TP5

Exercice – formes géométriques

Nous souhaitons modéliser des formes géométriques, telles que des cercles et des carrés. En particulier, nous voulons pouvoir tester si une forme géométrique est plus grande, en ce qui concerne sa surface, qu'une autre.

1. Proposez une interface `Forme` pour représenter des formes géométriques. Pour créer une interface, faites un clic-droit sur le nom de votre projet (ou sur l'un de ses *packages*) et sélectionnez *New* puis *Interface* (ou cliquez sur la flèche de l'icône raccourcie *New Java Class* dans la barre d'outils, et sélectionnez *Interface*). Dans la fenêtre qui s'ouvre, donnez un nom à l'interface, puis cliquez sur *Finish*. À l'instar des classes, l'interface apparaît maintenant dans votre projet.
2. Créez des classes `Cercle` et `Carre` implémentant l'interface `Forme`, qui permettent de représenter respectivement des cercles et des carrés. Vous pouvez ajoutez les interfaces qu'une classe implémente directement dans la fenêtre de création de la classe en appuyant sur le bouton *Add...*. Ou sinon, après avoir rajouté `implements Forme` à la déclaration de la classe, un carré rouge dans la marge gauche apparaît pour vous signaler des erreurs. Faites un simple clic sur ce carré pour faire apparaître les suggestions de correction et choisissez *Add unimplemented methods*.
3. Donnez une implémentation pour chaque méthode déclarée dans l'interface `Forme`, et ajoutez des constructeurs, des accesseurs et la méthode `toString` à chacune des classes.
4. Créez une classe `TestFormes` ayant une méthode statique `compare` qui prend deux formes géométriques en paramètre et affiche dans la console Java que la première a une plus grande ou une plus petite aire que la seconde, en précisant les aires de chacune.
5. Écrivez dans cette classe un programme de test qui crée des cercles et des carrés de différentes surfaces et les compare.

Exercice – pile de formes géométriques

Nous souhaitons créer une pile de formes géométriques, c'est-à-dire d'objets de type `Forme`. La spécification du type abstrait *Pile* est donnée ci-dessous :

Opérations :

$$\begin{aligned} creer &: && \to Pile \\ empiler &: Pile \times Element && \to Pile \\ depiler &: Pile && \to Pile \\ sommet &: Pile && \to Element \\ vide &: Pile && \to Booleen \end{aligned}$$

Pré-conditions :

$$\begin{aligned} sommet(p) &: non(vide(p)) \\ depiler(p) &: non(vide(p)) \end{aligned}$$

Axiomes :

$$\begin{aligned} vide(creer) &= vrai \\ vide(p) \implies depiler(empiler(p,e)) &= creer \\ vide(empiler(p,e)) &= faux \\ depiler(empiler(p,e)) &= p \\ sommet(empiler(p,e)) &= e \end{aligned}$$

1. Proposez une interface `PileFormes` pour représenter des piles d'objets de type `Forme`. Pensez à bien structurer votre projet en *packages*.
2. Créez une classe `PileFormesAvecTableauTailleFixe` qui implémente l'interface `PileFormes` en stockant l'ensemble des éléments de la pile dans un tableau de taille fixe. Lorsque la pile est pleine (*i.e.*, nombre d'éléments = taille du tableau), l'ajout d'un nouvel élément dans la pile est simplement ignoré.
3. Écrivez un programme de test dans une classe `TestPileFormes` qui crée et utilise une pile de type `PileFormes` contenant des objets de type `Cercle` et `Carre`.
4. Proposez une autre implémentation de l'interface `PileFormes` en stockant l'ensemble des éléments de la pile dans un tableau de taille extensible. Lorsque la taille du tableau devient insuffisante pour ajouter un nouvel élément, le tableau est étendu en doublant sa taille plus un.
5. Testez cette implémentation avec le même programme de test (code polymorphe).