



LaboratoireGEN No III

Compléments Objets actifs Synchronisation des horloges de gares

HES-SO – Mars 2017 - ELS

Objectifs

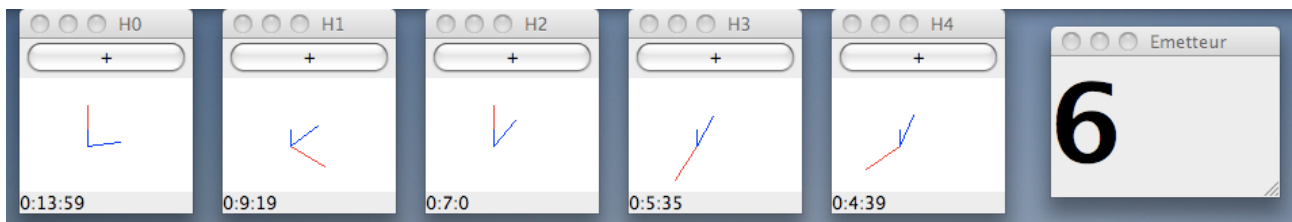
- Utilisation des mécanismes de synchronisation offerts par Java dans le cadre de la gestion d'applications multitâches. Notamment, seront utilisées les méthodes de synchronisation «`wait()`» et «`notifyAll()`».
- Re-structuration d'un programme existant sur la base du pattern **MVC**
- Utilisation du pattern **Observable-Observé**

Application à réaliser

Simulation de la synchronisation des horloges de gare.

Vous aurez sans doute remarqué les nombreuses horloges que les gares de suisse romande mettent à la disposition des usagers et du personnel d'exploitation. Comme ces horloges disposent à priori de leur propre mécanisme de mise à jour, nous pouvons nous attendre à certaines dérives: des horloges avancent, alors que d'autres retardent.

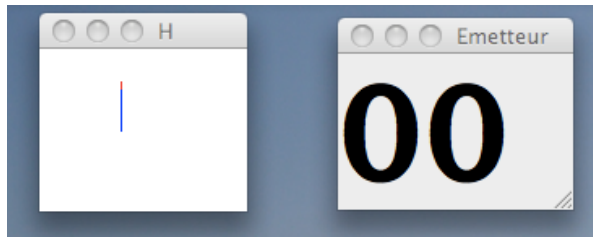
La bonne gestion du système impose toutefois que toutes les horloges du réseau ferroviaire soient synchronisées (à quelques secondes près). A cette fin, le réseau dispose d'un émetteur, - que l'on assimilera à l'horloge « juste » -, qui envoie un signal de synchronisation toutes les minutes exactement. Toutes les horloges sont supposées « écouter » ce signal ...



Etape 1

Copiez les fichiers du programme «Horloges» dans un dossier personnel, créez un projet avec votre IDE préféré et testez ce programme.

Vous pourrez constater que ce programme met en œuvre un émetteur et une horloge (une seule). En l'état, ni l'horloge ni l'émetteur ne « tournent ».

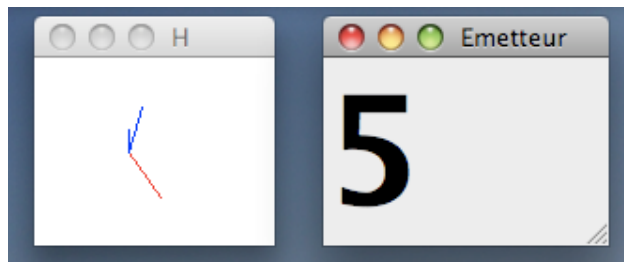


Par ailleurs, l'horloge et l'émetteur sont complètement découplés.

Le but de l'étape No1 est de vous familiariser avec le contenu de ce programme.



Au terme de cette étape l'horloge et l'émetteur doivent « tourner » de manière concurrente (→ objets actifs) tout en restant totalement découplés (l'horloge n'est pas synchronisée sur l'émetteur).



L'émetteur doit exécuter le code suivant toutes les « `dureeSeconde` » msec:

```
heureMettreAJour();
```

L'attente `dureeSecondes` doit s'effectuer en utilisant `Thread.sleep(msec)`

L'horloge doit exécuter le code suivant toutes les « `dureeSeconde` » msec:

```
incrementerSecondes();  
toile.repaint();
```

Etape 2

Refactoring!!

Dans son état actuel, le programme est mal structuré.



Transformez-le en adoptant le modèle MVC.

- L'amorce jouera le rôle de contrôleur général, dont le rôle se limitera à la création des objets.
- Emetteur et Pendule seront deux modèles qui seront observés chacun par leur vue respective. Deux nouvelles classes à créer **VueEmetteur** et **VuePendule**.

Note : Les deux « modèles » Emetteur et Pendule seront en fait deux « modèle-contrôleur » : jouant à la fois de modèle et de contrôleur pour l'émetteur, respectivement pour la pendule.

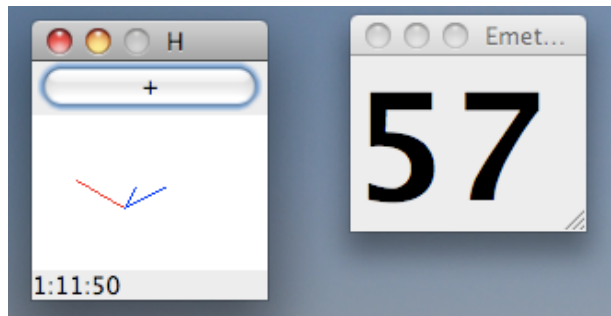
- Les vues de votre programme doivent être placées dans un paquetage « **views** »
- Les modèles de votre programme doivent être placés dans un paquetage « **models** »
- La mise en œuvre du modèle Observable-Observé s'appuiera sur la classe `Observable` et l'interface `Observer` de Java



Les deux modèles doivent être complètement nettoyés de tout aspect visuel.

Etape 3

Améliorer la vue du pendule



Complétez la classe `ToileGraphique` de manière à ce que :

1. L'heure soit indiquée en mode alphanumérique (`hh:mm:ss`) dans la partie « South » de la fenêtre.
2. Faire en sorte qu'un bouton « + », placé dans la partie « North » de la fenêtre soit opérationnel. Chaque pression du bouton aura pour effet d'incrémenter les minutes (sans toucher aux secondes)

Etape 4

Il s'agit maintenant de passer aux choses sérieuses et d'opérer la synchronisation effective des horloges.

Point 1 Plusieurs horloges

Dans « l'amorce » du programme, créez un certain nombre d'horloges supplémentaires.

Chaque horloge doit disposer de son propre label (titre de la fenêtre), comme par exemple «H0», «H1», ..., «H4».

Chaque horloge doit disposer d'une « valeur de la seconde » en msec qui lui sera personnelle. Par exemple, l'horloge «H0» peut avoir une valeur de 800 msec (elle avancera ainsi de plus de 10 secondes par minute), une autre pourra avoir une valeur de 1200 msec : cette dernière retardera d'autant.

Faites en sorte d'avoir quelques horloges « de référence » caractérisées par une valeur de seconde « juste », identique à celle de l'émetteur.

Point 2 Synchronisation

Pour se simplifier la vie, nous supposerons dans cette étape que toutes les horloges sont réglées mécaniquement de manière à tourner **toujours plus rapidement** que l'émetteur. Le mécanisme de synchronisation que vous mettrez en œuvre s'appuiera sur cette hypothèse.

Si d'aventure une horloge tournait plus lentement (défaut de construction), le mécanisme élaboré pour les autres ne doit pas « planter » à cause de cette horloge défectueuse.

Etape 5

Essayez maintenant d'élaborer un mécanisme qui soit capable de synchroniser à la fois les deux types d'horloges : celles qui tournent plus lentement, et celles qui tournent plus rapidement que l'émetteur.

Etape 6 – Contrôle du laboratoire

A la présentation de votre laboratoire, vous devrez être capable de présenter dans votre code:

1. Quelles sont les ressources critiques
2. Quels sont les threads qui manipulent simultanément ces différentes ressources critiques.



Contraintes de conception

1. Vous devez vous appuyer sur le support de cours « Threads avec Java »
2. Les horloges et l'émetteur doivent être couplés en utilisant le modèle **Observable-Observé**
3. La mise en œuvre dudit modèle Observable-Observé s'appuiera sur la classe `Observable` et l'interface `Observer` de Java
4. Les pendules ne devront recevoir un message `update` de la part de l'émetteur qu'une seule fois par minute