TD n°8 – Classes abstraites et interfaces

Ce TD/TP a pour but de mettre en œuvre les notions de classes abstraites et d'interfaces.

« Affichable »

Le but de cette exercice est de créer une classe abstraite `Affichable`, dotée d'une seule méthode abstraite *void affiche()*. Deux classes, `Entier` et `Flottant`, dérivent de cette classe. La méthode *main* utilise un tableau hétérogène d'objets de type `Affichable` qu'elle remplit en instanciant des objets de type `Entier` et `Flottant`.

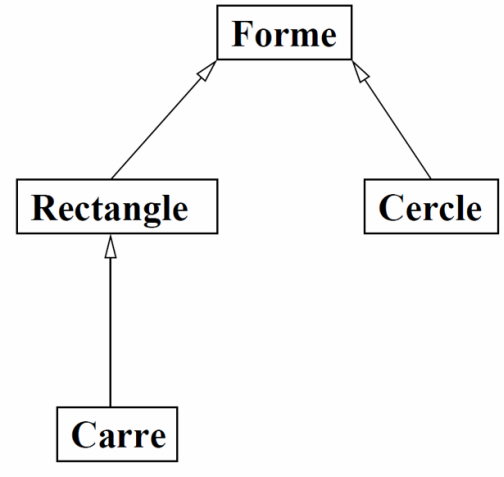
1. Réaliser les classes `Affichable`, `Entier` et `Flottant`, et `TestAffiche` permettant de mettre en œuvre la notion de classe abstraite et le polymorphisme.
   Vous différencierez les méthodes *affiche* des classes `Entier` et `Flottant` en présentant ces objets de la manière suivante :

   ```
   Je suis un entier de valeur 25
   Je suis un flottant de valeur 1.25
   ```

2. La classe `Affichable` ne contient aucun champ et une seule méthode abstraite. A quel type de classe cela correspond-il plus particulièrement ?
3. Créer une nouveau package dans lequel vous transformerez la classe `Affichable` en interface et mettrez à jour les classes `Entier` et `Flottant`.

« Surfaçable »

Reprendre les classes `Rectangle` et `Carre` commencées au TD5 et étendre le package avec une interface `Forme` et une classe `Cercle` comme sur la hiérarchie ci-dessous.



1. L'interface `Forme` contiendra l'en-tête de la méthode *double surface()* permettant de calculer la surface de formes géométriques, et la méthode *void afficheSurface()* par **défaut** permettant d'afficher :

   ```
   Forme géométrique de surface 12.5
   ```

2. Modifier les classes `Rectangle` et `Carre` afin d'implémenter l'interface `Forme`
    a. Modifier le type retour de la méthode *surface()* écrite au TD5 afin d'être compatible avec l'en-tête de l'interface. Que remarquez-vous ?
    b. Redéfinir la méthode *afficheSurface()* qui affichera respectivement :

```
Rectangle 3x5 de surface 15.0
Carre 5x5 de surface 25.0
```

3. Créer la classe `Cercle` contenant son rayon (de type double), implémentant `Forme`, et redéfinissant **uniquement** la méthode *surface()*.
4. Dans votre classe de tests contenant la méthode *main*, définir un tableau d'objets de type `Forme` et instancier à la fois des rectangles, carrés et cercles. Vérifiez l'exactitude de vos méthodes *surface()* dans chacune des classes et tester l'affichage des objets `Rectangle`, `Carre` et `Cercle`.
5. Ajouter une redéfinition de la méthode *afficheSurface()* dans `Cercle` et relancer les tests.
6. La classe `Forme` doit-elle être une classe abstraite ou une interface ? Justifier. Comparer avec la classe `Affichable` de l'exercice 1. Quelle est la relation entre les classes qui nous aide dans le choix entre classe abstraite et interface ?
7. Pour aller plus loin, créer un nouveau package *formesgeo* dans le *Java project* TD8.
   On suppose que la classe `Forme` contient maintenant un attribut : un `Point`, commun à tout objet représentant une forme (centre d'un cercle et coin inférieur gauche d'un rectangle ou d'un carré).
    a. Importer la dernière version de la classe `Point` afin de déclarer un attribut de type `Point` dans la classe abstraite `Forme`.
    b. Ecrire un constructeur par défaut appelant celui de la classe `Point`, ainsi qu'une méthode accesseur retournant le `Point`.
    c. Ecrire les en-têtes correctes des méthodes *surface()* et *afficheSurface()*.
    d. Mettre à jour les classes `Rectangle`, `Carre`, et `Cercle`.
    e. Lancer les tests.
    f. Quel est l'intérêt d'écrire un constructeur dans la classe abstraite `Forme` ?
8. Etendre le package *formesgeo* en ajoutant une classe `Ellipse` qui hérite de `Forme`, mais qui est une généralisation d'un `Cercle`. Une ellipse contient 2 attributs : son grand rayon et son petit rayon.
    a. Définir les méthodes nécessaires dans la classe `Ellipse` afin de l'inclure correctement dans la hiérarchie précédente.
    b. Modifier la classe `Cercle` en conséquence.
    c. Tester en instanciant ces nouveaux objets.
9. Créer les interfaces `Surfacable`, `Affichable`, `Deplacable` et `Pivotable` (cf. transparents de cours)
    a. Quelle(s) classe(s) doi(ven)t implémenter quelle(s) interface(s) ?
    b. Modifier votre code en conséquence.