Génie Logiciel - EMP - 2020/2021

TP 1

Dans ce TP, nous allons continuer dans avec la montre intelligente que nous avons vaguement entamé dans le TP précédent.

Objectifs

- Visualiser l'un des problèmes d'utilisation une implémentation basique de l'observable.
- Remédier aux problèmes par la simple utilisation de la délégation.
- Séparer l'implémentation des interfaces graphiques et des modèles métier.
- Utilisation du Patron de conception "**État**"

Objectifs supplémentaires

- Manipulation basique du gestionnaire de sources "**git**"

Activité 1.1 Test d'une implémentation basique du *TimerService*

Test d'un code initiale

Vous allez commencer à travailler sur la base d'un code qui vous est fourni. Pour récupérer le code, utiliser l'outil de gestion de source ***git*** avec les commandes[1]:

```
mkdir tp-2
cd tp-2
git clone https://github.com/sweeffon2/tp-gl.git
```

Vous remarquerez l'apparution d'un dossier **tp-gl**, qui représente le projet parent, avec plusieurs modules à l'intérieur (voir figure 1).

- le module **timer-service** ne contient que des abstractions
- le module **launcher** est le lanceur de notre logiciel, il va rassembler l'ensemble des modules pour les faire tourner ensembles. Donc C'est normale qu'il dépend de tous les autres modules!
- remarquez que l'interface *TimerService* hérite de l'interface *TimerChangeProvider*. Ce qui veut dire, que la classe qui va implémenter *TimerService* va implicitement implémenter *TimerChangeProvider*.
- dans le module **dummy-timer-service-impl**, vous trouverez la classe *DummyTimerServiceImple* qui nous offre une implémentation initiale du *TimerService*. Dans cette implémentation nous utilisons une liste pour tenir les observateurs.

Compilez, et exécuter

a) Positionnez vous dans le répertoire du projet parent

b) Construisez le projet en utilisant la commande :

```
mvn install
```

c) exécuter le projet en utilisant la commande

```
java -jar launcher/target/launcher-0.0.1-jar-with-dependencies.jar
```

Observez le code et expliquez ce qui ce passe lors de l'exécution ?

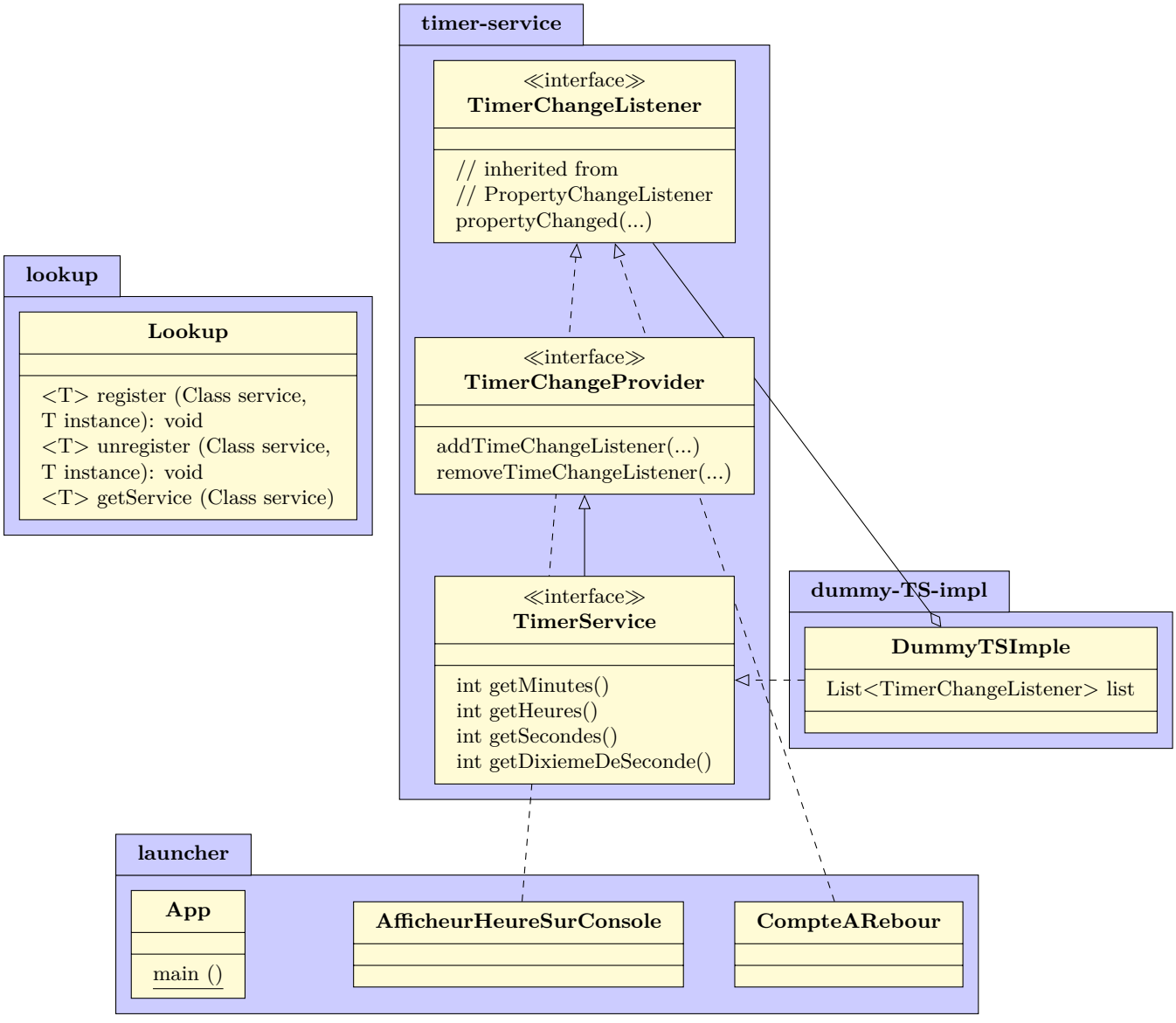

Figure 1: Diagramme de classe du projet initiale.

Maintenant Commentez la ligne n° 30 de la classe *App* de votre code, et activez la ligne n° 31. Que s'est t'il passé !

A présent, entourez la ligne 31 d'une boucle pour la répéter pendant 20 fois. Que constatez vous ? et pourquoi ?

[1]Vous pouvez aussi aller directement sur la page du lien et le télécharger sous format zip

Pour éviter ce problème, Nous allons créer un nouveau module: **timer-service-impl-withdelegation** contenant une classe: *TimerServiceImplWithDelegation*, qui va déléguer la gestion des observateur à un objet de Type *PropertyChangeSupport*.

Codez le, ajoutez les dépendances nécessaires dans le module **launcher**. Et enregistrez une instance de la nouvelle classe obtenu dans le Lookup (Ligne 17 de la classe *App*).

Recompilez, Exécutez et Appréciez l'exécution.

Activité 1.2 Ajout des interfaces graphiques

Nous disposons, à présent, d'une implémentation correcte du *TimerService*. Nous allons maintenant réaliser des interfaces graphiques, Comme suit:

a) Une interface graphique permettant d'afficher l'heure

b) Une interface graphique contenant deux actionneurs (le cas échéant des boutons) pouvant être utilisés pour le réglage de l'heure (réglage des secondes, des minutes, des heures). L'un des boutons (***MODE***) permettra de sélectionnez la propriété à changer, l'autre: ***INCREMENT*** permettra d'incrémenter la valeur de la propriété sélectionnée.
Astuce: Utilisez le patron d'état.

c) les deux interfaces graphiques doivent être faiblement dépendante. (pensez à créer des modules d'abstraction et des modules d'implémentation)

NB. Pensez à créer un module: **Model**, dans lequel vous implémentez une classe: *Gestionnaire de montre*, qui permet de centraliser les commandes.

A propos de ce TP

Travaillez en groupe, et réfléchissez à une solution commune. Ce TP est à Rendre à la dernière séance de Cours (avant l'examen Partiel)