FDS - M1 Informatique - HAI712I - Ingénierie Logicielle

Année 2021-2022, session 1 de janvier 2022.

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 *FramPict*, un framework jouet pour faire des images ou des dessin. Un dessin y est un ensemble de formes ; les formes sont à la base des lignes, des cercles ou des rectangles. Toute instance de la classe `Dessin` possède un titre et une collection de formes. Une forme est représentée par une instance d'une sous-classe de la classe `Forme`. *FramPict* est à la fois un logiciel utilisable (il est opérationnel) et à la fois un framework utilisable par des programmeurs qui peuvent l'étendre pour lui ajouter ou pour adapter ses fonctionnalités.

Nous discutons dans cet exercice d'une implantation en *Java* de *FramPict* et les questions portent sur sa réalisation, son architecture et ses qualités logicielles. Pour simplifier, on n'écrit pas de vraies méthodes de dessin ; on décrit un dessin par un texte. La méthode (`drawDescription()`) affiche au terminal une description textuelle (une chaîne de caractères) d'un dessin. L'implantation des différente sortes de formes (Ligne, Cercle, etc) utilise la classe prédéfinie `java.awt.Point`. Un tel point possède une abscisse et d'une ordonnée). Une ligne (un segment de droite) peut ainsi être représentée par deux points (une origine et une extrémité), un cercle par un point (son centre) et un entier (son rayon), un rectancle par deux points (son point supérieur gauche et son point inférieur droit) ; etc.

Le listing 2 donne des extraits du code des classes `Dessin` et `Forme` avec leur méthode respective `drawDescription()`. Ces 2 classes dans cette version représentent l'architecture de base du framework *FramPict*. Le listing 1 donne un exemple d'utilisation de ces classes [a].

a. Précision : on suppose que la redéfinition de la méthode `toString()` sur la classe `java.awt.Point` rend la chaine : "abscisse@ordonnée".

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

Listing 1 – exemple de création d'un dessin.

```
1  class Dessin {
2      protected Set<Forme> contenu; protected String title; //deux attributs
3      public Dessin(String t){ contenu = new HashSet<Forme>(); title = t;} //un constructeur
4      public void add(Forme s) { contenu.add(s); }
5      public void drawDescription(){
6          System.out.println(this.getDescription() + " obtenu par : ");
7          for (Forme sh : contenu) sh.drawDescription(); }
8      public String getDescription(){
9          return ( "Un dessin de " + title ); }
10     ...}

12 public abstract class Forme{
13     public void drawDescription() { System.out.println (this.getDescription());}
14     public abstract String getDescription();
15     ... }
```

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

A Réutilisation - Environ 5 points

1. Considérons la méthode `drawDescription`, applicable à toutes les formes, factorisée sur la classe `Forme`, ainsi que le listing 3.

   a) Commentez les instructions du listing 3 (2 lignes maximum).

   b) Expliquez pourquoi la méthode `drawDescription()` de `Forme` peut être considérée comme une fonction d'ordre supérieur ?

   c) En faisant un lien avec le code de la classe `Dessin`, et avec les questions a et b précédentes, expliquez succinctement quel type de paramétrage est utilisé, quel(s) est(sont) le(s) paramètre(s), quel(s) est(sont) le(s) argument(s) mis en jeux, lors de l'exécution des envois de messages du listing 3. A quel moment la liaison dynamique a-t-elle été nécessaire (ou pas) ?

```
1  Forme f;
2  f = new Cercle(new Point(5,5),1));
3  f.drawDescription();
4  f = new Ligne(new Point(5,2), new
       Point(5,5));
5  f.drawDescription();
```

Listing 3 –

2. Donnez en *Java* une définition de la classe `Ligne` conforme au résultat d'exécution montré au listing 1 : uniquement attribut(s), constructeur(s) et méthode(s) utiles à l'énoncé.
3. Tests Automatisés. Décrivez comment vous organiseriez un ensemble de jeu de tests **Junit** pour un framework comme *frampict*.

B Réutilisation et typage statique - Environ 6 points

Pour éviter qu'un utilisateur n'ajoute deux fois une même forme à un dessin (par exemple deux instances de cercles qui auraient le même centre et le même rayon), ce qui ne sert à rien, l'attribut `contenu` de *Dessin* est de type `Set` (`Set` est une interface qui définit les collections sans doubles) et référence (voir listing 2) une instance de la classe `HashSet` qui implante l'interface `Set`. Le détail de fonctionnement d'un HashSet n'est pas discuté dans cet exercice et peut être ignoré. Il suffit de savoir que si on essaye d'ajouter à un "*set*", un nouvel objet "equal" (selon le résultat de la méthode `... equal(...)`) à un objet déjà présent dans le *set*, alors le nouvel objet n'est pas ajouté.

1. Pour que le programme fonctionne, Il faut que les formes présentes dans la collection `contenu` puissent être comparées avec equal. On choisit en première solution de redéfinir la méthode `boolean equals(Object o)` definie sur `Object` dans toutes les sous-classes de `Forme`. Note : vous pouvez ignorerez entièrement ici les problèmes liés au *hashcode* des objets.
   Donnez le code Java de la redéfinition de la méthode equals sur votre classe `Ligne`.
2. Commentez votre redéfinition de la question précédente notamment en utilisant avec pertinence les termes associés aux règles de substitution (parfois dites règles de spécialisation) de Liskov.
3. Soit la variable `f1` donnée par :
   `Forme f1 = new Ligne(new Point(5,2), new Point(5,4));`
   Donnez la liste ordonnées des méthodes exécutées (pour chaque élément de la liste, donnez le nom de la méthode et la classe où elle est définie) suite à l'envoi du message :
   `f1.drawDescription();`
4. L'instruction : "`Vector<Forme> v = new Vector<Cercle>();`" est-elle acceptée par le compilateur Java ? Commentez.
5. En seconde solution, on imagine de factoriser une méthode `equal` sur la classe `Forme` ; voyez vous une solution pour ce faire qui pourrait convenir à notre problème ? si non pourquoi ? si oui donnez votre idée avec une phrase ou du code.

C Une modification d'architecture pour des dessins arborescents - Environ 4 point

1. Modifiez l'architecture de base de *frampict*, pour permettre qu'un dessin puisse être composé d'un dessin. Ce qui permettra d'exécuter les instructions du listing 4.

Donnez un UML précis et les éléments clé du code faisant que cela fonctionne. Si vous utilisez un schéma de conception connu, précisez le. Le code de l'indentation dans l'affichage n'est pas demandé mais c'est un plus si vous le donnez.

```
1  Dessin d1 = new Dessin("croquis annexe");
2  d1.add(new Cercle(new Point(5,5),1));
3  d1.add(new Cercle(new Point(7,7),1));
4  Dessin d2 = new Dessin("croquis principal");
5  d2.add(new Cercle(new Point(6,6),1));
6  d2.add(d1);
7  d2.drawDescription(); // affiche le texte ci-dessous
8  ----
9  Un dessin de croquis principal obtenu par :
10 - un cercle de centre: 6.0@6.0 et rayon 1.0
11 - Un dessin de croquis annexe obtenu par :
12   - un cercle de centre: 5.0@5.0 et rayon 1.0
13   - un cercle de centre: 7.0@7.0 et rayon 1.0
```

Listing 4 –

2. Améliorez, si-besoin, cette nouvelle architecture, ou expliquez en quoi la précédente suffit, afin de pouvoir définir une nouvelle méthode `dessinsEnglobants()` rendant, pour une forme donnée, la collection des dessins auquel elle appartient. Pour le "cercle de centre : 7.0@7.0" du listing 4, le résultat serait une collection contenant `d1` et `d2`.
   Vous avez liberté de choix pour décider si l'on peut, ou pas, mettre un dessin dans plusieurs autres, ou une forme dans plusieurs dessins distincts. Tout commentaire lié à un(des) schéma(s) de conception connu(s) est intéressant.

D Une modification d'architecture pour intégrer une classe d'une bibliothèque existante dans l'application - Environ 3 points

On repart de l'architecture initiale du framework *FramPict*

On souhaiterait qu'une instance de la classe `Point` de la bibliothèque `Java.awt` puisse être une forme utilisable dans un dessin (pour faire un point dans un dessin, comme l'oeil d'un personnage par exemple). On ne souhaite pas réécrire une classe `Point` puisqu'on en a trouvé une toute faite.

1. Pourquoi n'est-il pas possible qu'un `Java.awt.Point` soit utilisé comme une forme dans *FramPict*?
2. On imagine une solution via une classe d'adaptation `FormPoint`. Si `d1` est l'instance de dessin du listing 2, alors on pourra écrire : `d1.add(new FormPoint(4,5))`. `d1.drawDescription()` affichera en conséquence le texte suivant au terminal :
   "- un point de coordonnées 4.0@5.0".
   Donnez un schéma UML de la solution et les éléments clé du code faisant que cela fonctionne. Si vous utilisez un schéma de conception connu, commentez le.
3. En lien avec la redirection de messages utilisée dans cette solution, donnez dans ce contexte un exemple d'occurrence du problème de perte du receveur initial.

E Faire et défaire les dessins - Environ 2 points

On revient à la version de base de *frampict*. Mais on remplace le `HastSet` par un `Vector` afin de conserver l'ordre dans lequel les formes sont insérées dans les dessins.

On souhaite mettre en place un système de *Undo*. Envoyer le message undo() à un dessin annulerait le dernier ajoût d'une forme.

1. Proposez une solution à ce problème mettant en oeuvre le schéma de conception *Command*. Donnez un UML et/ou des éléments de code clé.