TELECOM SudParis

Exemple de questions en Spécification, Conception et Programmation

Les seuls documents autorisés sont ceux distribués en cours et en TP, et mis à disposition sur le site Web du module, ainsi que vos notes personnelles.

Notes :

- les questions qui suivent constituent un entraînement sur les parties « spécification », « conception » et « programmation » pour le contrôle final de la deuxième session du module CSC4102. Elles sont extraites d'un ancien sujet. Dans le cadre du module CSC4102, pour couvrir l'ensemble du programme, nous y ajouterions une question sur la « préparation des tests », une question sur la « qualité du modèle » (par exemple sur les patrons de conception), et une question sur les « idiomes JAVA » (par exemple sur les méthodes `equals` et `hashCode`) ;
- la durée estimée pour répondre à toutes les questions qui suivent est 1h30 ;
- un barème est donné à titre indicatif ;
- soyez concis et précis, et justifiez vos réponses par des commentaires appropriés ;
- soyez rigoureux dans la syntaxe UML ;
- soyez rigoureux dans la syntaxe `JAVA` ;
- **veillez à rendre une copie propre et lisible, avec une marge à gauche.**

1 Sujet

Étude de cas. Nous souhaitons mettre en œuvre un système de gestion de journaux d'événements (en anglais, *logs*) utilisé pendant l'exécution de programmes écrits en Java. Le principe d'un tel système est d'éviter d'écrire des instructions println en les remplaçant par des instructions du type « enregistrer ce message dans tel fichier en y ajoutant l'heure courante » ou « afficher ce message à la console avec telle couleur », ou encore « enregister ce message dans tel fichier et l'afficher à l'écran ». Dans le jargon de la programmation, la partie de formattage du message est communément nommée un *beautifier*. La figure 1 présente seulement une partie de ce système : la partie formattant le message, c'est-à-dire permettant de colorier le message, de mettre les caractères en majuscule, d'ajouter l'heure, etc. Pour information, la figure 1 utilise le patron de conception « Décorateur ».

Les premières questions vous demandent de compléter le diagramme de la figure 1 pour modéliser toute la partie statique du système. Les questions qui suivent vous demandent de mettre en œuvre dans des classes Java la partie modélisée dans la figure 1. La dernière question vous demande de compléter le diagramme de séquence de la figure 2.

L'objectif du système est de permettre à un programmeur en Java de créer des journaux. Un journal est associé à des *beautifieurs*. Généralement, le programmeur définit des *beautifieurs* en début de programme et les réutilise pour ses différents journaux. L'opération beautify consiste à transformer l'objet sous la forme d'une chaîne de caractères (avec la méthode toString) et à décorer cette chaîne de caractères. Voici quelques informations sur les méthodes beautify des *beautifieurs* présentés dans la figure 1 :

- la méthode beautify de la classe EmptyBeautifier ne fait rien de plus que vérifier que la référence o n'est pas nulle avant d'appeler la méthode toString ;
- la même méthode de la classe ColorDecorate concatène les chaînes de caractères suivantes : la couleur (p.ex. l'attribut BLUE), l'appel à la méthode toString sur l'argument, et la chaîne de caractères de l'attribut RESET (pour revenir à la couleur initiale) ;
- la méthode de la classe UpperCaseBeautifier transforme les caractères minuscules en caractères majuscules. En Java, une chaîne de caractères s est transformée en majuscule avec l'instruction s.toUpperCase() ;
- la méthode beautify de la dernière classe BeforeAfterDecorator complète le message avec un préfixe (attribut contentBefore) et un suffixe (attribut contentAfter). Notez que les préfixes et les suffixes ont chacun leur *beautifieur* ; ils donc « *beautifiés* » pendant l'appel à la méthode beautify.

Le système complet permet de gérer plusieurs journaux. Chaque journal est aussi associé à une configuration de journalisation. Une configuration de journalisation indique le niveau de gravité (un entier) accepté par le journal : par exemple, un message de niveau debug ne sera pas journalisé si le niveau spécifié dans la configuration est fatal (c'est-à-dire uniquement les erreurs fatales qui arrêtent le programme). Le programmeur définit des configurations de journalisation et les réutilise pour ses différents journaux.

Un journal est enfin associé à plusieurs flots de sortie : p.ex. pour afficher les messages à la fois dans la console et dans un fichier ; mais, un flot de sortie n'est utilisé que par un journal. Nous considérons quatre catégories de flots de sortie : console (comme println), fichier (journalise dans un fichier de texte, un message par ligne), tube nommé (envoie le message « *beautifié* » à un autre processus ou *thread* local), et *socket* (envoie le message « *beautifié* » à un processus qui potentiellement s'exécute sur une autre machine.

2 Questions

Question 1 (3,5pt) Complétez le diagramme de classes de la figure 1 pour modéliser tout le système de gestion de journaux. N'y mettez pas les attributs.

Vous pouvez rendre l'énoncé avec les éléments que vous ajoutez à la figure 1, mais n'oubliez pas de mettre votre nom sur la feuille.

Question 2 (1 pt) Écrivez le code de Beautifier.

Question 3 (2,5 pts) Écrivez le code de la classe UpperCaseBeautifier.

Question 4 (5 pts) Écrivez le code de la classe ColorDecorator. Vous devez remplir la collection colors dans le constructeur avec les couleurs définies en attribut de classe (BLUE, etc.). Vous devez refuser de créer un objet de cette classe si color passé en paramètre n'est pas dans la collection colors en levant une exception de type IllegalArgumentException (nous supposons que la classe IllegalArgumentException existe).

Question 5 (2 pts) Écrivez en utilisant `JUnit` le code de la classe JUnitColorDecorator qui contient une méthode qui test le cas d'erreur de la question précédente.

Question 6 (3,5 pts) Écrivez le code de la classe BeforeAfterDecorator.

Question 7 (3,5pt) Complétez le diagramme de séquence de la figure 2 pour modéliser les interactions correspondant aux lignes 24–27 de la méthode main de la figure 3. Ignorez les objets calendar et dateFormat. Par ailleurs, l'obtention de l'heure courante est obtenue par l'instruction dateFormat.format(calendar.getTime()).

Vous pouvez rendre l'énoncé avec les éléments que vous ajoutez à la figure 2, mais n'oubliez pas de mettre votre nom sur la feuille.

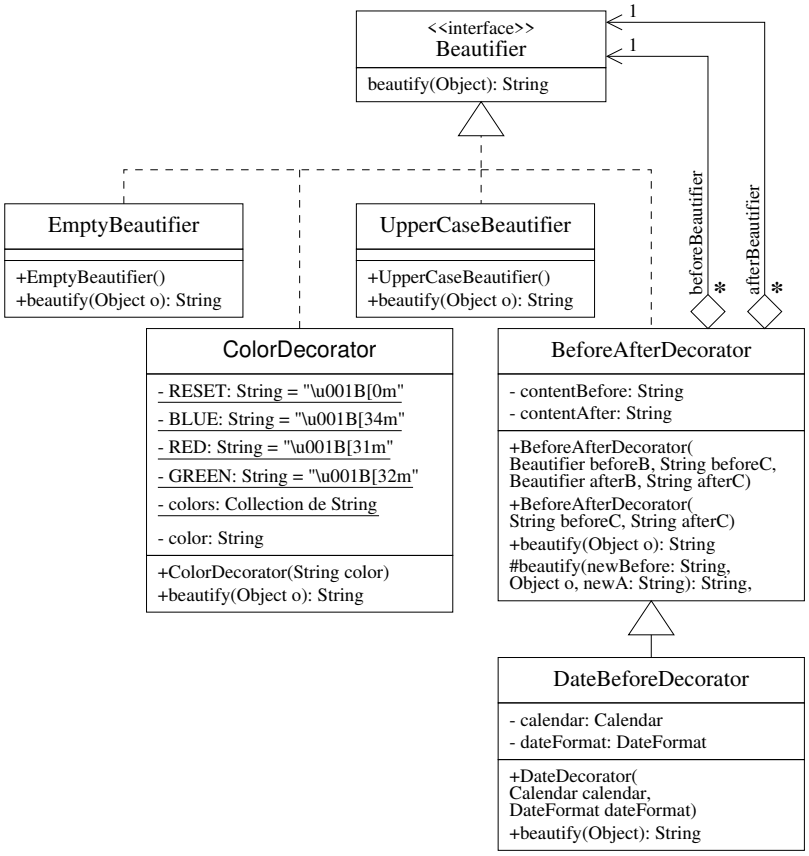
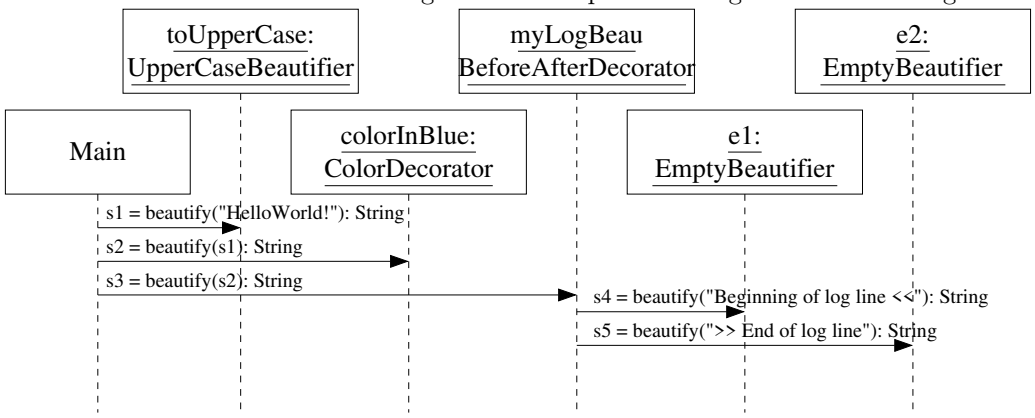


FIGURE 1 – Diagramme de classes des *beautifieurs*.

FIGURE 2 – Diagramme de séquence des lignes 18–19 de la figure 3.




```
1  import java.text.DateFormat;
2  import java.util.Calendar;
3  import java.util.GregorianCalendar;
4  import java.util.Locale;
5
6  import beautifier.Beautifier;
7  import beautifier.BeforeAfterDecorator;
8  import beautifier.ColorDecorator;
9  import beautifier.DateBeforeDecorator;
10 import beautifier.UpperCaseBeautifier;
11
12 public class Main {
13         public static void main(String[] args) {
14                 Beautifier colorInBlue = new ColorDecorator(ColorDecorator.ANSI_BLUE);
15                 Beautifier toUpperCase = new UpperCaseBeautifier();
16                 Beautifier myLogBeau = new BeforeAfterDecorator(
17                                 "Beginning of log line <<", ">> End of log line");
18                 String myLogLine = myLogBeau.beautify(
19                                 colorInBlue.beautify(toUpperCase.beautify("HelloWorld!")));
20                 System.out.println(myLogLine);
21                 Calendar calendar = new GregorianCalendar();
22                 DateFormat dateFormat = DateFormat.getDateTimeInstance(
23                                 DateFormat.MEDIUM, DateFormat.MEDIUM, new Locale("FR", "fr"));
24                 Beautifier colorInRed = new ColorDecorator(ColorDecorator.ANSI_RED);
25                 Beautifier addDate = new DateBeforeDecorator(colorInRed, calendar,
26                                 dateFormat);
27                 myLogLine = addDate.beautify(toUpperCase.beautify("HelloWorld!"));
28                 System.out.println(myLogLine);
29         }
30 }
```

FIGURE 3 – Exemple de méthode main utilisant les *beautifieurs*.