TD7 : Modèle de Conception et 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.

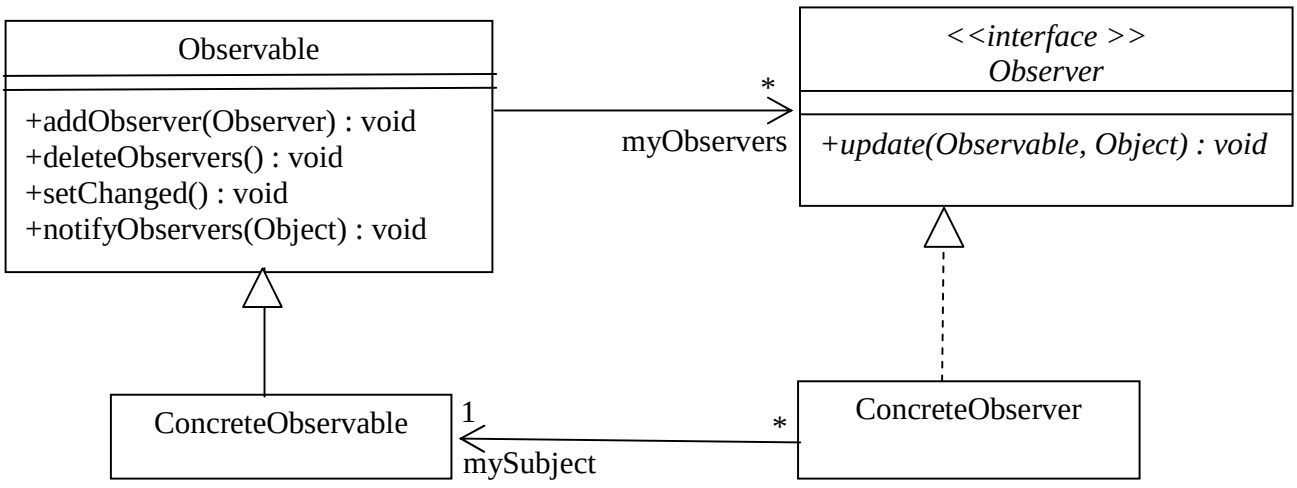
Inconvénients des Modèles de Conception
L'utilisation des Modèles de Conception n'est pas des plus simples et ne convient assurément pas à tout le monde. Les modèles de conceptions requièrent une conception détaillée. Il introduit en outre un niveau approfondi de complexité qui nécessite une attention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

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

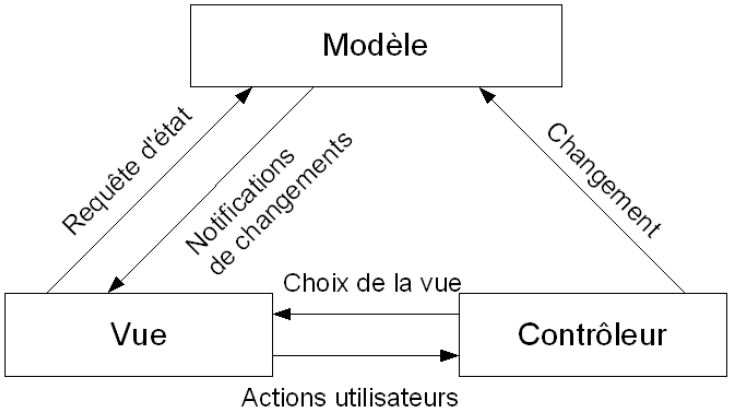
Il en résulte que l'usage des modèles de conception peut être trop complexe pour les petites applications, et même pour bon nombre d'applications moyennes. Le travail supplémentaire n'en vaut certainement pas la peine dans ces cas-là. Prendre le temps de définir une architecture complexe ne sera peut-être pas rentable.

Exercice 1 : Codez les classes Observable, ConcreteSubject et ConcreteObserver en java

Modèle de conception « observateur/observable » : Les observables envoient des notifications à leurs observateurs en cas de changement. En cas de notification, les observateurs exécutent leur méthode update ; ils peuvent obtenir des informations complémentaires par l'appel à la méthode getState de l'observable.



MVC



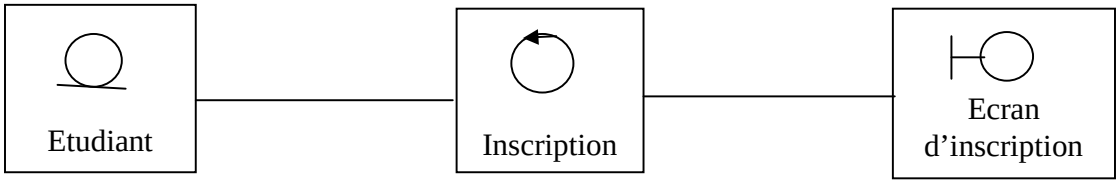
Le **Modèle-Vue-Contrôleur** (en abrégé MVC, de l'anglais (*Model-View-Controller*) est une architecture logicielle et une méthode de conception logicielle .Cette méthode a été mise au point en 1979 par Trygve Reenskaug, qui travaillait alors sur Smalltalk dans les laboratoires de recherche Xerox PARC[1].

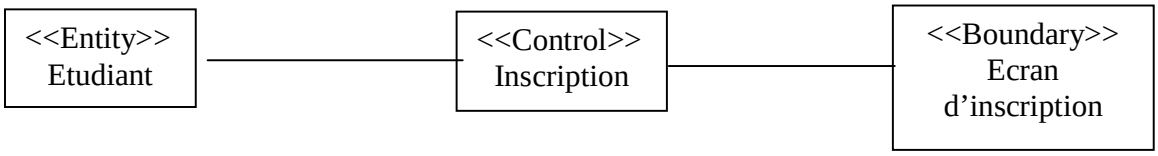
MVC impose la séparation entre les données, les traitements et la présentation. C'est pour cette raison que l'application est divisée en trois composants fondamentaux: le modèle, la vue et le contrôleur. Chacun de ces composants tient un rôle bien défini.

La **vue** correspond à l'interface avec laquelle l'utilisateur interagit. Sa première tâche est de présenter les résultats renvoyés par le modèle. Sa seconde tâche est de recevoir toutes les actions de l'utilisateur (clic de souris, sélection d'une entrée, boutons…). Ces différents événements sont envoyés au contrôleur. La vue n'effectue aucun traitement, elle se contente d'afficher les résultats des traitements effectués par le modèle. Plusieurs vues, partielles ou non, peuvent afficher des informations d'un même modèle. En UML2, le stéréotype <<Boundary>> s'applique aux classes de la composante vue.

Le **modèle** représente le comportement de l'application : traitements des données, interactions avec la base de données, etc. Il décrit ou contient les données manipulées par l'application. Il assure la gestion de ces données et garantit leur intégrité. Dans le cas typique d'une base de données, c'est le modèle qui la contient. Le modèle offre des méthodes pour mettre à jour ces données (insertion, suppression, changement de valeur). Il offre aussi des méthodes pour récupérer ces données. Les résultats renvoyés par le modèle sont dénués de toute présentation. En UML2, le stéréotype <<Entity>> s'applique aux classes de la composante modèle.

Le **contrôleur** prend en charge la gestion des événements de synchronisation pour mettre à jour la vue ou le modèle et les synchroniser. Il reçoit tous les événements de l'utilisateur et enclenche les actions à effectuer. Si une action nécessite un changement des données, le contrôleur demande la modification des données au modèle et ensuite avertit la vue que les données ont changé pour qu'elle se mette à jour. Certains événements de l'utilisateur ne concernent pas les données mais la vue. Dans ce cas, le contrôleur demande à la vue de se modifier. Le contrôleur n'effectue aucun traitement, ne modifie aucune donnée. En UML2, le stéréotype <<Control>> s'applique aux classes de type « contrôleur ».

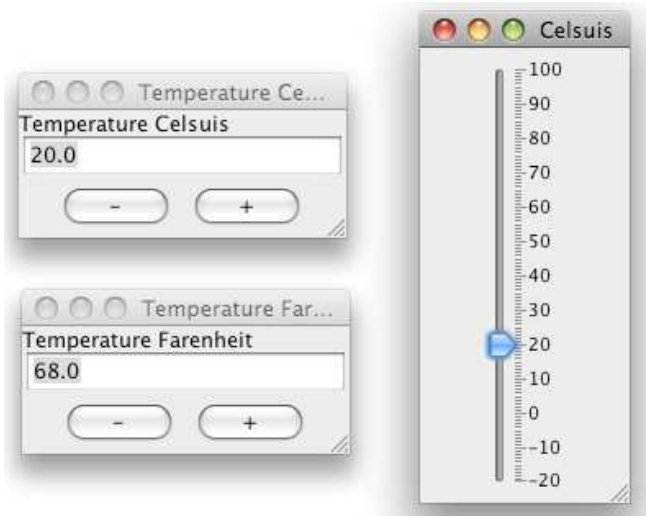




Exercice 2

Extrait d'un TP de Master1 – Université de Lille 1 - (resp. Gery Casiez)

L'objectif est de créer une interface permettant le contrôle d'une température en degrés Celsius ou Fahrenheit. L'interface se compose de trois vues représentant la même température sous des formes différentes. La modification d'une des vues doit mettre automatiquement à jour les autres vues.



L'interface de contrôle de la gestion de la température

1. Concevez une application en UML implémentant l'application et donnez le squelette du code en JAVA (utilisez le modèle MVC et le modèle Observable/Observateur).
2. Elaborez le diagramme de communication associé au scénario suivant :
    - l'application est lancée;
    - l'utilisateur modifie la température via la notation Celsius;
    - les affichages sont mis à jour;
    - l'application est fermée via la vue « Fahrenheit ».