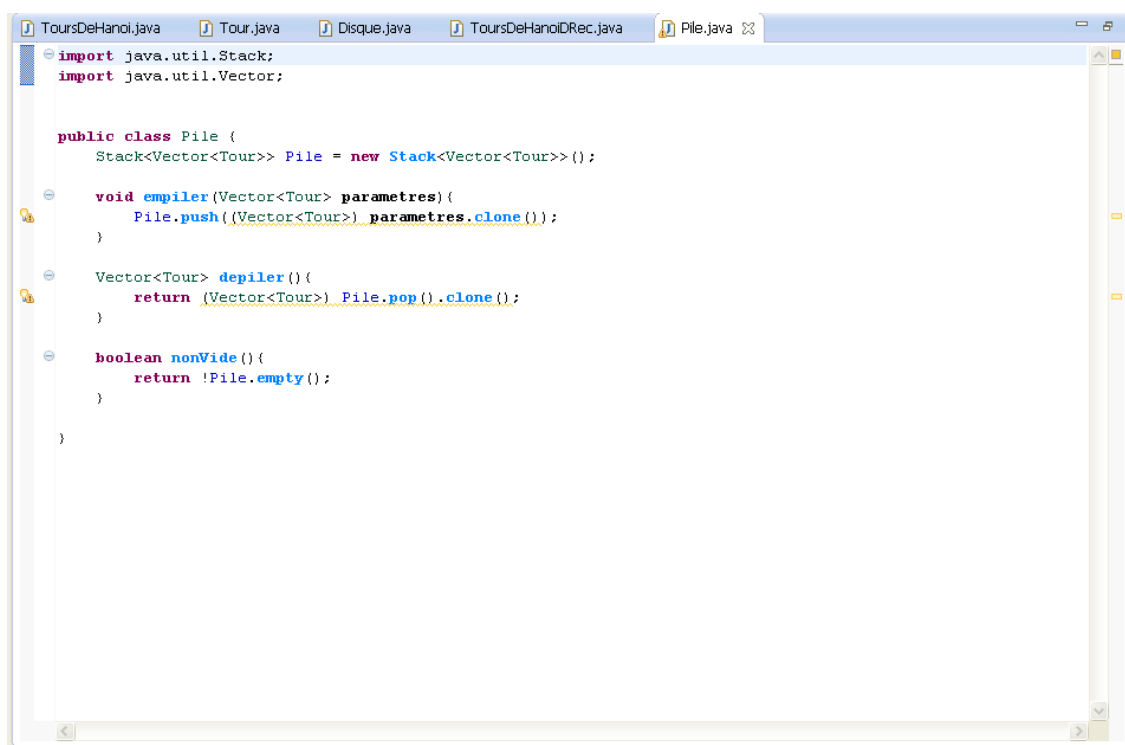
Prise d'écran de la classe Disque



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

Dérécursivation :

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

A screenshot of a Java IDE window with several tabs open: ToursDeHanoi.java, Tour.java, Disque.java, ToursDeHanoiDRec.java, and Pile.java. The Pile.java tab is active, showing the following code:

```
import java.util.Stack;
import java.util.Vector;

public class Pile {
    Stack<Vector<Tour>> Pile = new Stack<Vector<Tour>> ();

    void empiler(Vector<Tour> parametres) {
        Pile.push((Vector<Tour>) parametres.clone());
    }

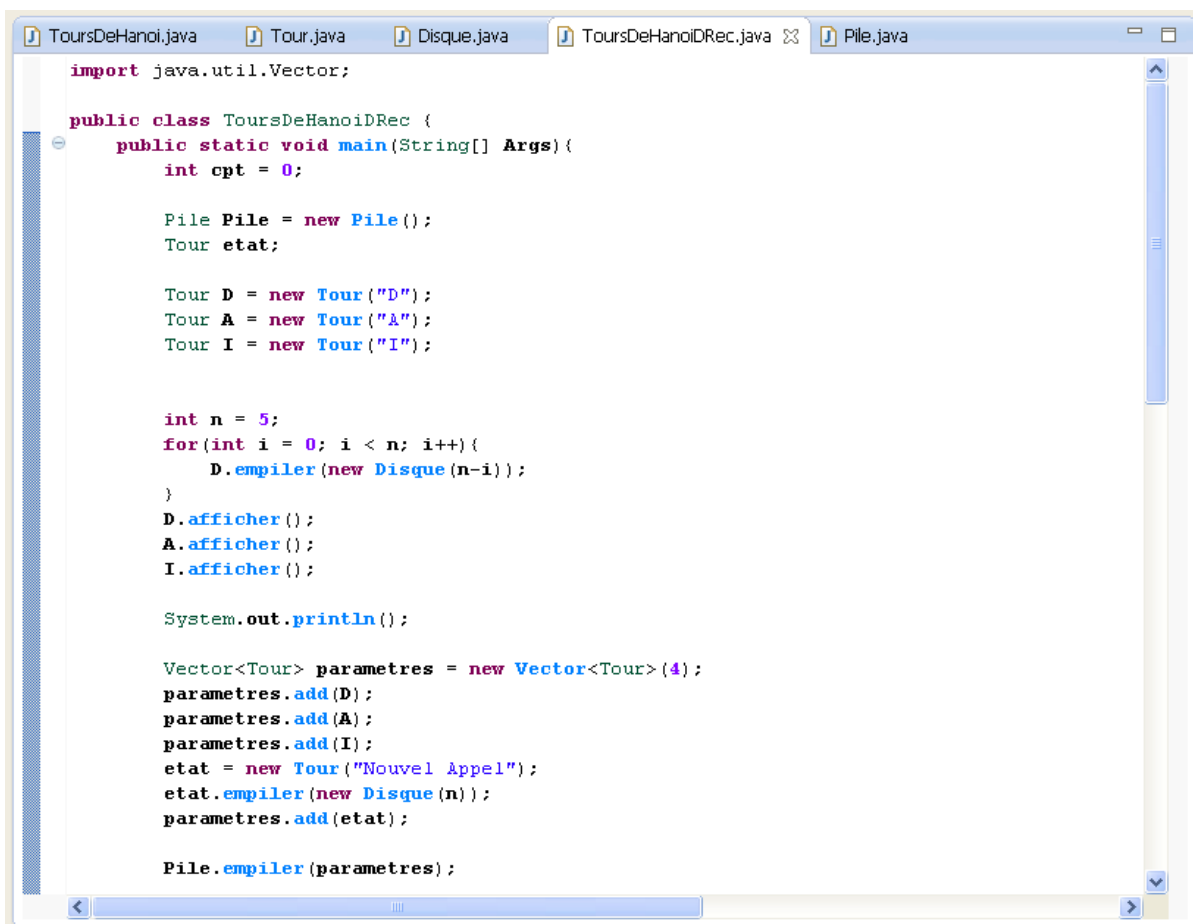
    Vector<Tour> depiler() {
        return (Vector<Tour>) Pile.pop().clone();
    }

    boolean nonVide() {
        return !Pile.empty();
    }
}
```

Prise d'écran de la classe Pile

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

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 « état » 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).

A screenshot of a Java IDE window showing the code for the `ToursDeHanoiDRec` class. The code is in French and shows the initialization of variables and the start of a loop. The code is as follows:

```
import java.util.Vector;

public class ToursDeHanoiDRec {
    public static void main(String[] Args) {
        int cpt = 0;

        Pile Pile = new Pile();
        Tour etat;

        Tour D = new Tour("D");
        Tour A = new Tour("A");
        Tour I = new Tour("I");

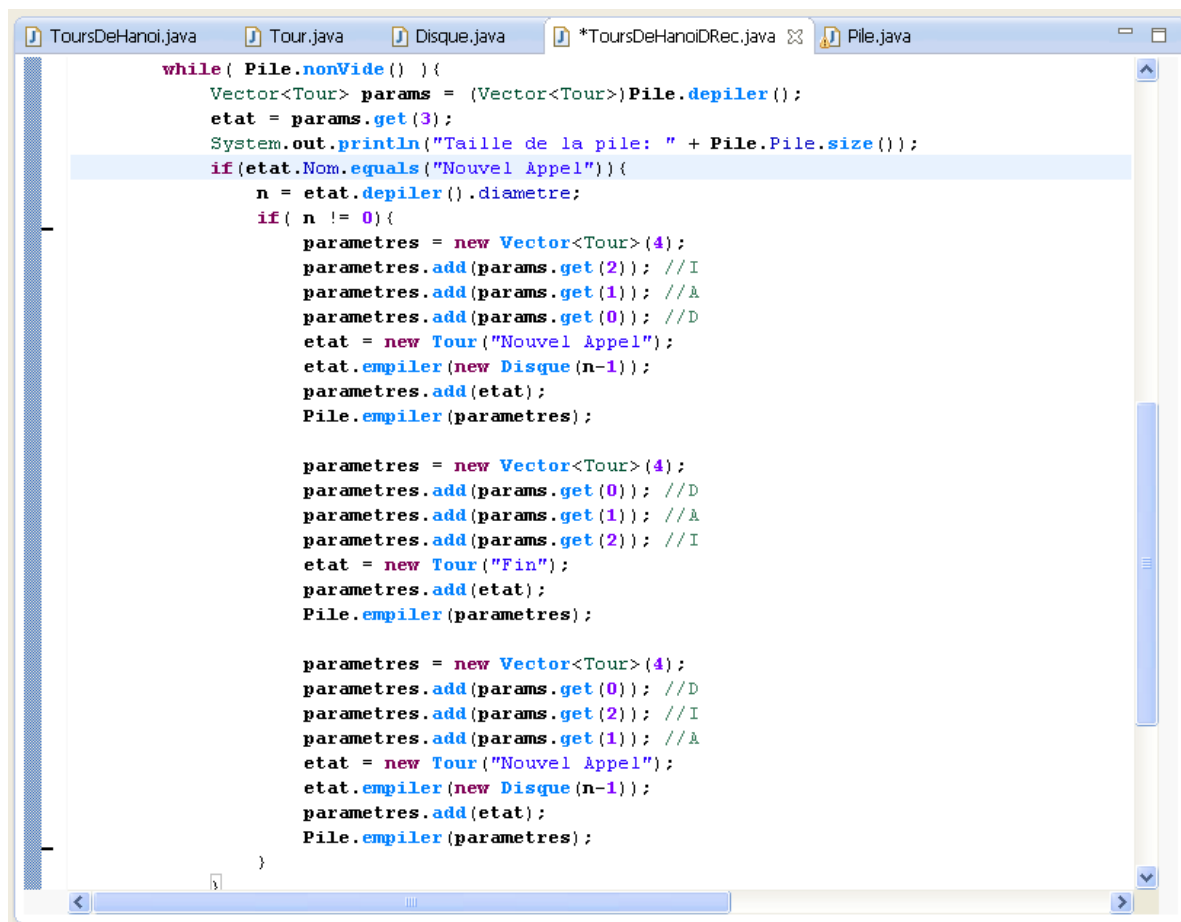
        int n = 5;
        for(int i = 0; i < n; i++){
            D.empiler(new Disque(n-i));
        }
        D.afficher();
        A.afficher();
        I.afficher();

        System.out.println();

        Vector<Tour> parametres = new Vector<Tour>(4);
        parametres.add(D);
        parametres.add(A);
        parametres.add(I);
        etat = new Tour("Nouvel Appel");
        etat.empiler(new Disque(n));
        parametres.add(etat);

        Pile.empiler(parametres);
    }
}
```

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



```
while( Pile.nonVide() ) {  
    Vector<Tour> params = (Vector<Tour>)Pile.depiler();  
    etat = params.get(3);  
    System.out.println("Taille de la pile: " + Pile.Pile.size());  
    if(etat.Nom.equals("Nouvel Appel")){  
        n = etat.depiler().diametre;  
        if( n != 0){  
            parametres = new Vector<Tour>(4);  
            parametres.add(params.get(2)); //I  
            parametres.add(params.get(1)); //A  
            parametres.add(params.get(0)); //D  
            etat = new Tour("Nouvel Appel");  
            etat.empiler(new Disque(n-1));  
            parametres.add(etat);  
            Pile.empiler(parametres);  
  
            parametres = new Vector<Tour>(4);  
            parametres.add(params.get(0)); //D  
            parametres.add(params.get(1)); //A  
            parametres.add(params.get(2)); //I  
            etat = new Tour("Fin");  
            parametres.add(etat);  
            Pile.empiler(parametres);  
  
            parametres = new Vector<Tour>(4);  
            parametres.add(params.get(0)); //D  
            parametres.add(params.get(2)); //I  
            parametres.add(params.get(1)); //A  
            etat = new Tour("Nouvel Appel");  
            etat.empiler(new Disque(n-1));  
            parametres.add(etat);  
            Pile.empiler(parametres);  
        }  
    }  
}
```

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.



```
file.empiler(parameters);

parameters = new Vector<Tour>(4);
parameters.add(params.get(0)); //D
parameters.add(params.get(1)); //A
parameters.add(params.get(2)); //I
etat = new Tour("Fin");
parameters.add(etat);
Pile.empiler(parameters);

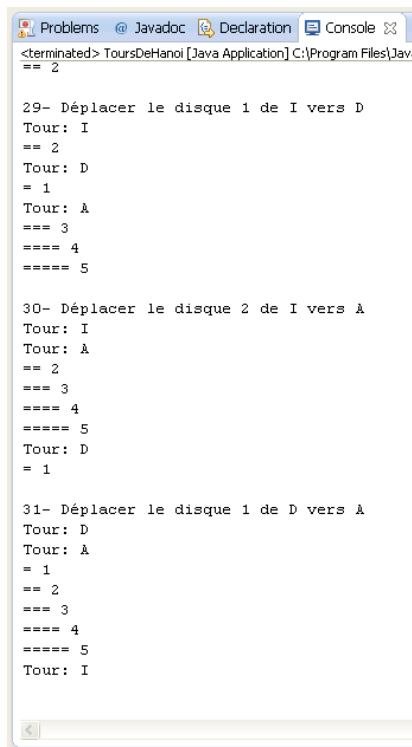
parameters = new Vector<Tour>(4);
parameters.add(params.get(0)); //D
parameters.add(params.get(2)); //I
parameters.add(params.get(1)); //A
etat = new Tour("Nouvel Appel");
etat.empiler(new Disque(n-1));
parameters.add(etat);
Pile.empiler(parameters);
}
}
else{
cpt++;
System.out.println(cpt + "- Déplacer le disque " + params.get(0).auSommet()
params.get(1).empiler(params.get(0).depiler());

params.get(0).afficher();
params.get(1).afficher();
params.get(2).afficher();
}
}
}
```

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

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 :



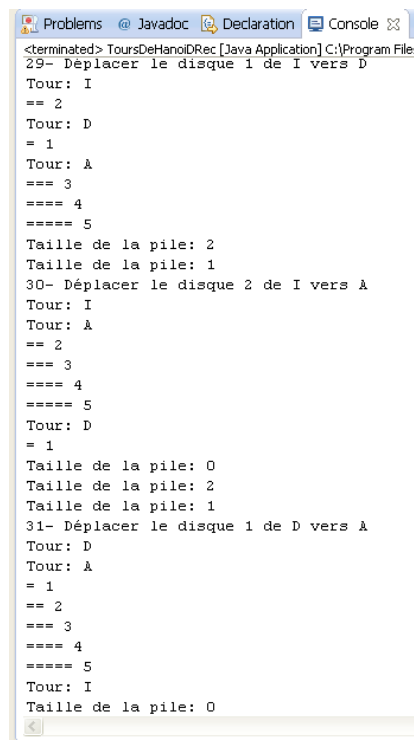
```
Problems @ Javadoc Declaration Console
<terminated> ToursDeHanoi [Java Application] C:\Program Files\Java\
== 2

29- Déplacer le disque 1 de I vers D
Tour: I
== 2
Tour: D
= 1
Tour: A
=== 3
==== 4
===== 5

30- Déplacer le disque 2 de I vers A
Tour: I
Tour: A
== 2
=== 3
==== 4
===== 5
Tour: D
= 1

31- Déplacer le disque 1 de D vers A
Tour: D
Tour: A
= 1
== 2
=== 3
==== 4
===== 5
Tour: I
```

***Déroulement récursif
avec 5 Disques***



```
Problems @ Javadoc Declaration Console
<terminated> ToursDeHanoiDRec [Java Application] C:\Program Files\
29- Déplacer le disque 1 de I vers D
Tour: I
== 2
Tour: D
= 1
Tour: A
=== 3
==== 4
===== 5
Taille de la pile: 2
Taille de la pile: 1
30- Déplacer le disque 2 de I vers A
Tour: I
Tour: A
== 2
=== 3
==== 4
===== 5
Tour: D
= 1
Taille de la pile: 0
Taille de la pile: 2
Taille de la pile: 1
31- Déplacer le disque 1 de D vers A
Tour: D
Tour: A
= 1
== 2
=== 3
==== 4
===== 5
Tour: I
Taille de la pile: 0
```

***Déroulement itératif
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 itérative, 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.