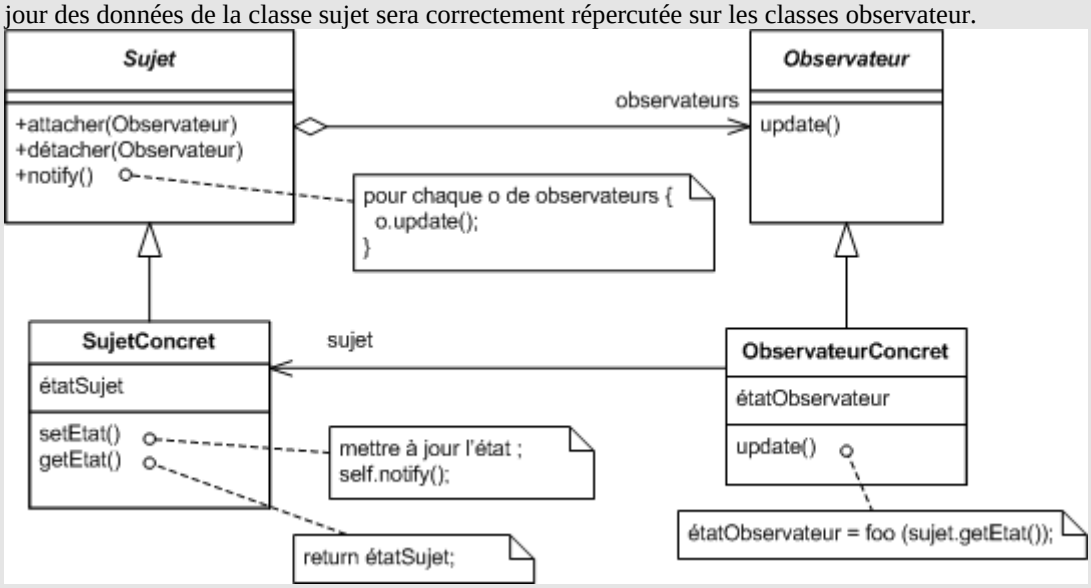
Exercice sur 2 séances : La montre chronomètre

Cet exercice peut s'aborder comme trois exercices relativement indépendants, correspondant aux différentes fonctionnalités proposées. L'architecture logicielle proposée s'appuie sur le design pattern Observer (voir encadré).

Design Pattern Observer

Ce design pattern [Gamma 95] ou patron de solution offre une solution propre aux problèmes de mise à jour des données de classes liées entre elles. Un des problèmes du partitionnement d'une application en classes est le maintien de la cohérence des parties de l'application. Pour favoriser la réutilisabilité, il n'est pas souhaitable de coupler fortement les classes stockant les données (classes *sujet*) de celles qui les affichent ou plus généralement les utilisent (classes *observateur*). Pourtant il faut assurer qu'une mise à jour des données de la classe sujet sera correctement répercutée sur les classes observateur.



Le *design pattern* « Observer » décrit une solution à ce problème. Un sujet peut avoir un nombre indéterminé d'observateurs, et tous les observateurs sont notifiés du changement quand le sujet subit une mise à jour. En réponse, les observateurs interrogent le sujet pour connaître son nouvel état et mettre ainsi à jour leur propre état.
Ce type d'architecture est parfois appelé aussi *publish/subscribe (*publication/abonnement) : le sujet publie des notifications, les observateurs s'y abonnent s'ils le souhaitent. Le point important est que le sujet ignore quels sont ses observateurs, fournissant un modèle bien découplé.

Enoncé

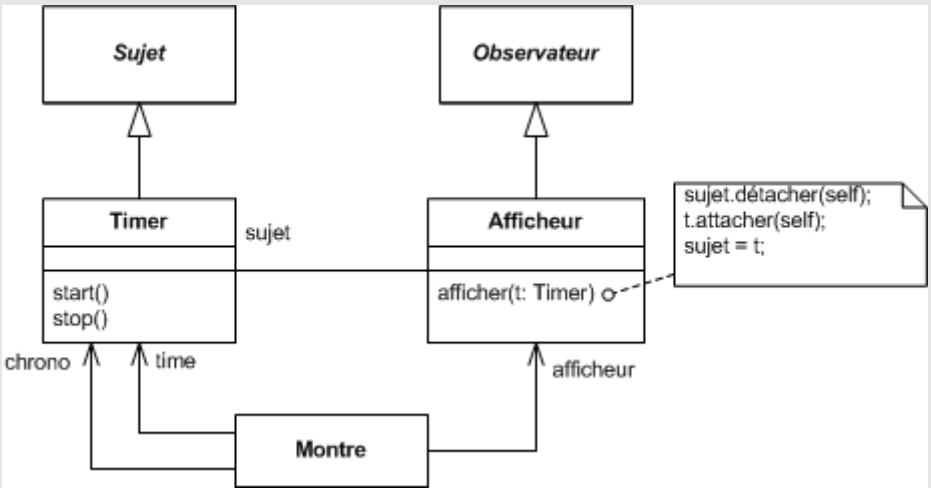
Soit une montre digitale proposant une fonction horloge et une fonction chronomètre. Elle est munie de quatre boutons :

- *light* : permet d'allumer une lumière. Une pression sur ce bouton allume une lumière pendant une durée de deux secondes.
- *mode* : permet d'alterner entre les trois modes principaux de la montre : l'affichage de l'heure courante, le mode chronomètre, et le mode réglage (pour mettre à jour l'heure courante).
- *start-stop* : ce bouton permet de lancer et d'arrêter le chronomètre quand on se trouve dans le mode chrono. Ce bouton permet également d'alterner entre le réglage de l'heure, des minutes ou des secondes dans le mode réglage.

- *set* : ce bouton permet d'incrémenter les heures (ou minutes…) en mode réglage et de remettre à zéro le chronomètre.

Nous allons décomposer l'étude du fonctionnement de cette montre en plusieurs étapes.

On se donne une classe Timer qui sait gérer des dates/heures, et qui est munie de toutes les opérations de mise à jour qui vous paraîtront utiles, ainsi que d'une méthode *start()* et *stop()* qui lancent et arrêtent respectivement l'écoulement du temps. Le lien entre l'afficheur à cristaux liquides de la montre et les instances de Timer gérant respectivement le chronomètre et l'heure courante sera réalisé à travers une instance du design pattern *Observer* (voir encadré).



1. **Modes principaux et éclairage.**

   Modélisez dans un premier temps le comportement lié à l'utilisation du bouton *mode* de la montre. Ajoutez le comportement lié à l'utilisation de la lumière (bouton *light*).

2. **Mode chronomètre.**

   En mode chronomètre, le bouton *start-stop* lance ou arrête le chronomètre. Le bouton *set* permet de remettre le chronomètre à zéro s'il est arrêté. Modélisez ces comportements.

   Ajoutez la possibilité de retrouver l'état précédent en changeant de mode : il est possible de lancer le chronomètre, puis de basculer en mode affichage de l'heure, et de revenir au mode chronomètre : le chronomètre continue à tourner pendant cette opération. (On pourra utiliser un pseudo-état historique.)

3. **Mode réglage.**

   En mode réglage, le bouton *start-stop* permet d'alterner entre le réglage de l'heure, des minutes, ou des secondes. Le bouton *set* incrémente l'heure courante d'une heure en mode réglage de l'heure, d'une minute en mode réglage des minutes, ou remet les secondes à zéro en mode réglage des secondes. L'afficheur fera clignoter le champ actuellement sélectionné. Modélisez ces comportements.

Modélisez les différents comportements de la montre à l'aide de diagrammes états transition.

Implémentez une montre en Java selon un modèle évenementiel qui suit votre description de machines à états. Une IHM en Swing sera proposée, munie des trois boutons et de l'afficheur.

Un corrigé type pour la partie modélisation de la montre vous sera fournie à l'issue du de la première séance.