



Corrigé Barème

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 « enregistrer ce message dans tel fichier et l'afficher à l'écran ». Dans le jargon de la programmation, la partie de formatage du message est communément nommée un *beautififier*. 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.

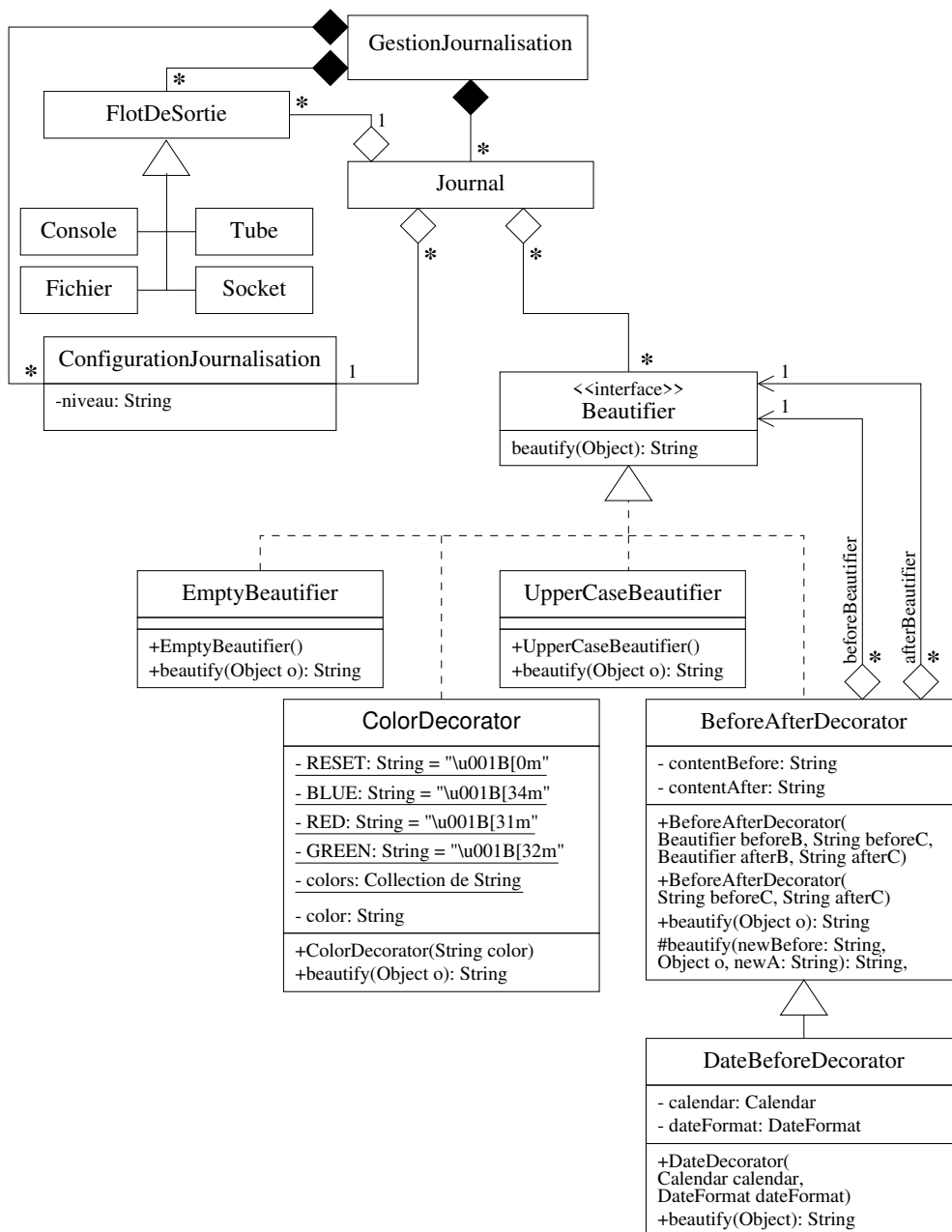
2 Questions

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.

Réponse:

La figure suivante présente le diagramme de classes du système complet.



Erreurs ou faiblesses fréquemment observées :

- composition de Journal vers ConfigurationJournalisation ou Beautifier
- multiplicité 1..1 entre ConfigurationJournalisation ou Beautifier et Journal

Barème de correction sur 3,5 point(s) :

Répartition :

0,5 qualité de l'UML

0,5 classe façade

0,5 classes Journal, FlotDeSortie et ConfigurationJournalisation

0,5 associations autour de Journal

0,5 généralisation spécialisation FlotDeSortie

0,5 multiplicité entre FlotDeSortie et Journal

0,5 multiplicité entre ConfigurationJournalisation ou Beautifier et Journal

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

Réponse:

```
package beautifier;

public interface Beautifier {
    String beautify(Object o);
}
```

Erreurs fréquemment observées :

- pas une interface
- méthode avec une définition

Barème de correction sur 1 point(s) :

Répartition :

0,5 interface

0,5 méthode sans définition

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

Réponse:

```
package beautifier;

import java.util.Locale;

public class UpperCaseBeautifier implements Beautifier {
    public UpperCaseBeautifier() {
    }
    @Override
    public String beautify(Object o) {
        return (o == null) ? "" : o.toString().toUpperCase(Locale.ROOT);
    }
}
```

```

    }
}

```

Erreurs fréquemment observées :**Barème de correction sur 2 point(s) :**

Répartition :

0,5 implements

0,5 constructeur vide ou pas de constructeur

0,5 test non null

0,5 appel à toString

0,5 return

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).

Réponse:

```

package beautifier;

import java.util.List;
import java.util.Vector;

public class ColorDecorator implements Beautifier {
    public static final String ANSI_RESET = "\u001B[0m";
    public static final String ANSI_BLACK = "\u001B[30m";
    public static final String ANSI_RED = "\u001B[31m";
    public static final String ANSI_GREEN = "\u001B[32m";
    public static final String ANSI_YELLOW = "\u001B[33m";
    public static final String ANSI_BLUE = "\u001B[34m";
    public static final String ANSI_PURPLE = "\u001B[35m";
    public static final String ANSI_CYAN = "\u001B[36m";
    public static final String ANSI_WHITE = "\u001B[37m";
    private static List<String> colors;
    private String color;

    public ColorDecorator(final String color) throws IllegalArgumentException {
        if (colors == null) {
            colors = new Vector<String>();
            colors.add(ANSI_RESET);
            colors.add(ANSI_BLACK);
            colors.add(ANSI_RED);
            colors.add(ANSI_GREEN);
            colors.add(ANSI_YELLOW);
            colors.add(ANSI_BLUE);
            colors.add(ANSI_PURPLE);
            colors.add(ANSI_CYAN);
            colors.add(ANSI_WHITE);
        }
        if (colors.contains(color)) {

```

```
                this.color = color;
            } else {
                throw new IllegalArgumentException("Unknown color");
            }
        }

        @Override
        public String beautify(Object o) {
            return color + o + ANSI_RESET;
        }
    }
}
```

Erreurs fréquemment observées :

Barème de correction sur 5 point(s) :

Répartition :

- 0,5 attributs privés
- 0,5 attributs de classe
- 0,5 attribut List<String>
- 0,5 vecteur + initialisation du vecteur
- 0,5 remplissage du vecteur
- 0,5 non remplissage si déjà rempli
- 0,5 test présence dans vecteur
- 0,5 levée d'exception
- 0,5 déclaration de levée d'exception
- 0,5 beautify avec concaténation de chaînes de caractères

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.

Réponse:

```
package beautifier.tests;

import org.junit.Test;

import beautifier.ColorDecorator;

public class JUnitColorDecorator {

    @Test(expected=IllegalArgumentException.class)
    public void testConstructorUnknownColor() {
        new ColorDecorator("unknown");
    }
}
```

Erreurs fréquemment observées :

Barème de correction sur 2 point(s) :

Répartition :

0,5 @Test
0,5 expected
0,5 appel constructeur
0,5 valeur pour levée d'exception

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

Réponse:

```
package beautifier;

public class BeforeAfterDecorator implements Beautifier {
    private Beautifier beforeBeautifier;
    private String contentBefore;
    private Beautifier afterBeautifier;
    private String contentAfter;

    public BeforeAfterDecorator(final Beautifier beforeBeautifier,
                                final String stringBefore, final Beautifier afterBeautifier,
                                final String stringAfter) {
        this.beforeBeautifier = beforeBeautifier;
        this.contentBefore = stringBefore;
        this.afterBeautifier = afterBeautifier;
        this.contentAfter = stringAfter;
    }

    public BeforeAfterDecorator(final String stringBefore,
                                final String stringAfter) {
        this.beforeBeautifier = new EmptyBeautifier();
        this.contentBefore = stringBefore;
        this.afterBeautifier = new EmptyBeautifier();
        this.contentAfter = stringAfter;
    }

    @Override
    public String beautify(final Object o) {
        return beforeBeautifier.beautify(contentBefore) + " " + o
            + " " + afterBeautifier.beautify(contentAfter);
    }

    protected String beautify(final String newContentBefore, final Object o,
                                final String newContentAfter) {
        String before = (newContentBefore == null) ? contentBefore
            : newContentBefore;
        String after = (newContentAfter == null) ? contentAfter
            : newContentAfter;
        return beforeBeautifier.beautify(before) + " " + o + " "
            + afterBeautifier.beautify(after);
    }
}
```

Erreurs fréquemment observées :

Barème de correction sur 3,5 point(s) :

Répartition :

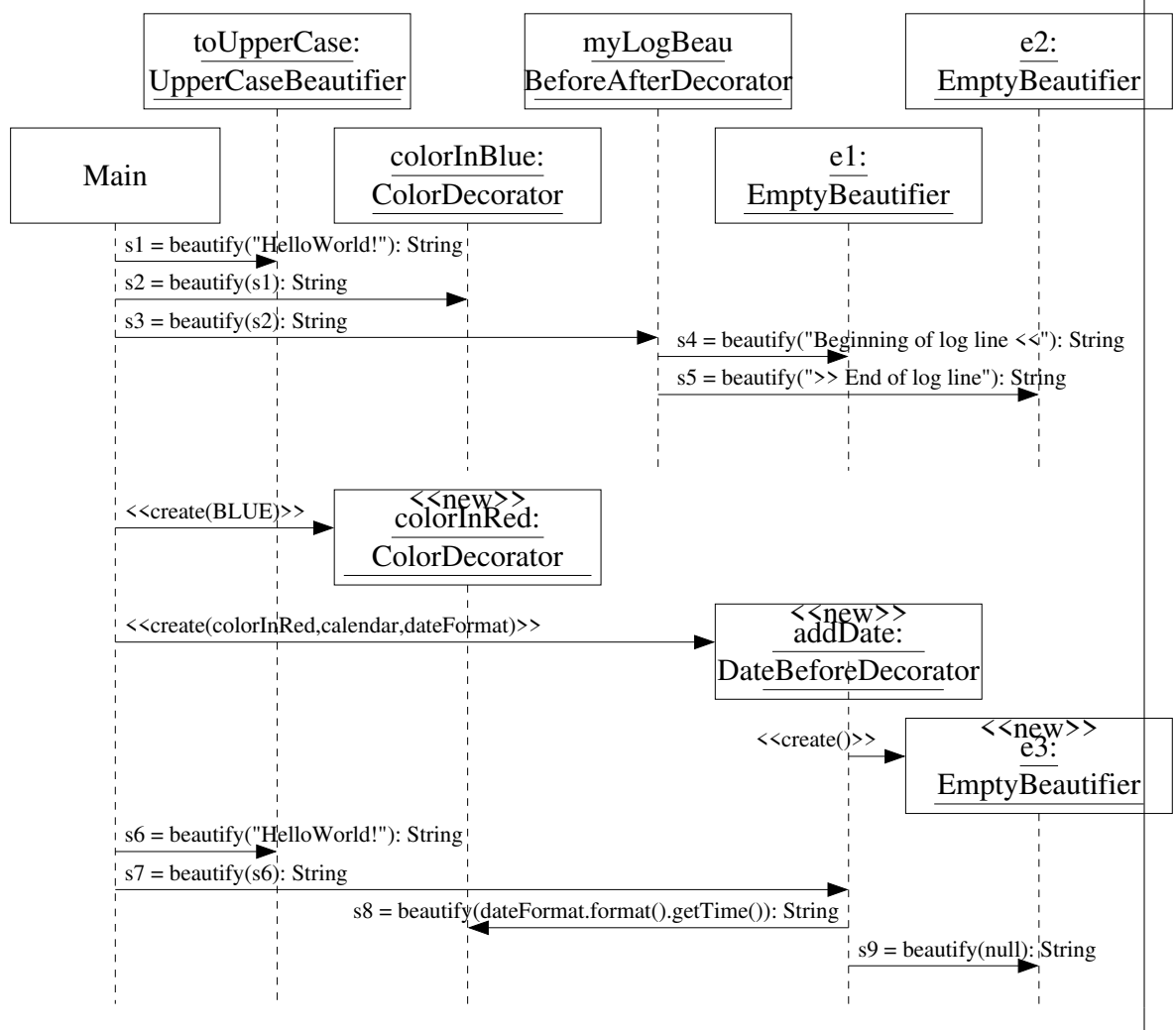
- 0,5 constructeur avec beautifieurs
- 0,5 constructeur sans beautifieur (utilisation de EmptyBeautfier)
- 0,5 beautify avec un argument
- 0,5 beautify avec trois arguments
- 0,5 méthodes publiques
- 0,5 méthode protégée

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.

Réponse:

La figure suivante présente le diagramme de séquence des lignes~24--27.



Erreurs ou faiblesses fréquemment observées :

Barème de correction sur 3 point(s) :

Répartition :

0,5 qualité de l'UML

0,5 création objet colorInRed

0,5 new

0,5 création objet addDate

0,5 beautify sur UpperCaseBeautifier

0,5 beautify sur addDate

0,5 beautify imbriqué sur colorInRed

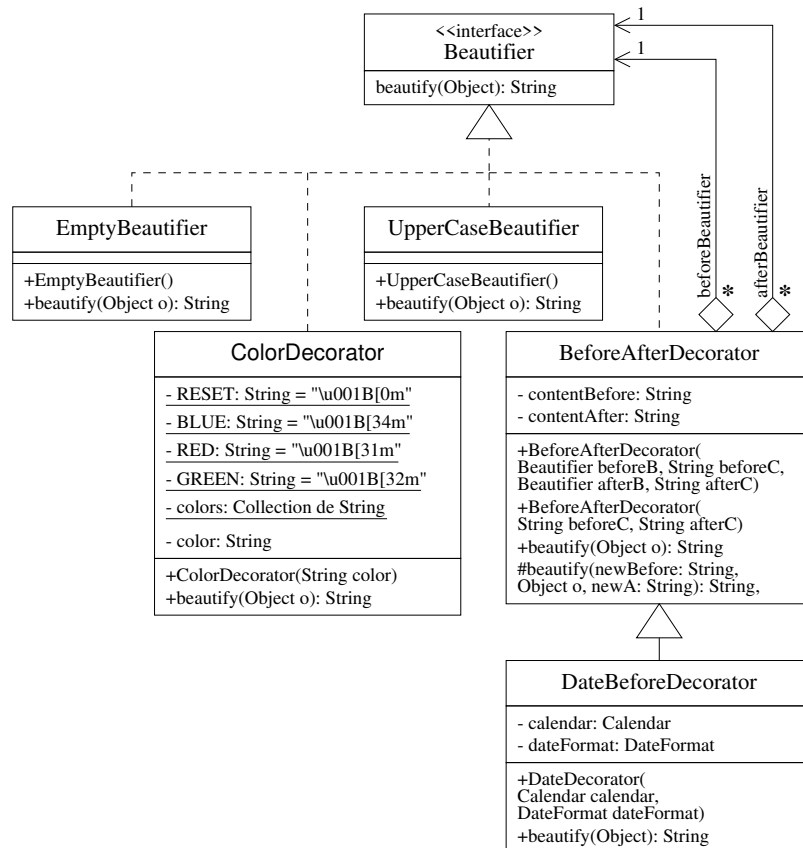
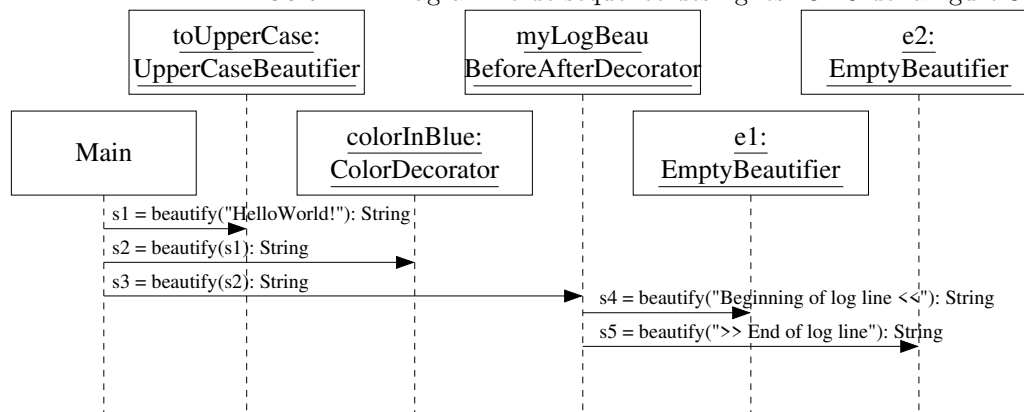


FIGURE 1 – Diagramme de classes des *beautifieurs*.

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.getDateInstance(
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*.