TD n°2 - Correction

Piles, Tri et Tours de Hanoi

Exercice 1 [Piles] Écrire une classe implantant une pile d'éléments.

1. Comment représenter la pile vide ?
2. Définir la classe `Pile`. Le constructeur de cette classe construira la pile vide.
3. Définir une méthode permettant de tester si une pile est vide.
4. Définir la méthode `empile`(ajoute un élément au sommet de la pile)
5. Définir la méthode `depile` (retourne le sommet et le retire de la pile)
6. Définir la méthode `sommet` (retourne le sommet de la pile sans le retirer)

On considère désormais qu'un élément de pile encapsule une valeur entière.

1. Comment représenter un élément de pile ?
2. Définir la classe `ElementPile` qui représente un élément d'une pile. Les attributs de cette classe seront privés. Définir les constructeurs et les accesseurs de cette classe.
3. Définir dans la classe `Pile` la méthode `affiche` qui affiche le contenu d'une pile.

Tester la création d'une pile et sa manipulation en empilant puis dépilant divers éléments...

Correction : début Pile.java

```
public class Pile {
    private int pos; // position dans le tableau
    private ElementPile [] p;

    public Pile () { // creation d'une pile vide de taille 10
        p = new ElementPile[10];
        pos = 0;
    }
    public Pile (int taille) { // creation d'un pile vide de taille donnee
        p = new ElementPile[taille];
    }

    public void empile (ElementPile e) {
        if (pos == p.length) { // plus d'expace
            ElementPile[] tmp = new ElementPile[p.length+10];
            for (int i = 0; i < p.length; i++) tmp[i] = p[i];
            p = tmp;
        }
        p[pos++] = e;
    }

    public ElementPile depile () {
        return p[--pos];
    }
```

```
    public ElementPile sommet () {
        return p[pos-1];
    }

    public boolean estVide () {
        return pos == 0;
    }

    public void affiche () {
        System.out.print("Pile: [");
        for (int i = pos-1; i >= 0; i--) p[i].affiche ();
        System.out.println (" ]");
    }
}
```

fin Pile.java

début ElementPile.java

```
public class ElementPile {
    private int valeur;

    public ElementPile () {
        this.valeur = 0;
    }

    public ElementPile (int valeur) {
        this.valeur = valeur;
    }

    public int getValue () {
        return valeur;
    }

    public void affiche () {
        System.out.print (" "+valeur);
    }
}
```

fin ElementPile.java

Exercice 2 [Tri par insertion et piles] Écrire un programme de tri par insertion d'un ensemble de nombres entiers. Les données sont stockées dans une pile A et le programme doit retourner une pile B contenant ces nombres triés avec le minimum au sommet de la pile. L'algorithme proposé est le suivant : on utilise une pile C qui est vide au début. Tant que la pile A n'est pas vide, on considère les deux cas suivants :

- si la pile B est vide ou si l'élément au sommet de A est plus petit que celui de B :
  on retire l'élément au sommet de la pile A pour empiler dans la pile B, puis si la pile C n'est pas vide on retire tous les éléments de la pile C pour empiler dans la pile B.
- sinon :
  on déplace l'élément au sommet de la pile B à la pile C.

Définir une classe Tri qui contient trois piles A, B et C, une méthode `tri`(Pile A, pile B, Pile C) et la méthode `main()`. La pile A peut être construite à partir d'un tableau d'entiers en utilisant la méthode `empile`. Tester avec la pile A $= \{4, 3, 2, 5, 8, 2, 6, 9, 3\}$.

Correction : début Tri.java

```
public class Tri {

    private Pile A, B, C;

    public Tri () {}

    public Tri (Pile A, Pile B, Pile C) {
        this.A = A;
        this.B = B;
        this.C = C;
    }

    private void tri () {
        while (!A.estVide()) {
            if (B.estVide() || B.sommet().getValue() < A.sommet().getValue()) {
                B.empile(A.depile());
                while (!C.estVide()) B.empile(C.depile());
            }
            else C.empile(B.depile());
        }
    }

    public Pile tri(Pile A, Pile B, Pile C) {
        this.A = A;
        this.B = B;
        this.C = C;
        tri();
        return B;
    }

    public static void main (String[] args) {
        int[] tab = {4,3,2,5,8,2,6,9,3};
        Pile A = new Pile();
        for (int i = 0; i < tab.length; i++) A.empile (new ElementPile (tab[i]));
        Tri t = new Tri();
        t.tri (A, new Pile(), new Pile()).affiche();
    }
}
```

fin Tri.java

Exercice 3 [Jeu des Tours de Hanoi] **Pour ceux qui ont encore du temps**

Étant donné trois piles, et n disques de taille différente empilés sur la première, les plus petits au dessus des plus grands :
Un déplacement consiste à choisir une pile A non vide, et à enlever le disque au sommet pour le mettre au sommet d'une autre pile B, si le sommet de cette pile n'est pas un disque plus petit (sinon, le déplacement est impossible).
On veut déplacer tous les disques de la première pile sur la seconde, en se servant de la dernière. L'algorithme proposé est le suivant : Pour déplacer n disques de la pile A sur la pile C en utilisant la pile B, on déplace (récursivement) $n-1$ disques de A vers B, puis on déplace le disque restant sur le sommet de la pile A vers la pile C, et on déplace (récursivement) $n-1$ disques de B vers C.

Définir une classe `Hanoi` qui représente le jeu : trois piles et le nombre de disques. Définir le constructeur qui initialise le jeu à partir d'un nombre de disques à empiler et initialise les piles dans la situation de départ. Définir la méthode `affiche` qui affiche le contenu des piles. Définir la méthode privée `deplace` et la méthode `joue` qui lance le jeu. Enfin, définir la méthode statique `main` qui à partir d'un entier saisi en ligne de commande initialise et lance le jeu.

Pour visualiser plus agréablement le déroulement du jeu, modifier la méthode deplace afin d'afficher l'état du jeu après chaque déplacement. On vous propose d'utiliser une classe `Afficheur` qui propose un affichage un peu élaboré et quelques options pour le déroulement du jeu. Recopier le ficher `/ens/capron/pub/Afficheur.java` dans votre répertoire de travail. Modifier la classe `Hanoi` en ajoutant un champ de type `Afficheur`. Le constructeur devra prendre en paramètre un objet de ce type et l'affecter a votre champ. Remplacer le contenu de la méthode `affiche` en effectuant simplement un appel à la méthode `afficher` de l'`Afficheur` qui prend en paramètre les trois piles puis le nombre de disques. Compiler le tout et lancer le jeu en exécutant le `main` de la classe `Afficheur`.

Correction : début Hanoi.java

```
public class Hanoi {
    private Afficheur aff;
    private Pile A, B, C;
    private int nbDisques;

    public Hanoi (int nbDisques, Afficheur aff) {
        this.aff = aff;
        this.nbDisques = nbDisques;
        A = new Pile();
        B = new Pile ();
        C = new Pile ();
        for (int i = nbDisques; i > 0; i--) A.empile(new ElementPile (i));
    }

    public void affiche () {
        aff.affiche(A, B, C, nbDisques);
    }

    private void deplace (Pile src, Pile dest, Pile tmp, int n) {
        if (n > 0) {
            deplace(src, tmp, dest, n-1);
            dest.empile(src.depile());
            affiche();
            deplace(tmp, dest, src, n-1);
        }
    }

    public void joue () {
        deplace (A, B, C, nbDisques);
    }
}
```

fin Hanoi.java