TD2 : Modèle de Conception et architecture MVC

En génie de Logiciel, un modèle de conception (*design pattern* en anglais) est un concept destiné à résoudre les problèmes récurrents suivant le paradigme objet. En français on utilise aussi les termes de **motif de conception** ou de patron de conception.

Les modèles de conception décrivent des solutions standard pour répondre à des problèmes d'architecture et de conception des logiciels. On peut considérer un patron de conception comme une formalisation de bonnes pratiques, ce qui signifie qu'on privilégie les solutions éprouvées.

Il ne s'agit pas de fragments de code, mais d'une manière standardisée de résoudre un problème qui s'est déjà posé par le passé. On peut donc considérer les patrons de conception comme un outil de capitalisation de l'expérience appliqué à la conception logicielle.

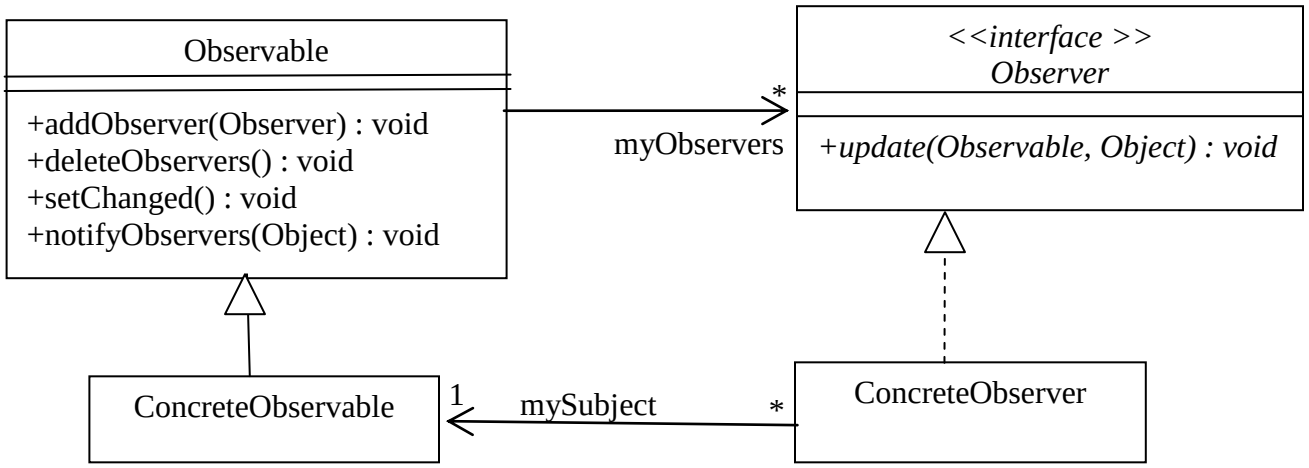
Inconvénients des Modèles de Conception
L'utilisation des Modèles de Conception n'est pas des plus simples et ne convient assurément pas à tout le monde. Les modèles de conceptions requièrent une conception détaillée. Il introduit en outre un niveau approfondi de complexité qui nécessite une attention de tous les instants aux détails.

La charge de travail supplémentaire induite ne doit pas être sous-estimée. Ne l'oubliez pas.

Il en résulte que l'usage des modèles de conception peut être trop complexe pour les petites applications, et même pour bon nombre d'applications moyennes

Exercice 1 En Java, écrivez le code de l'interface `Observer` et le squelette des classes `Observable`, `ConcreteObservable` et `ConcreteObserver` (sans le corps des méthodes).

Modèle de conception « observateur/observable » : Les observables envoient une notification à leurs observateurs en cas de changement de leur état (c'est-à-dire qu'ils exécutent les méthodes setChanged, et notifyObservers). En cas de notification, la méthode update des « ConcreteObserver » concernés est appelé (donc cette méthode est exécutée). Durant l'exécution de cette méthode, les ConcreteObservable peuvent obtenir des informations complémentaires par l'appel d'une ou de plusieurs méthodes de la classe ConcreteObservable.
La classe Observable ainsi que l'interface Observer font parties de l'API Java.



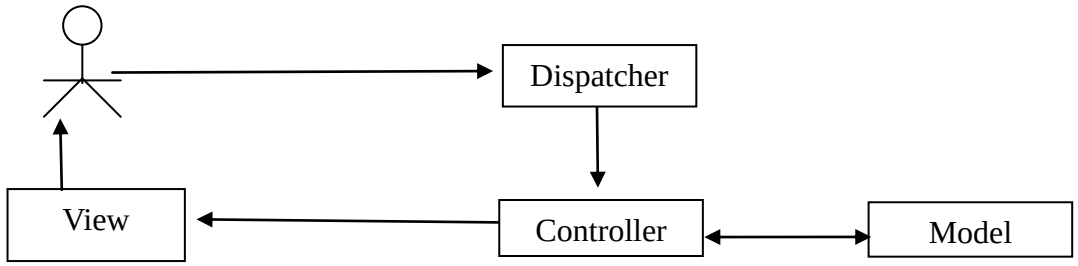
MVC

Le **Modèle-Vue-Contrôleur**, en abrégé MVC, de l'anglais *Model-View-Controller* est une architecture logicielle. Cette architecture a été mise au point en 1979 par Trygve Reenskaug, qui travaillait alors sur Smalltalk dans les laboratoires de recherche Xerox PARC.

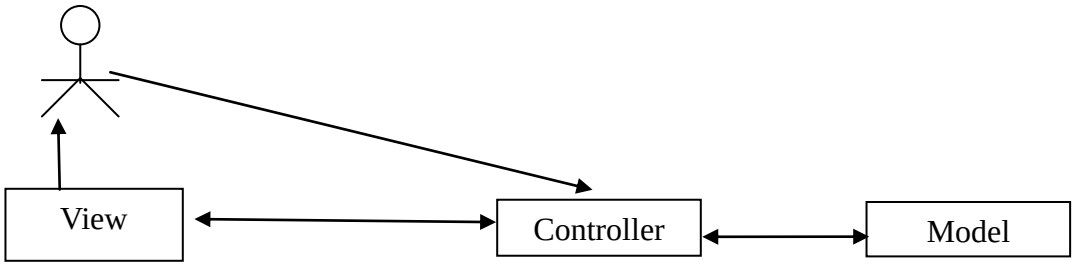
MVC l'une des premières approches pour décrire et mettre en œuvre des architectures basées sur la notion de responsabilité. L'idée était de bien séparer les données, la présentation et les traitements des événements utilisateur. Il en résulte les trois parties : le modèle, la vue et le contrôleur.

Bien que développé à l'origine pour les applications "standard", le MVC a été largement adopté comme une architecture dans le monde des applications WEB. Plusieurs "système" Web commerciales et non commerciales ont mises en œuvre ce modèle. L'interprétation de ce modèle, principalement dans la façon dont les responsabilités varie d'un système à un autre.

Par exemple, l'architecture de cakePHP est la suivante :



qui est une adaptation de l'architecture MVC dite passive :



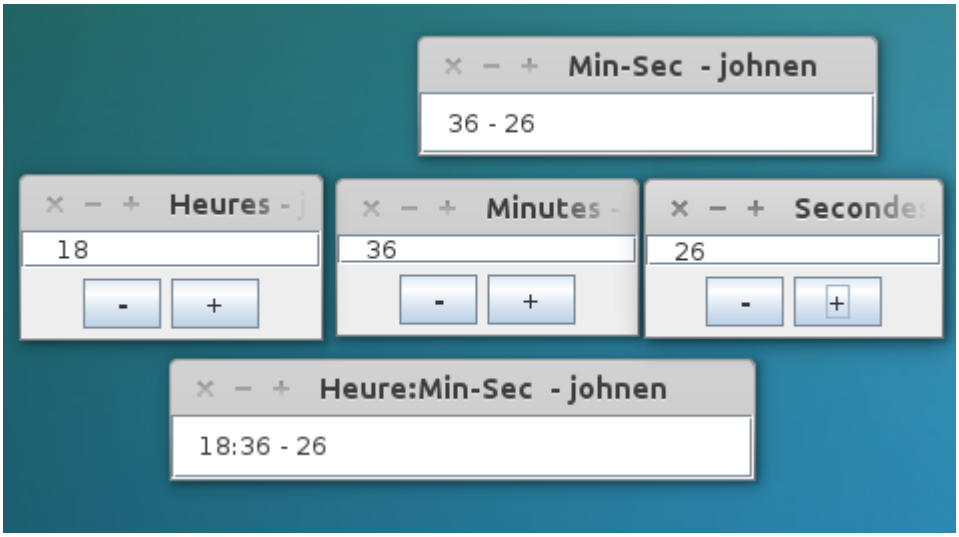
Pour plus d'information

http://www.albertzuurbier.com/index.php/programming/84-mvc-vs-mvp-vs-mvvm
http://forums.codeguru.com/showthread.php?517017-A-MVP-pattern-doubt
http://forums.codeguru.com/showthread.php?517594-Is-MVC-and-MVP-supervising-controller-the-same

Exercice 2

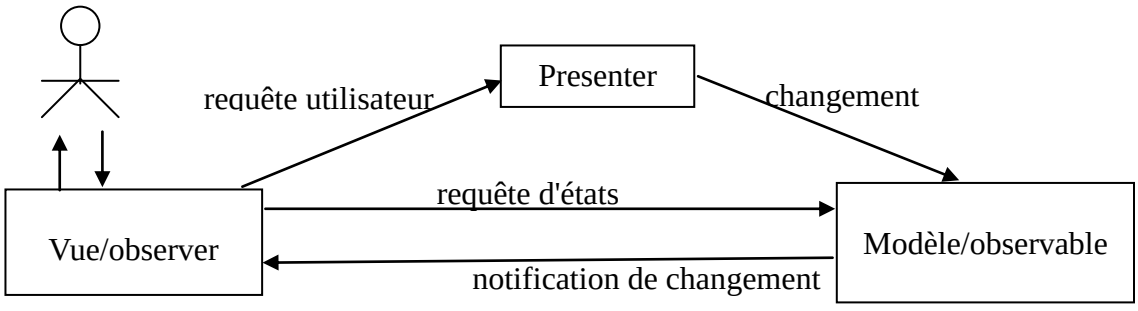
L'objectif est de créer un minuteur (heure, minute seconde) pouvant être contrôlé via plusieurs interfaces. Dans notre illustration, il y a 5 fenêtres associées au même minuteur. La modification des valeurs du minuteur doit être reportée dans toutes les fenêtres concernées.

- La valeur des secondes est entre 0 et 59 (ainsi que la valeur des minutes) ; la valeur des heures est entre 0 et 23.
- L'incrémentation des seconde et minutes se propage ; donc ajouter une seconde à 5h59mm59s a pour résultat 6h0mm0s ; et retirer 5 secondes à 7h00mn2s a pour résultat 6h59mm58s



5 interfaces graphiques du même minuteur

1. Ecrivez le diagramme de classes correspondant à l'application vous devez utiliser le patron de conception Observable/Observateur dans le cadre d'une architecture logicielle variante du MVC, nommé MVP. Plus précisément, vous devez suivre le schéma suivant.



2. Elaborez le diagramme de communication associé au scénario suivant :

Après que l'application soit lancée, l'utilisateur augmente la valeur du nombre de seconde via le bouton « plus » de fenêtre « Seconde», les affichages sont mis à jour.

Lors **d'un autre TP**, vous implémenterez en Java l'application que vous avez conçus (diagramme de classes).
Votre code doit « passer » les tests suivants (fichier déposé sous moodle du cours M3105) :

```
public class MinuteurTests {
      static int HMS_VALEUR = 10;
      static int ZERO = 0;
      static int UN = 1;
      static int NEG_UN = -1;
      MinuteurModele modele;
      MinuteurControleur  controleur;

      // initialiser les variables modele controleur.
      //annotation "@Before" : initial() est exécuté avant les tests
      @Before
      public void initial() {
            modele = new MinuteurModele(ZERO, ZERO, ZERO);
            controleur = new MinuteurControleur(modele);  }

      // annotation "@Test" : méthode suivante est un test
      // tester les méthodes setHeure(), getHeure(),setMinute()
      // getMinute(), setSeconde() et getSeconde().
      // MinuteurModele.MAX_HEURE == 24
      // MinuteurModele.MAX_MINSEC== 60
      @Test
      public void testSetEtGet() {   /*code omis*/  }


      // tester la méthode incHeure(x) de MinuteurControleur
      /* 23h plus une heure donne 0h */
      /* 0h plus une heure donne 1h */
      /* 0h moins une heure donne 23h */
      /* 23h moins une heure donne 22h */
      @Test
      public void testIncEtDecHeure() {/*code omis*/  }


     // tester la méthode incMinute(x) de MinuteurControleur
     /* 10h:59m plus une minute donne 11h:0m */
     /* 11h:0m plus une minute donne 11h:1m */
      /* 10h:0m moins une minute donne 9h:59m */
      /* 9h:59m moins une minute donne 9h:58m */
      @Test
      public void testIncEtDecMinute() {/*code omis*/ }


      // tester la méthode incSeconde(x) de MinuteurControleur avec x = 1;
       /* 10m:59s plus une seconde donne 11m:0s */
      /* 11m:0s plus une seconde donne 11m:1s */
      /* 10h:59m:59s plus une seconde donne 11h:0m:0s */
      /* 23h:59m:59s plus une seconde donne 0h:0m:0s */
      @Test
      public void testIncSeconde() { /*code omis*/ }
```

```
// tester la méthode incSeconde(x) de MinuteurControleur avec x = -1;
      @Test
      public void testDecSeconde() {
            /* 10m:0s moins une seconde donne 9m:59s */
            controleur.setMinute(HMS_VALEUR); controleur.setSeconde(ZERO);
            controleur.incSeconde(NEG_UN);
            assertEquals(HMS_VALEUR - UN, mn.getMinute());
            assertEquals(MinuteurModele.MAX_MINSEC - UN, mn.getSeconde());
            /* 9m:59s moins une seconde donne 9m:58s */
            controleur.incSeconde(NEG_UN);
            assertEquals(HMS_VALEUR - UN, mn.getMinute());
            assertEquals(MinuteurModele.MAX_MINSEC - DEUX, mn.getSeconde());
            /* 10h:0m:0s moins une seconde donne 9h:59m:59s */
            controleur.setHeure(HMS_VALEUR) ;
           controleur.setMinute(ZERO); controleur.setSeconde(ZERO);
            controleur.incSeconde(NEG_UN);
            assertEquals(HMS_VALEUR - UN, mn.getHeure());
            assertEquals(MinuteurModele.MAX_MINSEC - UN, mn.getMinute());
            assertEquals(MinuteurModele.MAX_MINSEC - UN, mn.getSeconde());
            /* 0h:0m:0s moins une seconde donne 23h:59m:59s */
            controleur.setHeure(ZERO);
             controleur.setMinute(ZERO); controleur.setSeconde(ZERO);
            controleur.incSeconde(NEG_UN);
            assertEquals(MinuteurModele.MAX_HEURE - UN, mn.getHeure());
            assertEquals(MinuteurModele.MAX_MINSEC - UN, mn.getMinute());
            assertEquals(MinuteurModele.MAX_MINSEC - UN, mn.getSeconde()); }
}
```

A rendre :

- Le diagramme de classes cohérent avec le code du test *testDecSeconde*
- Le diagramme de communication
- Le code du minuteur correspondant au diagramme de classes et qui fonctionne correctement (il passe tous les tests).

Optionnel – créer un Thread (processus légers/fil d'exécution) qui incrémente toutes les 5 secondes le Minuteur. Pour créer un processus léger, il faut

- créer un objet de la classe Thread (ou d'une classe héritant de la classe Thread).
- exécuter sur cette objet la méthode `start()` ; cette méthode crée le fil d'exécution qui invoque sa méthode `run()` pour connaître les actions à exécuter.

Optionnel - Ecrivez le diagramme de classes en suivant le schéma correspondant à la version originale de l'architecture MVC.