Synthèse du Travail Pratique : Pattern Observateur et Timer Service

Rapport sur les étapes et la résolution des problèmes de concurrence

Novembre 2025

1 Introduction

Ce rapport résume les étapes du Travail Pratique visant à mettre en œuvre le pattern **Observateur (Observer)** en Java, autour de l'interface `TimerService`. Il détaille l'implémentation des clients, l'identification d'un bogue de concurrence et la solution adoptée via `PropertyChangeSupport`.

2 Étape (c) : Implémentation de la Classe Horloge

L'objectif était de créer une horloge (`Horloge`) affichant l'heure à chaque seconde, tout en respectant le principe d'abstraction (dépendance uniquement de l'interface `TimerService`).

- La classe `Horloge` dépend de l'interface `TimerService` (par injection dans le constructeur), et non de l'implémentation concrète (Principe d'Abstraction).
- `Horloge` implémente l'interface `TimerChangeListener` et s'enregistre via `timerService.addTimeChangeListener(` (Observation).
- La méthode `propertyChange` est appelée et affiche l'heure lorsque la propriété des secondes (`SECONDE_PROP`) change (Réaction).

3 Étape (d) : Introduction du Compte à Rebours

La classe `CompteARebours` a introduit une complexité en se désinscrivant dynamiquement du `TimerService` lorsque son compteur atteint zéro.

3.1 Problème : Le Bogue de Modification Concurrente

L'instanciation de multiples observateurs dynamiques (10 `CompteARebours` avec des valeurs aléatoires) engendrait des bogues lors de l'exécution (souvent une `ConcurrentModificationException`).

3.1.1 Explication

Le bogue était causé par la modification de la liste des auditeurs (`listeners`) au sein de `DummyTimeServiceImpl` alors qu'elle était en cours d'itération pour la notification des changements.

- `DummyTimeServiceImpl` itère sur `listeners` pour appeler `propertyChange` sur chaque client.
- Un client (`CompteARebours`) atteint zéro et appelle `removeTimeChangeListener(this)`.
- La modification de la liste pendant l'itération active est détectée et lève une exception.

4 Étape (e) : Résolution avec `PropertyChangeSupport`

La solution standard en Java pour ce type de problème est de déléguer la gestion des observateurs à la classe `java.beans.PropertyChangeSupport`.

4.1 Changements Structurels

1. **Contrat d'Interface :** `TimerChangeListener` a été modifié pour hériter de `java.beans.PropertyChangeListe`
2. **DummyTimeServiceImpl :**
   — L'ancienne `List` des auditeurs a été remplacée par un champ `PropertyChangeSupport` que nous désignons par `listeners` dans nos explications.
   — Les méthodes `add/removeTimeChangeListener` et les notifications (`firePropertyChange`) utilisent ce support.
3. **Clients (Horloge, CompteARebours) :** La signature de la méthode d'écoute a été mise à jour : `public void propertyChange(PropertyChangeEvent evt)`.

4.2 Résolution du Bogue de Démarrage (`NPE`)

Une `NullPointerException` est survenue au démarrage car l'appel à `setTimeValues()` (qui déclenche les notifications) se produisait avant l'initialisation de l'objet `listeners` (`PropertyChangeSupport`).

La résolution consiste à garantir l'initialisation de l'objet `listeners` avant tout usage. La méthode la plus fiable en Java est l'initialisation immédiate du champ 'final'.

```
// La résolution: Initialisation immédiate du champ 'listeners'
// (l'objet PropertyChangeSupport) pour garantir sa disponibilité.
private final PropertyChangeSupport listeners = new PropertyChangeSupport(this);
// Dans le constructeur, setTimeValues() est appelé APRES que listeners soit initialisé.
```

4.3 Conclusion sur la Résolution Définitive

`PropertyChangeSupport` résout le bogue de concurrence. Il gère les ajouts et suppressions de listeners de manière sécurisée, assurant que les modifications intervenant pendant la notification n'interfèrent pas avec la boucle de distribution des événements.

5 Étape (f) - Bonus : Application Graphique (GUI)

La dernière étape consistait à créer une application permettant d'afficher l'heure sur une interface graphique en utilisant le `TimerService`.

5.1 Implémentation de l'Horloge Graphique

Nous avons utilisé Java Swing pour créer une fenêtre simple contenant un `JLabel` pour afficher l'heure. La classe `HorlogeGUI` agit comme l'Observateur du `TimerService` et met à jour l'interface utilisateur à chaque changement de seconde.

— La classe `HorlogeGUI` implémente `TimerChangeListener`.
— L'affichage est mis à jour en toute sécurité sur le thread de l'interface utilisateur grâce à `SwingUtilities.invokeL`
— La connexion au `TimerService` est réalisée via l'injection de dépendances au constructeur, maintenant ainsi l'architecture découplée.