FDS - M1 Informatique - HAI712I - Ingénierie Logicielle

Année 2021-2022, session 2 de mars 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.

On reprend le contexte du sujet de session1. **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 List<Forme> contenu; protected String title; //deux attributs
3      public Dessin(String t){ contenu = new ArrayList<Forme>(); title = t;} //un constructeur
4      public void add(Forme s) { contenu.add(s); }
5      public void revove(Forme s){ contenu.remove(s); }
6      public void drawDescription(){
7          System.out.println(this.getDescription() + " obtenu par : ");
8          for (Forme sh : contenu) sh.drawDescription(); }
9      public String getDescription(){
10         return ( "Un dessin de " + title ); }
11     ...}

13 public abstract class Forme{
14     public void drawDescription() { System.out.println (this.getDescription());}
15     public abstract String getDescription();
16     public abstract double surface();
17     ... }
```

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

A Réutilisation - Environ 6 points

Considérons les listings 2 et 3.

1. Dans le listing 3, on trouve l'instruction `f.drawDescription();`.

   a) De quel type d'instruction s'agit-il ?

   b) Ce type d'instruction est-il fréquemment utilisés dans les frameworks, notamment ceux pour la programmation web ? si oui, pourquoi (qu'a-t-elle d'important pour un framework) ?

```
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. Dans le listing 3, cette instruction `f.drawDescription()` est utilisée 2 fois.
   a) L'exécution de chacune de ces occurrences de l'instruction va invoquer une méthode, qui aura pour nom `drawDescription()`.
   En considérant le code des listings 2 et 3, et en se mettant à la place du compilateur réalisant une analyse statique du code, y-a-t-il une ambiguité quand à savoir quelle méthode `drawDescription()` va être appelée ? Autrement dit, le compilateur doit-il compiler ces instructions en activant la liaison dynamique, ou bien est-ce inutile ?
   b) Chacune des exécutions de ces 2 instructions identiques provoquent l'affichage au terminal de textes différents. On dirait que la méthode exécutée est paramétrée par le texte à afficher. Pourtant la méthode `drawDescription()` ne possède aucun paramètre apparent. Expliquez ce phénomène.
3. Quand une méthode reçoit un objet en argument, que peut-elle faire avec cet objet ? Répondez en faisant un lien avec le code de la classe `Forme`, et avec les questions précédentes.
4. Imaginez le code de la classe `Cercle`, sous-classe de Forme, permettant d'obtenir le listing 1.
   - Est-il nécessaire de redéfinir la méthode `drawDescription()` sur Cercle ? pourquoi ? Si oui, expliquez quel contrainte doit être respectée quand au type de la valeur de retour et montrez que votre redéfinition respecte cette contrainte.
   - Même question exactement pour la méthode `getDescription()`.
5. Décrivez globalement comment vous organiseriez un framework de jeu de tests **Junit** pour un **framework** comme *Frampict* (notez qu'il est question de réutilisation). Donnez le code d'une méthode de test applicable aux formes et une des méthodes `setUp()`.

B Réutilisation et typage statique - Environ 5 points

Pour optimiser l'impression des dessins, on souhaite que la collection des formes (attribut `contenu` de *Dessin* de type `ArrayList`) constituant un dessin soit triée selon la surface des formes, par ordre décroissant.

Pour cela, chaque sous-classe de `Forme` définit une méthode `surface()` [1].

Pour trier, on va utiliser la méthode statique `sort` de la classe `java.util.Collections` (avec un "s" à la fin, à ne pas confondre avec `java.util.Collection`) qui définit des méthodes statiques qui opèrent sur les collections.

Voici la signature et la documentation de la méthode `sort(...)`.
- `public static <T extends Comparable<? super T>> void sort (List<T> list)`
- "Sorts the specified list into ascending order, according to the natural ordering of its elements. All elements in the list must implement the Comparable interface. Furthermore, all elements in the list must be mutually comparable (that is, e1.compareTo(e2) must not throw a ClassCastException for any elements e1 and e2 in the list)."

On note que cette méthode `sort(...)` fonctionne si les éléments de la collection `list` à trier sont des `Comparable<...>` (`java.lang.comparable<T>`), grace à la méthode compateTo(...). Si e1 et e2 sont des `Comparables<...>`, `e1.compareTo(e2)` rend -1 ou 0 ou 1 selon que e1 est inférieur, égal ou supérieur à e2.

1. En utilisant les versions des classes de base données au listing 2, on essaye de définir sur la classe `Dessin`

1. Pour une ligne, on dira que la surface est la distance entre le point origine et le point extrémité. Note : il ne vous est pas demandé d'écrire les méthodes `surface()` ; on suppose qu'elles existent.

la méthode `sort` suivante :
`public void sort() { Collections.sort(contenu); }`
Le compilateur refuse en affichant le message d'erreur suivant : "Bound mismatch : The generic method `sort(List<T>)` of type `Collections` is not applicable for the arguments (`ArrayList<Forme>`). The inferred type Forme is not a valid substitute for the bounded parameter `<T extends Comparable<? super T>>`"
- a) Expliquez l'erreur.
- b) Modifiez ce qui doit l'être pour que cette définition compile sans erreur de compilation dans la classe `Dessin`.

2. Modifiez ce qui doit l'être partout où c'est nécessaire, pour que les instructions du listing 4 s'exécutent comme indiqué au listing 5.

```
1 Dessin d2 = new
      Dessin("ballonBaudruche");
2 d2.add(new Cercle(new Point(5,5),2));
3 d2.add(new Ligne(new Point(5,2), new
      Point(5,9)));
4 d2.add(new Cercle(new Point(7,7),1));
5 d2.add(new Ligne(new Point(5,2), new
      Point(5,5)));
6 d2.drawDescription();
7 d2.sort();
8 System.out.println("--- après tri");
9 d2.drawDescription();
```

Listing 4 –

```
Un dessin de TestDeTri obtenu par :
– un cercle de centre: 5.0@5.0 et rayon 2.0
– une ligne de 5.0@2.0 à 5.0@9.0
– un cercle de centre: 7.0@7.0 et rayon 1.0
– une ligne de 5.0@2.0 à 5.0@5.0
--- après tri
Un dessin de TestDeTri obtenu par :
– une ligne de 5.0@2.0 à 5.0@5.0
– un cercle de centre: 7.0@7.0 et rayon 1.0
– une ligne de 5.0@2.0 à 5.0@9.0
– un cercle de centre: 5.0@5.0 et rayon 2.0
```

Listing 5 –

3. Traduisez en français la contrainte suivante sur le type T : `<T extends Comparable<? super T>>`. Expliquez en quoi vous avez respecté cette contrainte dans votre réalisation.

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

1. Donnez un schéma UML modélisant une relation d'aggrégation où chaque instance d'une classe A utilise au plus 5 instances d'une classe B.
2. Modifiez l'architecture de base de *frampict*, pour permettre qu'un dessin puisse être composé d'un dessin et ce sans que la méthode drawDescription de la classe Forme n'ait à être modifiée. Ceci permettra d'exécuter les instructions du listing 6 pour donner le résultat du listing 7.
Donnez un UML précis et les éléments clé du code faisant que cela fonctionne.

```
1 Dessin d3 = new Dessin("croquis annexe");
2 d3.add(new Cercle(new Point(5,5),1));
3 d3.add(new Cercle(new Point(7,7),1));
4 Dessin d4 = new Dessin("croquis
      principal");
5 d4.add(new Cercle(new Point(6,6),1));
6 d4.add(d3);
7 d4.drawDescription();
```

Listing 6 –

```
Un dessin de croquis principal obtenu
    par :
– un cercle de centre: 6.0@6.0 et rayon 1.0
– Un dessin de croquis annexe obtenu par :
  – un cercle de centre: 5.0@5.0 et rayon
      1.0
  – un cercle de centre: 7.0@7.0 et rayon
      1.0
,
```

Listing 7 –

3. 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 6, le résultat serait une collection contenant `d3` et `d4`.
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 Ajout de fonctionnalités à un objet individuel : Environ 5 points

On souhaite pouvoir ajouter puis éventuellement retirer, à l'exécution, des fonctionnalités à tout objet de type `Forme`, sans pour autant modifier les classes de Forme existantes ni leur méthodes car ces ajouts d'informations supplémentaires ne les concerne pas toutes.

Par exemple, on souhaite que la description d'une forme, choisie individuellement, affiche des informations supplémentaires, comme un commentaire de ce qu'elle représente dans un dessin, ou sa couleur. On voit dans le listing suivant une application de cette idée où on ajoute à un cercle le commentaire qu'il représente un ballon et qu'il est de couleur rouge, sans avoir modifié la classe Cercle.

```
1  Dessin b = new Dessin("un ballon de baudruche");
2  b.add(new Commentaire("le ballon",
3          new Couleur(Color.red,
4                  new Cercle(new Point(3,4), 1))));
5  b.add(new Commentaire("la ficelle",
6          new Ligne(new Point(3,0), new Point(3,3))));
7  b.drawDescription() // produit le texte suivant :
8  "Un dessin de un ballon de baudruche obtenu par :
9  - un cercle de centre: 3.0@4.0 et rayon 1.0, de couleur: rouge (le ballon)
10 - une ligne de 3.0@0.0 à 3.0@3.0 (la ficelle)''
```

Listing 8 – Création de formes avec des fonctionnalités additionnelles.

1. Proposez une architecture pour réaliser cette spécification.
2. Dans cette architecture, donnez le code de la classe `Commentaire`.
3. Faites un dessin représentant l'objet référencé par la variable `b` et les autres objets référencés dans les champs de celui-ci.
4. En quoi l'associativité de l'opérateur Java de concaténation des `String` est-elle importante dans cet exercice ? Associez à votre réponse un commentaire sur le schéma *Décorateur*.
5. Connaissez vous un autre exemple où une architecture identique a été utilisée ?