UNIVERSITÉ DE ROUEN

Architecture Logicielle – TP 05 2016–2017

Le patron *Passerelle*

Exercice 1 –

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 format : *HTML* et *PDF*. L'interface en question se présente comme ceci :

```
1  public interface FormGenerator {
2    public void addTitleHeader(String formTitle); // Titre centre
3    public void addSeparator();       // Ligne de separation
4    public void addNameLine();        // Nom : __________
5    public void addFirstNameLine();   // Prenom : __________
6    public void addEmailLine();       // Courriel : __________
7    public String send();             // Retourne le flux du formulaire construit
8  }
```

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 champs de formulaire consistant en le nom du champs 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 *Formulaire* qui contient le code, commun à tous les formats, de la méthode *addTitleHeader*. Deux classes concrètes, *FormulaireHTML* et *FormulairePDF* ont été dérivées de la classe *Formulaire* pour implémenter les deux formats.

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

```
1
2  public class TestForm {
3    public FormGenerator getFormGenerator() {
4      // Obscure fonction retournant un generateur de formulaire.
5      // Vous la definirez pour vos tests.
6      // Ex: return new HTMLForm();
7    }
8
9    public testGenerator(FormGenerator fg) {
10     fg.addTitleHeader("URouen Corp.");
11     fg.addSeparator();
12     fg.addNameLine();
13     fg.addFirstNameLine();
14     fg.addSeparator();
15     fg.addEmailLine();
16
17     String Result = fg.send();
18     System.out.println(Result);
19   }
20
21   public static void main(String[] args) {
22     testGenerator(getFormGenerator());
23   }
24 }
```

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

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 *FormulaireEnHTML* et *FormulaireEnPDF*.
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 abstraction qui permettra de séparer les aspects linguistique et format. Donnez le diagramme UML de cette solution, puis réalisez son implémentation.
Question 1.3 : Implémentez la solution validée.

Exercice 2 –
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 différent types de logger : un pour les avertissement, un pour les erreurs, faisant automatique 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.