M1 Informatique - HMIN102 - Ingénierie Logicielle

Année 2017-2018, session 1 de janvier 2018.

Durée : 2h00. Documents non autorisés. Toutes les parties sont indépendantes, ordonnez les comme il vous convient. Le barême est globalement indicatif. La précision et concision des réponses est prise en compte ; si 1 terme suffit, ne pas en donner 10 en espérant que l'un d'eux soit la bon ; le bon choix des termes est aussi important que la validité des codes.

Contexte. Plaçons nous dans le contexte de la réalisation de *examdraw*, un éditeur de dessins minimal. Chaque dessin (instance de la classe `Dessin`) y possède un titre et une collection de figures, c'est-à-dire de lignes, cercles, rectancles, etc. *Examdraw* est un framework extensible par composition ou spécialisation.

L'implanteur peut notamment l'étendre par de nouvelles sous-classes de la classe `Figure`. Nous allons discuter d'une version version "examen" de l'implantation *Java* ce programme. Pour ne pas compliquer l'exercice avec du code graphique *AWT* ou *SWING*, on réalisera l'affichage via une méthode (`drawDescription()`) qui affiche sur un terminal une description textuelle (une chaîne) d'un dessin. L'implantation des différente figures (Ligne, Cercle, etc) utilise la classe prédéfinie `java.awt.Point` (Un *point* est un objet doté d'une abscisse et d'une ordonnée). Une ligne peut ainsi être implantée par deux points, un cercle par un point (son centre) et un entier (son rayon), uncarré par un point (un de ses coins) et une longueur de côté, etc.

Fig. 1 : un dessin réalisé avec *examdraw* version graphique.

Ci-contre (listing 1) un exemple de création et d'affichage d'un dessin (les coordonnées ne sont pas signifiantes). Ci-dessous (listing 2) des extraits du code des classes `Dessin` et `Figure` qui constituent un exemple minimal de framework. Les deux méthodes `drawDescription()` (celle de `Dessin` et celle de `Figure`) représentent le coeur exécutable du framework. Elles sont réutilisables et adaptables sans modification de leur code code car elle sont paramétrées. (Note : La méthode `String toString()` possède une définition sur la classe `Object` et est redefinie sur de nombreuses sous-classes, dont `java.awt.Point` ; la méthode `toString()` de `Point` rend la chaine : "abscisse@ordonnée")

```
1  Dessin d1 = new Dessin("mon premier dessin");
2  d1.add(new Cercle(new Point(5,5),1));
3  d1.add(new Ligne(new Point(5,2), new Point(5,4));
4  d1.drawDescription(); // affiche le texte ci-dessous
5  ...
6  Pour dessiner mon premier dessin, tracez :
7  - un cercle de centre: 5.0@5.0 et rayon 1.0
8  - une ligne de 5.0@2.0 à 5.0@4.0
```

Listing 1 – code de création d'un dessin.

```
1  class Dessin {
2      protected Set<Figure> contenu; protected String title; // deux attributs
3      public Dessin(String t){ contenu = new HashSet<Figure>(); title = t;} //constructeur
4      public void add(Figure s) { contenu.add(s); } //méthode add
5      public void drawGraphic() { ... } //affichage graphique non traité dans cet examen

7      public void drawDescription() { //affichage textuel pour cet examen
8          System.out.println("Pour dessiner " + title + ", tracez : ");
9          for (Figure f : contenu) f.drawDescription(); } ... }

11 public abstract class Figure{ ...
12     public void drawDescription() { System.out.println ( this.toString() ) ;} ... }
```

Listing 2 – code partiel des classes `Dessin` et `Figure`

1 Réutilisation - Environ 6 points

1. Dans tout le reste de l'examen, vous n'aurez pas une seule fois à redéfinir la méthode `drawDescription()` de `Figure`, en effet elle est paramétrée. Nommez et décrivez le type de paramétrage qui a été utilisé.
2. Expliquez ce qu'est une fonction d'ordre supérieur. Expliquez pourquoi la méthode `drawDescription()` de `Figure` peut être considérée comme une fonction d'ordre supérieur ?
3. Donnez en Java la définition de la classe `Cercle` : uniquement attribut(s), constructeur(s) et méthode(s) utiles à l'énoncé (n'écrivez pas les accesseurs en lecture, on supporsera qu'ils existent néanmoins).
4. Y-a-t-il une affectation polymorphique explicite ou induite dans le code du listing 2? Si oui, quel rôle joue-t-elle

dans la réutilisabilité d'*examdraw*?

5. Tests Automatisés. Décrivez comment vous organiseriez un ensemble de jeu de tests **Junit** pour *examDraw*.

2 Réutilisation et Typage statique - Environ 3 points

1. Pour éviter qu'un utilisateur n'ajoute deux fois une même figure dans un dessin, la classe `Dessin` utilise un ensemble (instance de `(HashSet)`), collection non ordonnée et sans duplicats, pour stocker la collection des figures. Un *hashSet* se manipule comme un *vector* mais l'insertion d'un nouvel élément entraine une comparaison (via l'envoi du message `equals(...)`) avec chacun des éléments présents au moment de l'insertion. Pour que la méthode `add` de `Dessin` fonctionne, il est donc nécessaire de redéfinir la méthode `equals` dans toutes les sous-classes de `Figure`.
   → Donnez le code de la redéfinition de la méthode `boolean equals(...)` de `Object` sur votre classe `Cercle`.
2. Commentez votre redéfinition précédente en utilisant avec pertinence les termes associés aux règles de substituabilité de Liskov.
3. Soit la variable `f1` définie et valuée par l'affectation "`Figure f1 = new Cercle(new Point(3,3),1);`", donnez la liste ordonnées des exécutions de méthodes (donnez le nom et classe de définition) suite à l'envoi du message (appel de méthode) : `f1.drawDescrition()`.

3 Lignes de produit - Environ 3 points

1. On imagine maintenant une ligne de produit *examdraw-spl* permettant de créer des variantes d'*examdraw*. Pour générer un *examdraw* spécifique, *examdraw-spl* est doté d'un *Modèle de FeatureModel*; celui de la figure 2 en est une représentation réduite.
   Citez trois avantages à l'utilisation de l'ingénierie des lignes de produits logiciels.
2. Donnez un exemple de configuration valide, et un exemple de configuration non valide. Donnez un exemple de modification du feature model qui ne change pas son ensemble de configurations valides.
3. Imaginons que nous rajoutions une contrainte cross-tree "figure requires couleur". Que pourriez-vous dire sur la caractéristique "couleur" ? Quelle modification apporter au feature model dans ce cas-là ?

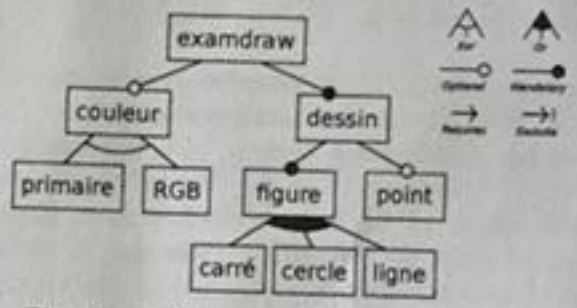

Fig. 2 : un *feature model* pour *examdraw*.

4 Schéma *Adapter* : Environ 3 points

On souhaite qu'une instance de la classe `Java.awt.Point` de la bibliothèque Java puisse faire partie des figures constituant un dessin (par exemple pour faire l'oeil du bonhomme de la figure 1).

1. `d1` étant le dessin du listing 1, et `Point` étant `Java.awt.Point`, pourquoi n'est-il pas possible d'écrire : `d1.add(new Point(x,y))` ?
2. Définissez en Java une classe d'adaptation `PointAdapter`, la plus simple possible, rendant la chose possible et opérationnelle. Expliquez votre construction.
3. Donnez dans ce contexte un exemple du problème de perte du receveur initial rencontré en utilisant la composition ou l'aggrégation.

5 Des dessins arborescents : environ 3 point

1. Proposez une modification de l'architecture d' *examdraw* permettant qu'un dessin puisse être composé d'un dessin. Vous pouvez bien sûr utiliser un schéma connu (auquel cas nommez le).
2. Dans votre nouvelle architecture, donnez le code de la méthode *equals* de la classe `Dessin`.
3. Faite une modification à l'architecture résultante pour qu'à partir de toute figure, on puisse donner le plus petit dessin auquel elle appartient.

6 Schéma *Command* : environ 2 points

Soit à mettre en place dans *examdraw* un système de *Undo* généralisé. On s'intéresse ici à la partie qui concerne l'annulation (*undo*) des ajouts de figures donc à la possibilité d'enlever dans l'ordre inverse les figures que l'on a ajoutées dans un dessin. A noter que la collection des figure est de type ensemble (*Set*) et que l'on n'a aucune garantie sur l'ordre dans lequel les figures y sont stockées.

1. Proposez via des description de classes bien choisis (superclasses, attributs, entête de méthodes, envois de message clé) une architecture mettant en oeuvre le schéma *Command* pour traiter ce cahier des charges.