Architecture Logicielle – TP 07 2022–2023

Le patron *Passerelle*

Exercice 1 – *Captain on the bridge !*

Une compagnie de *n'importe quoi*[1] a mis en place un système pour produire des formulaires d'abonnement à une lettre d'information dans plusieurs formats de manière uniformisée, c'est-à-dire que la même interface sert à produire les formulaires dans tous les types de formats supportés par l'entreprise. Pour l'instant, l'entreprise utilise deux formats : *HTML* et *PDF*. L'interface en question se présente comme ceci :

```
1  package forms;
2
3  public interface FormGenerator {
4    public void addTitleHeader(String formTitle); // Émet un titre centré.
5    public void addSeparator();       // Émet une ligne de separation
6    public void addNameLine();        // Émet un champ nom : _____
7    public void addFirstNameLine();   // Émet un champ prénom : _____
8    public void addEmailLine();       // Émet un champ courriel : _____
9    public String send();             // Retourne le flux du formulaire construit
10 }
```

Le but de la méthode *addTitleHeader* est de produire l'en-tête du document avec le nom de la compagnie suivie du titre du formulaire donné en paramètre. Les autres méthodes *addXxx()* produisent chacun un champ de formulaire consistant en le nom du champ correspondant (par exemple "Nom : " pour *addNameLine()*) suivi d'un espace de saisie, par exemple un *input* pour le format HTML, ou d'une ligne horizontale pour le format PDF.

L'implémentation actuelle comporte une classe abstraite *CommonFormGenerator* qui contient le code, commun à tous les formats, de la méthode *addTitleHeader*. Deux classes concrètes, *HTMLFormGenerator* et *PDFFormGenerator* ont été dérivées de la classe *CommonFormGenerator* pour implémenter les deux formats.

Le code suivant montre un exemple d'utilisation du générateur de formulaire :

```
1  import forms.*;
2
3  public class TestForm {
4    public static FormGenerator getFormGenerator() {
5      // Obscure fonction retournant un generateur de formulaire.
6      // Vous la definirez pour vos tests.
7      // Ex: return new HTMLFormGenerator();
8      return new ...
9    }
10
11   public static void testGenerator(FormGenerator fg) {
12     fg.addTitleHeader("Your newsletter subscription");
13     fg.addSeparator();
14     fg.addNameLine();
15     fg.addFirstNameLine();
16     fg.addSeparator();
17     fg.addEmailLine();
18
19     String Result = fg.send();
20     System.out.println(Result);
21   }
22
23   public static void main(String[] args) {
```

1. Le *n'importe quoi* est un produit dont personne n'a besoin mais que les services marketing excellent à vendre.

```
24        testGenerator(getFormGenerator());
25    }
26  }
```

Problème : l'entreprise étend maintenant ses activités à l'international et souhaite maintenant pouvoir produire les même formulaires en langue anglaise. Le stagiaire employé pour implémenter cette évolution, qui n'a pas suivi le Master GIL, propose de dériver deux nouvelles classes *HTMLFormGeneratorEng* et *PDFFormGeneratorEng*.

Question 1.1 : Dessinez le diagramme UML de cette solution et faites en une critique, en particulier, imaginez ce qu'il se passe lors de l'ajout d'une nouvelle langue (ex : espagnol, hongrois, ...), et/ou d'un nouveau format (ex : XML, odt).

Question 1.2 : Proposez une solution faisant intervenir une couche d'abstraction et une couche d'implémentation qui permettront de séparer les aspects format et linguistique. Donnez le diagramme UML de cette solution.

Indication : dans le présent cas, il peut être difficile d'identifier les responsabilités de haut niveau (abstraction) et de bas niveau (implémenteurs), si bien que l'on pourrait avoir l'impression que le patron pourrait être appliqué dans les deux sens. Il n'en est rien et pour vous aider à trouver la bonne hiérarchie, il vous est conseillé de :

— rédiger complètement l'interface *Implementor*[2],

— écrire complètement le code des implémenteurs concrets (ce n'est pas bien long).

Dans le mauvais sens, il y aura beaucoup de duplication de code. Dans le bon sens, les responsabilités seront correctement encapsulées.

Question 1.3 : Donnez un diagramme d'objets correspondant au choix d'un formulaire en anglais au format PDF.

Question 1.4 : Implémentez la solution validée.

Exercice 2 – *Petit pont*

Dans une application, on souhaite disposer d'un système de journalisation des événements (*logger*). Au début on souhaite simplement disposer d'une méthode `log(String)` qui envoie les messages sur la sortie standard du programme. Par la suite, on voudrait pouvoir envoyer ces messages vers un fichier ou une base de données. Enfin, on souhaiterait que le programme puisse utiliser différents types de logger : un pour les avertissements, un pour les erreurs, faisant automatiquement précéder leur message par `"warning :"` et `"error :"` respectivement.

Question 2.1 : Proposez une architecture et donnez son diagramme UML.

Question 2.2 : Fournissez une implémentation de votre solution.

2. Mais ça, vous le faites systématiquement ! N'est-ce pas ?