Université Hassan II
Faculté des Sciences Ain Chock – Casablanca
Département Mathématique et Informatique

Série de TD N° 3 **Héritage, polymorphisme et abstraction**	Licence Professionnelle Développement Informatique POO en Java

Exercice 1 : Gestion des formes

Créez une classe abstraite qui s'appelle **Forme2D** qui représente, de façon générique, des formes géométriques à deux dimensions. Cette classe doit contenir :

— un attribut dont la visibilité est **private** qui s'appelle couleur et qui représente la couleur de la forme en question (vous pouvez utiliser le type Java Color).

— un constructeur par défaut (qui ne prend aucun paramètre) qui crée une forme 2D dont la couleur est noire.

— un constructeur qui prend en paramètre une couleur et qui crée une forme 2D dont la couleur correspond à celle-ci.

— un accesseur qui permet de récupérer la valeur de la couleur de la forme 2D.

— une méthode abstraite **aire** (qui est sensée retourner l'aire de la forme 2D). (N.B. Cette méthode sera redéfinie dans les classes filles de Forme2D pour calculer l'aire correctement)

— une méthode abstraite **perimetre** (qui est sensée retourner le périmètre de la forme 2D). (N.B. Cette méthode sera redéfinie dans les classes filles de Forme2D pour calculer le périmètre correctement)

Maintenant, créez une classe **Rectangle** qui hérite de la classe **Forme2D**. La classe Rectangle doit contenir :

— deux attributs supplémentaires longueur et largeur.

— un constructeur qui prend en paramètres une couleur, une longueur et une largeur et qui instancie le rectangle correspondant.

— les accesseurs adéquats pour les deux attributs **longueur** et **largeur**.

— une redéfinition de la méthode **aire** qui retourne l'aire du rectangle.

— une redéfinition de la méthode **perimetre** qui retourne le périmètre du rectangle.

Créez également une classe Disque qui hérite elle aussi de la classe Forme2D et qui contient :

— un attribut qui s'appelle rayon.

— un constructeur qui prend une **couleur** et un **rayon** en paramètre et qui crée le disque correspondant.

— un accesseur pour l'attribut **rayon**.

— une méthode **diametre** qui renvoie le diamètre du disque.

— les redéfinitions des méthodes **aire** et **perimetre** qui calculent l'aire et le périmètre d'un disque correctement.

Au choix dans l'une des deux classes Rectangle et Disque, essayez d'écrire un modificateur setCouleur qui prend en paramètre une couleur et qui l'attribue à la forme concernée (autrement dit, essayez de modifier l'attribut couleur que Rectangle et Disque héritent de leur classe mère Forme2D). Est-ce que ça marche ? Pourquoi ?

Dans une fonction main, Déclarer un tableau de 4 formes, deux rectangles et deux disques. Puis avec une boucle for afficher l'air et le périmètre de tous les formes dans le tableau.

```
import java.awt.Color;

public class Rectangle extends Forme2D{

	private double longueur, largeur;

	public Rectangle(double longueur, double largeur, Color couleur) {
		super(couleur);
		this.largeur = largeur;
		this.longueur = longueur;
	}

	public double air() {
		return largeur*longueur;
	}

	public double perimetre() {
		return 2*(largeur+longueur);
	}

	public double getLongueur() {
		return longueur;
	}

	public void setLongueur(double longueur) {
		this.longueur = longueur;
	}

	public double getLargeur() {
		return largeur;
```

```
	}

	public void setLargeur(double largeur) {
		this.largeur = largeur;
	}

}
```

```
import java.awt.Color;

public class Disque extends Forme2D{

	private double rayon;

	public Disque(double rayon, Color couleur) {
		super(couleur);
		this.rayon = rayon;
	}

	public double air() {
		return 2*rayon*rayon*Math.PI;
	}

	public double perimetre() {
		return 2*rayon*Math.PI;
	}

	public double diametre() {
		return 2*rayon;
	}

}
```

Exercice 2 : Interfaces et Héritage Multiple en Java

Dans cet exercice, vous allez créer un système utilisant des interfaces pour illustrer l'héritage multiple en Java. L'objectif est de modéliser différents types d'animaux avec des comportements spécifiques.

1. Créez une interface Comportement avec les méthodes suivantes :
   - void manger()
   - void dormir()
2. Créez une interface Vol qui étend Comportement et ajoute la méthode :
   - void voler()
3. Créez une interface Nager qui étend Comportement et ajoute la méthode :
   - void nager()

4. Créez une classe Animal qui implémente l'interface Comportement. Cette classe doit avoir :
   - Un attribut nom de type String.
   - Un constructeur qui prend en paramètre le nom de l'animal.
   - Une redéfinition des méthodes manger() et dormir() pour afficher des messages.
5. Créez une classe Oiseau qui étend Animal et implémente Vol :
   - Redéfinissez la méthode voler() pour afficher un message spécifique.
6. Créez une classe Poisson qui étend Animal et implémente Nager :
   - Redéfinissez la méthode nager() pour afficher un message spécifique.
7. Créez une classe Canard qui étend Oiseau et implémente à la fois Vol et Nager :
   - Redéfinissez les méthodes voler() et nager() pour afficher des messages indiquant que le canard peut faire les deux.
8. Dans la classe Main, créez des instances de Oiseau, Poisson, et Canard. Appelez leurs méthodes pour démontrer leur comportement.

```
// Interface Comportement
interface Comportement {
   void manger();
   void dormir();
}
// Interface Vol
interface Vol extends Comportement {
   void voler();
}
// Interface Nager
interface Nager extends Comportement {
   void nager();
}
// Classe de base : Animal
class Animal implements Comportement {
   protected String nom;

   public Animal(String nom) {
      this.nom = nom;
   }
   @Override
   public void manger() {
      System.out.println(nom + " mange.");
   }
   @Override
   public void dormir() {
      System.out.println(nom + " dort.");
   }
}
```

```java
// Classe Oiseau
class Oiseau extends Animal implements Vol {
    public Oiseau(String nom) {
        super(nom);
    }
    @Override
    public void voler() {
        System.out.println(nom + " vole dans le ciel.");
    }
}

// Classe Poisson
class Poisson extends Animal implements Nager {
    public Poisson(String nom) {
        super(nom);
    }
    @Override
    public void nager() {
        System.out.println(nom + " nage dans l'eau.");
    }
}

// Classe Canard
class Canard extends Oiseau implements Nager {
    public Canard(String nom) {
        super(nom);
    }
    @Override
    public void nager() {
        System.out.println(nom + " nage et fait des ploufs.");
    }
}

// Classe principale
public class Main {
    public static void main(String[] args) {
        Oiseau oiseau = new Oiseau("Mésange");
        Poisson poisson = new Poisson("Carpe");
        Canard canard = new Canard("Canard Colvert");

        oiseau.manger();
        oiseau.dormir();
```

```
        oiseau.voler();
        System.out.println();

        poisson.manger();
        poisson.dormir();
        poisson.nager();
        System.out.println();

        canard.manger();
        canard.dormir();
        canard.voler();
        canard.nager();
    }
}
```