TD2 : Classes Java

Florian Boudin

Module X0IC020 - 2013

Exercice 1

Vous devez créer la classe `ResultatModule` qui permet de mémoriser les notes obtenues à un module. Cette classe doit posséder les champs suivants : identifiant du module, note de contrôle continu (CC), note de travaux pratiques (TP) et une note d'examen.

1. Ajoutez à la classe un constructeur de signature `ResultatModule(String unIdent)` ainsi que les méthodes permettant d'accéder (`getIdentifiant`, `getNoteCC`, `getNoteTP` et `getNoteExamen`) et de modifier (`setIdentifiant`, `setNoteCC`, `setNoteTP` et `setNoteExamen`) les champs de la classe. Vous ajouterez un second constructeur permettant de renseigner tous les champs de la classe.
2. Ajoutez la méthode `moyenne` qui calcule la moyenne des notes du module et la retourne, ainsi qu'une méthode `valide` qui retourne vrai si la moyenne du module est supérieure à 10. Pour simplifier les calculs, on suppose que les coefficients des notes sont 0.3 pour le CC, 0.2 pour le TP et 0.5 pour l'examen, et ce quel que soit le module.
3. Discutez des différentes définitions possibles de l'égalité entre deux instances de la classe `ResultatModule`. Ajoutez ensuite la méthode de signature `boolean equals(ResultatModule unModule)` en fonction de votre définition.
4. Plusieurs structures de données et algorithmes sur celles-ci supposent que les objets peuvent être comparés par une relation d'ordre. Java exprime cette hypothèse en demandant qu'une classe implémente l'interface `Comparable`. Discutez des différentes définitions possibles de l'ordre des instances de `ResultatModule`. Modifiez la classe `ResultatModule` pour qu'elle implémente l'interface `Comparable` et ajouter la méthode de signature `int compareTo(ResultatModule unModule)`.

   Extrait de la documentation Java : compareTo returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.
5. Il arrive que les objets doivent être comparés selon plusieurs relations : parfois, selon l'identifiant, parfois selon la moyenne, etc. Dans ce cas, associer à la classe un ordre naturel en lui faisant implémenter l'interface `Comparable` n'est pas suffisant. L'API de Java fournit un autre interface pour cela : `Comparator`. Pour les autres ordres, proposez plusieurs implémentations de l'interface `Comparator` (e.g. `IdentifiantComparator`, `NoteExamenComparator`, etc.).

   On peut alors créer un objet comparateur et le passer en argument à certaines méthodes qui l'utilisent, par exemple :

```
Comparator<ResultatModule> comparateur = new NoteExamenComparator();

ResultatModule[] tableau = new ResultatModule[3];
tableau[0] = new ResultatModule("Module 1", 10, 11, 12); // 11.2
tableau[1] = new ResultatModule("Module 2", 11, 12, 11); // 11.2
tableau[2] = new ResultatModule("Module 3", 12, 10, 10); // 10.6

Arrays.sort(tableau);
System.out.println(Arrays.toString(tableau));
// Affichage de [Module 3, Module 1, Module 2]

Arrays.sort(tableau, comparateur);
System.out.println(Arrays.toString(tableau));
// Affichage de [Module 3, Module 2, Module 1]
```

Exercice 2

Soient les deux classes décrites ci-dessous.

```
public class SuperClasse {
  int a = 0;

  public void f() {
    a++;
  }

  public void affiche() {
    System.out.println(a);
  }
}

public class SousClasse extends SuperClasse {
  int b = 4;

  public void g() {
    b--;
  }
}
```

1. Dans la classe utilisatrice décrite ci-dessous, indiquez ce que fait chacune des instructions. Corrigez les éventuelles erreurs et indiquez la sortie du programme.

```
public class Demo {
  public static void main(String[] args) {
    SuperClasse classe1 = new SuperClasse();
    SousClasse classe2 = new SousClasse();
    classe1.f();
    classe1.g();
    classe2.f();
```

```
        classe2.g();
        classe1.affiche();
        classe2.affiche();
    }
}
```

2. On souhaite ajouter une fonction `somme` dans la classe `SousClasse` qui retourne la somme des valeurs des champs `a` et `b`. Est-il possible d'ajouter une telle méthode dans la classe `SousClasse` ? Si oui, quel est le code Java de la méthode ?
3. Surcharger la méthode `f` pour que le champ `a` soit incrémenté de 2 et non pas de 1.

Exercice 3

Reprenez le code de la classe `Personne` que vous avez créé en faisant les exercices de la feuille de TD1. Nous souhaitons définir la classe `Employe`, sous-classe de la classe `Personne` dont les champs spécifiques sont : son année d'entrée dans l'entreprise et son salaire mensuel.

1. Définissez le constructeur `Employe(String unNom, int uneAnnee, int unSalaire, int uneAnneeEntree)` ainsi que les méthodes permettant d'accéder et de modifier les champs spécifiques à cette classe.
2. Ajoutez la méthode `augmentation(float pourcentage)` qui permet d'augmenter le salaire d'un employé de `pourcentage`%.
3. Ajoutez la méthode `anciennete(int anneeCourante)` qui retourne la chaîne de caractères "nouveau" si, par rapport à l'année passée en paramètre, l'employé est entré dans l'entreprise depuis moins de deux ans. La chaîne de caractères "habitué" sera retournée si l'employé compte de 2 à 8 ans d'ancienneté, et "ancien" si l'employé compte plus de 8 ans d'ancienneté.

Exercice 4

Dans cet exercice, vous allez encore une fois utiliser la notion d'héritage pour représenter des formes mathématiques. Le cercle, le rectangle et le triangle sont des formes. Les classes qui les définissent héritent de la classe mère forme. Le diagramme ci-dessous illustre les différentes classes que vous allez devoir créer.

La classe Forme

Sachant que toute forme géométrique est définie par une couleur et une position en X et Y définissant les coordonnées du point de référence pour placer la forme à l'écran, et que la couleur varie entre 0 et 10, le champ X est compris entre 0 et 800 et le champ Y est compris entre 0 et 600.

1. Définir les champs de la classe `Forme` en mode `protected`.
2. Définir des constantes pour la largeur (800) et la hauteur (600) de la fenêtre d'affichage ainsi que pour le nombre de couleurs maximum proposé (10).

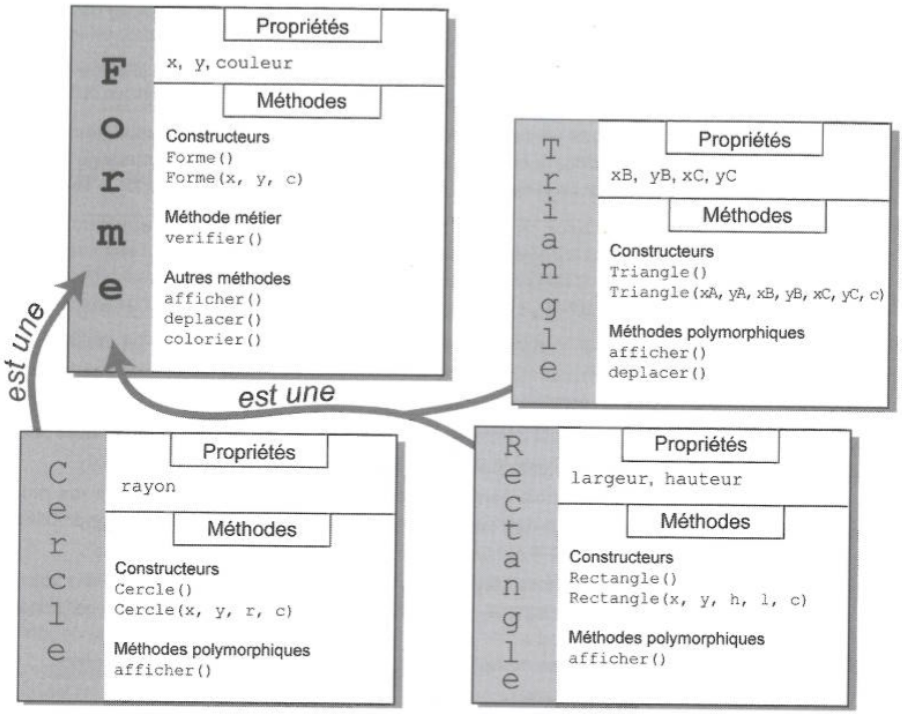

FIGURE 1 – Diagramme des classes Cercle, Rectangle, Triangle et Forme.

3. Créer la méthode `verifier()` et définissez la comme une méthode métier (invisible). Cette méthode vérifie la validité des valeurs pour tous les champs de la classe (couleur, x, y) et qui demande la saisie d'une valeur tant que celle-ci n'est pas dans l'intervalle demandé. L'appel à la méthode pourra s'effectuer de la façon suivante :

```
couleur = verifier("couleur", 0, couleurMax);
```

   Surchargez la méthode `verifier()` en créant une méthode vérifiant une valeur passée en paramètre.

```
couleur = verifier("couleur", 0, couleurMax, uneCouleur);
```

4. Écrire un constructeur :
   - Par défaut qui permet de saisir les données d'une forme. Les données saisies doivent être vérifiées en utilisant la première forme de la méthode `verifier()`.
   - Muni de trois paramètres permettant d'initialiser directement les champs de la classe Forme. Les données passées en paramètres doivent être vérifiées en utilisant la seconde forme de la méthode `verifier()`.
5. Écrire la méthode `deplacer()` qui déplace une forme à partir des valeurs passées en paramètres. Par exemple, si le point de référence est positionné en (100, 100), la méthode `deplacer(10, 10)` apour résultat de placer le point de référence en (110, 110). Les nouvelles coordonnées doivent être vérifiées.

6. Écrire la méthode `colorier()` qui change la couleur de la forme en fonction de la valeur passée en paramètre. La valeur de la nouvelle couleur doit être vérifiée.
7. Écrire la méthode `afficher()` qui affiche les champs de la classe `Forme`.

La classe Rectangle

Sachant qu'un rectangle est une forme géométrique possédant une hauteur dont la valeur est comprise entre 0 et 600 et une largeur dont la valeur est comprise entre 0 et 800.

1. Définir la classe `Rectangle` à partir de la classe `Forme`.
2. Définir les champs de la classe `Rectangle` en mode privé.
3. Écrire un constructeur :
   - Par défaut qui permet de saisir la hauteur et la largeur d'un rectangle. Ces valeurs doivent être vérifiées.
   - Muni de cinq paramètres permettant d'initialiser directement l'ensemble des champs x, y, couleur, largeur et hauteur. Ce dernier utilise le constructeur de `Forme` et doit vérifier les valeurs passées en paramètres.
4. Écrire la méthode `afficher()` qui affiche les champs des classes `Rectangle` et `Forme`.
5. Écrire les méthodes `perimetre()` et `surface()`.

La classe Cercle

Sachant qu'un cercle est une forme géométrique possédant un rayon dont la valeur est comprise entre 0 et 600.

1. Définir la classe `Cercle` à partir de la classe `Forme`.
2. Définir les champs de la classe `Cercle` en mode privé.
3. Écrire un constructeur :
   - Par défaut qui permet de saisir le rayon d'un cercle. Cette valeur doit être vérifiée.
   - Muni de quatre paramètres permettant d'initialiser directement l'ensemble des champs x, y, couleur et rayon. Ce dernier utilise le constructeur de `Forme` et doit vérifier les valeurs passées en paramètres.
4. Écrire la méthode `afficher()` qui affiche les champs des classes `Cercle` et `Forme`.
5. Écrire les méthodes `perimetre()` ($2\pi R$) et `surface()` (πR^2).
6. Écrire la méthode `estDedans()` retournant un booléen et permettant de tester si un point (x, y) se trouve à l'intérieur du cercle.

La classe Triangle

Sachant qu'un triangle est une forme géométrique possédant trois sommets dont les valeurs en X sont comprises entre 0 et 800 et en Y sont comprises entre 0 et 600.

1. Définir la classe `Triangle` à partir de la classe `Forme`.
2. Définir les champs de la classe `Triangle` en mode privé. Les coordonnées de la classe `Forme` correspondent au premier sommet.
3. Écrire un constructeur :
   - Par défaut qui permet de saisir le rayon d'un triangle. Cette valeur doit être vérifiée.

  - Muni de sept paramètres permettant d'initialiser directement l'ensemble des champs x, y, couleur, x1, y1, x2 et y2. Ce dernier utilise le constructeur de `Forme` et doit vérifier les valeurs passées en paramètres.

4. Écrire la méthode `afficher()` qui affiche les champs des classes `Cercle` et `Forme`.

L'application FaireDesFormes

1. Ecrire une application qui permet la création d'un cercle, d'un rectangle et d'un triangle.
2. Vérifiez que les valeurs des champs de chaque classe ne peuvent être saisies en dehors des limites imposées. Affichez les valeurs des formes.
3. Déplacer toutes les formes de 10 pixels en X et 20 en Y. Que se passe-t-il pour le triangle ? Comment remédier au problème ?
4. Afficher les valeurs de périmètre et de surface des formes. Tester la méthode `estDedans()` de la classe `Cercle`.

Sources

Ce document est une adaptation d'un sujet de TD similaire rédigé par Philippe Lamarre dans le cadre du cours de Programmation Orientée Objet (S31I070) et des exercices du livre de Java de Anne Tasso. Rédigé par Florian Boudin, 2011–12.