Le patron *Singleton*

Exercice 1 – Codage d'un Logger

Soit le code suivant (`al-tp6-fichiers/Appli.java`) :

```java
import myLib.*;

public class Appli {
  private LibStuff stuff;

  public void initApp() {
    // Info:"Initializing App..."
    // ... some code here
    stuff= new LibStuff();
    // Info:"Initialization done!"
  }

  public void closeApp() {
    // Info:"Closing app..."
    // ... some code here
    // Info:"Ressources freeed!"
  }

  public void run() {
    boolean errors=false;
    for(int i=0; i<10; ++i) {
      int action = (int)(100.0*Math.random());
      if (action<50) {
        stuff.doSomething(action);
      } else {
        // Error:"Invalide action : "+action
        errors=true;
      }
    }
    if (errors) {
      // Warning : some errors occured during processing.
    }
  }

  public static void main(String[] args) {
    Appli app=new Appli();
    app.initApp();
    app.run();
    app.closeApp();
  }
}
```

On souhaite pouvoir journaliser les différents aspects du cycle de vie de cette application à des fins de déverminage (debugging). Pour cela, on souhaite utiliser un objet, représentant un journal des évènements (*log*), sur lequel on pourra effectuer les actions suivantes :

— émettre un message d'information (*INFO*),
— émettre un message d'avertissement (*WARNING*),
— émettre un message d'erreur fatale (*ERROR*),

Chacune de ces actions sera réalisée par une méthode spécifique prenant en argument le message à émettre sous forme d'une chaîne de caractères.
On souhaite en outre que les messages soient préfixés par leur numéro (ordre d'émission), leur type (*INFO* *WARNING* ou *ERROR*), et envoyés dans la console d'exécution (`System.out`).
Enfin, on note que les points d'émission et leur type sont marqués dans le code source sous forme de commentaires et que la bibliothèque `mylib/LibStuff.java` doit également faire l'objet d'émissions de messages.
Question 1.1 : Quelle solution proposez-vous de mettre en place ? Donnez son diagramme UML.
Le diagramme UML ne rend pas compte des détails d'implémentation du patron.
Question 1.2 : Qualifiez complètement l'implémentation de la solution que vous allez mettre en place, justifiez les et proposez des **tests logiciels** permettant de valider toutes ces caractéristiques.

Exercice 2 – *Évolution du Logger*

On reprend l'exercice précédent et on se propose de fournir un deuxième type journal d'événements qui enverra cette fois les message dans un fichier portant le même nom que l'application [1] avec l'extension `.log`. De plus, pour ce type de logger, les messages ne seront plus numérotés.
Les deux loggers seront disponibles dans l'application et l'utilisateur pourra activer le second type en ajoutant le paramètre `filelog` sur la ligne de commande `java Appli filelog`, sinon c'est le premier logger qui sera activé.
Enfin, on souhaite ne pas avoir à modifier l'application cliente obtenue à l'exercice 1, mis à part l'initialisation.
Question 2.1 : Quelle solution allez-vous mettre en place ? Donnez son diagramme UML complet et détaillez son implémentation, en particulier l'initialisation.
Question 2.2 : Les modifications apportées violent-elle des principes ?
Question 2.3 : Ajoutez des tests logiciels permettant de valider son fonctionnement.

1. Astuce : on pourra utiliser la propriété système `sun.java.command`.