Licence Informatique **UE J1IN4W01**
Université de Bordeaux **Programmation 2**
2015-2016 **Travaux Dirigés - feuille 4**

Interface : Exemple d'implémentation d'une pile de formes géométriques

Exercice 1. On souhaite créer une classe `Circle` qui permet la représentation d'un cercle dans un plan grâce respectivement à son centre et son rayon. Compléter le code donné ci-dessous par l'implémentation des méthodes de la forme canonique de la classe `Circle` :

```
public class Circle{

    public Circle(Point2D center, double radius) {
      ...
    }
    public Circle(double radius) {
      ...
    }
    public double getArea() {
      ...
    }
    public Point2D getCenter() {
      ...
    }
    public void setCenter(Point2D p) {
      ...
    }
    public double getRadius(){
      ...
    }
    public void setRadius(double radius){
      ...
    }
    public Circle clone() {
      ...
    }
    public String toString() {
      ...
    }
    public boolean equals(Circle c) {
      ...
    }
    private void updateArea() {
      ...
    }

    private Point2D center;
    private double radius;
    private double area;
}
```

Exercice 2. Soit la classe `Rectangle` vue en feuille2 et donnée ci-dessous.

```
class Rectangle {
    public Rectangle(Point2D topRightCorner, Point2D bottomLeftCorner) {
      ...
    }
    public double getArea() {
      ...
    }
```

```
    public Point2D getBottomLeftCorner() {
      ...
    }
    public Point2D getTopRightCorner() {
      ...
    }
    public void setBottomLeftCorner(Point2D p) {
      ...
    }
    public void setTopRightCorner(Point2D p) {
      ...
    }
    public Rectangle clone() {
    ...
    }
    public String toString() {
      ...
    }
    public boolean equals(Rectangle r) {
      ...
    }
    private void updateArea() {
      ...
    }

    private Point2D bottomLeftCorner;
    private Point2D topRightCorner;
    private double area;
}
```

1. Proposer une interface `Shape` qui permet de manipuler des formes comme `Circle` et `Rectangle`.
2. Modifier les classes `Circle` et `Rectangle` de manière à ce qu'elles implémentent l'interface `Shape`.
3. Ecrire un programme de test.

Exercice 3. On souhaite créer une pile d'objets du type `Shape`. La spécification du type abstrait **Pile** est donné comme il suit :

Opérations : **créer** : $\rightarrow$ **Pile**
empiler : **Pile** $\times$ **Element** $\rightarrow$ **Pile**
dépiler : **Pile** $\rightarrow$ **Pile**
sommet : **Pile** $\rightarrow$ **Element**
vide : **Pile** $\rightarrow$ **Booleen**

Préconditions : **sommet**(p) : ***non*** **vide**(p)

Axiomes :

vide(**créer**) = ***vrai***
vide(p) $\Rightarrow$ **dépiler**(**empiler**(p, e)) = **créer**
vide(**empiler**(p, e)) = ***faux***
dépiler(**empiler**(p, e)) = p
sommet(**empiler**(p, e)) = e

1. Proposer une interface `ShapeStack` pour représenter des piles d'objets du type `Shape`.
2. Créer une classe `ShapeStackFixedSizeArray` qui implémente l'interface `ShapeStack` en stockant l'ensemble des éléments de la pile dans un tableau de taille fixe.
3. Ecrire un programme `TestShape` qui crée et utilise une pile de type `ShapeStack` et qui contient des objets du type `Circle` et `Rectangle`.
4. Proposer une autre implémentation de l'interface `ShapeStack` en stockant l'ensemble des éléments de la pile dans un tableau de taille extensible à savoir le tableau contenant les éléments sera étendu lorsque sa taille deviendra insuffisante pour ajouter un nouvel élément. Tester cette implémentation avec le même programme test.

Exercice 4. Le but de cet exercice est d'appliquer l'algorithme des tours de Hanoï à des empilements de formes variées classées suivant leur aire. On dispose de trois tours. La première est composée de formes d'aire croissante du bas vers le haut, les deux autres sont vides. Il s'agit de déplacer tous les éléments de la première tour vers la troisième en conservant leur ordre. Les tours de Hanoï fonctionnent comme des piles : on peut seulement retirer l'élément situé en haut d'une tour pour le poser au sommet d'une autre tour.
Une illustration de cet algorithme est donnée dans la Fig. 1

1. Comprendre et tester le programme `Hanoi.java`. Ajouter le code nécessaire pour afficher chaque déplacement d'une forme (tour de départ et tour d'arrivée).
2. Modifier le programme `Hanoi.java` en utilisant plusieurs types de piles pour les différentes tours.

```
public class Hanoi {
        //      Algorithme des tours de Hanoi
        public static void moveShapes(ShapeStack fromTower, ShapeStack toTower,
            ShapeStack viaTower, int nbShapes)
        {
                if (nbShapes > 0) {
                        moveShapes (fromTower, viaTower, toTower, nbShapes-1);
                        toTower.push(fromTower.top());
                        fromTower.pop();
                        moveShapes (viaTower, toTower, fromTower, nbShapes-1);
                }
        }


        public static void main(String[] args)
        {
                int n = 2;
                //      Creation des trois tours de formes
                ShapeStack tower0 = new ShapeStackFixedSizeArray (n*2);
                ShapeStack tower1 = new ShapeStackFixedSizeArray (n*2);
                ShapeStack tower2 = new ShapeStackFixedSizeArray (n*2);

                //      La premiere tour est remplie de formes d'aires
                    croissantes
                for (int r = 1; r <=n; ++r){
                        Circle c = new Circle(r);
                        tower0.push(c);
                        Square sq = new Square(2*r);
                        tower0.push(sq);
                }

                //      Affichage des trois tours avant les deplacements
                System.out.println ("Avant les deplacements");
                System.out.println("Tour 1: ");
                System.out.println(tower0);
                System.out.println("Tour 2: ");
                System.out.println(tower1);
                System.out.println("Tour 3: ");
                System.out.println(tower2);

                //      Deplacement des formes de la 1ere tour vers la 3ieme
                Hanoi.moveShapes(tower0, tower2, tower1, n*2);

                //      Affichage des trois tours apres les deplacements
                System.out.println ("Apres les deplacements");
                System.out.println("Tour 1: ");
                System.out.println(tower0);
                System.out.println("Tour 2: ");
                System.out.println(tower1);
                System.out.println("Tour 3: ");
                System.out.println(tower2);
        }
}
```

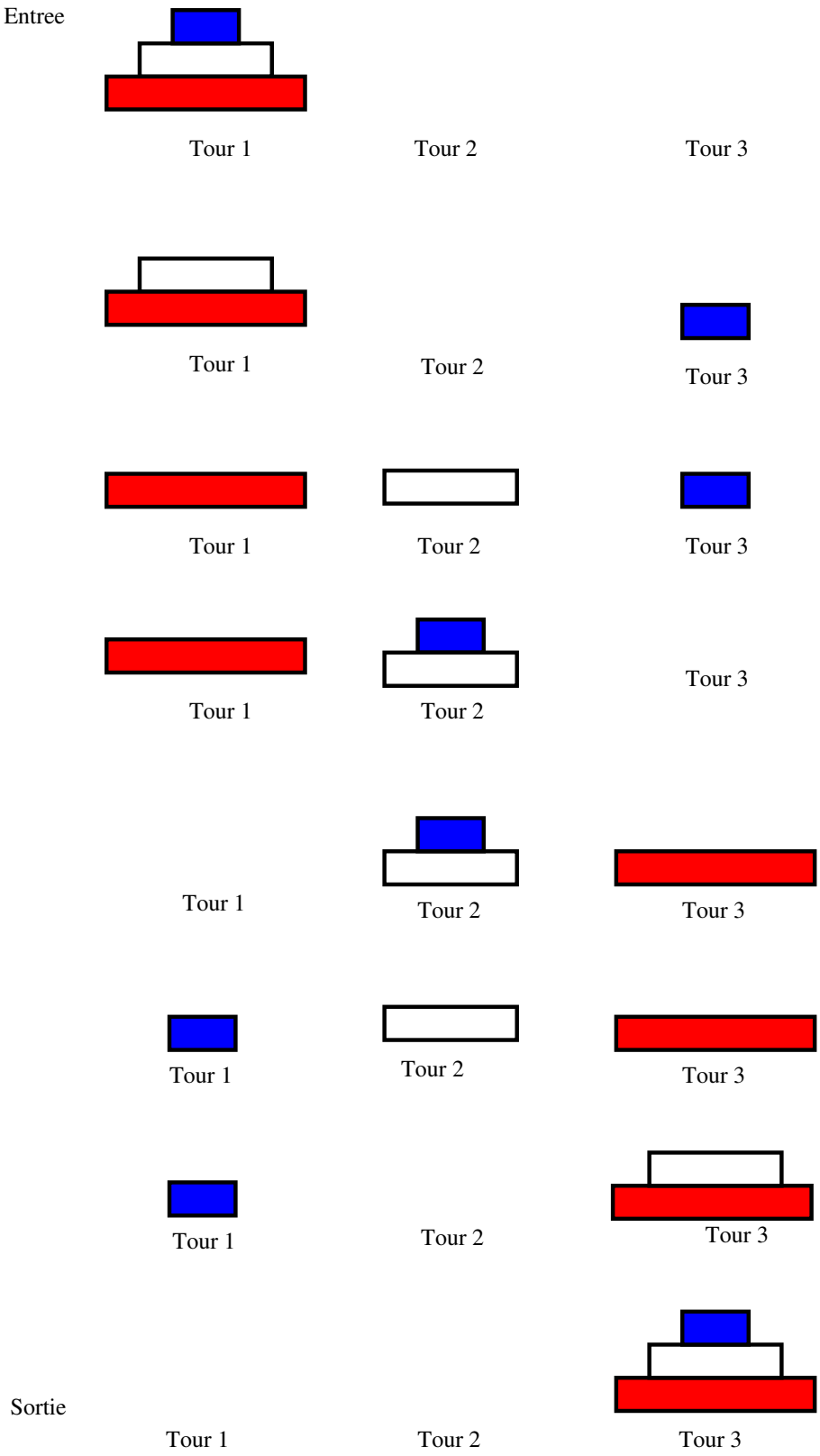

FIGURE 1 – Illustration de l'algorithme des tours de Hanoï avec 3 formes