TD4 : Architecture 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.

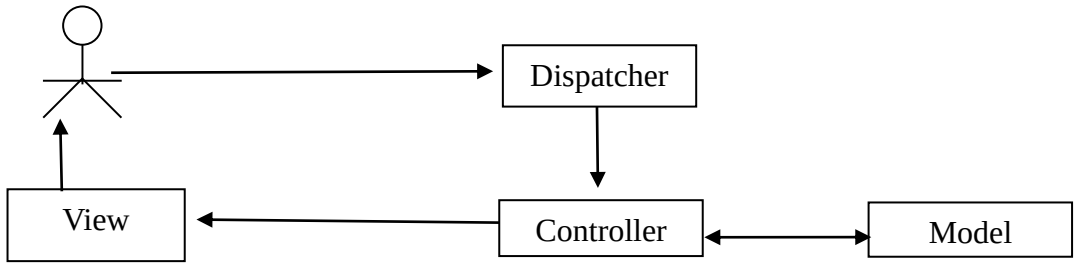
MVC est l'une des premières approches pour décrire et mettre en œuvre des architectures basées sur la notion de responsabilité. L'idée était de bien séparer les données, la présentation et les traitements des événements utilisateur. Il en résulte les trois parties :

- modèle : données (accès et mise à jour)
- vue : interface utilisateur
- contrôleur : gestion des événements et synchronisation

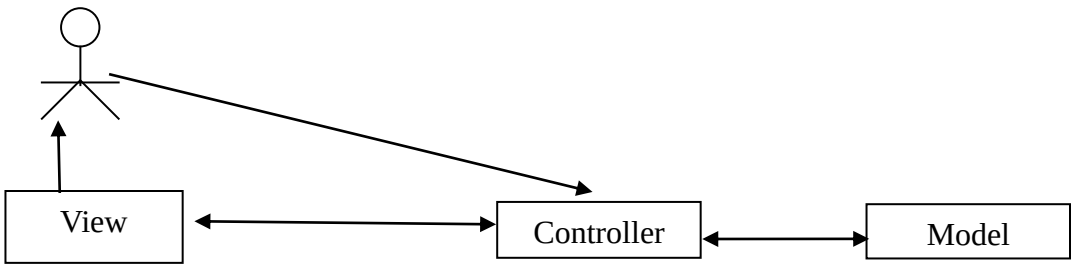
Bien que développé à l'origine pour les applications « standard », le MVC a été largement adopté comme une architecture dans le monde des applications WEB. Plusieurs « systèmes Web » commerciaux et non commerciaux ont mises en œuvre ce modèle. L'interprétation de ce modèle, principalement dans la façon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

Dans une architecture MVC, la vue consulte directement le modèle (lecture) sans passer par le contrôleur. Par contre, elle doit nécessairement passer par le contrôleur pour effectuer une modification (écriture).

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

L'objectif de l'application est de créer une horloge (heure, minute seconde) pouvant être contrôlée via plusieurs interfaces. Dans notre illustration, il y a 5 fenêtres associées à la même horloge. Une modification des valeurs de l'horloge doit être reportée dans toutes les fenêtres concernées. Dans les trois fenêtres de droite (Hour, Minute, Second), il est possible de modifier un élément via les deux boutons, mais aussi via la zone textuelle.

- 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 secondes et des minutes se propage ; donc ajouter une seconde à 5h59min59s a pour résultat 6h0min0s ; et retirer 5 secondes à 7h00min2s a pour résultat 6h59min58s.

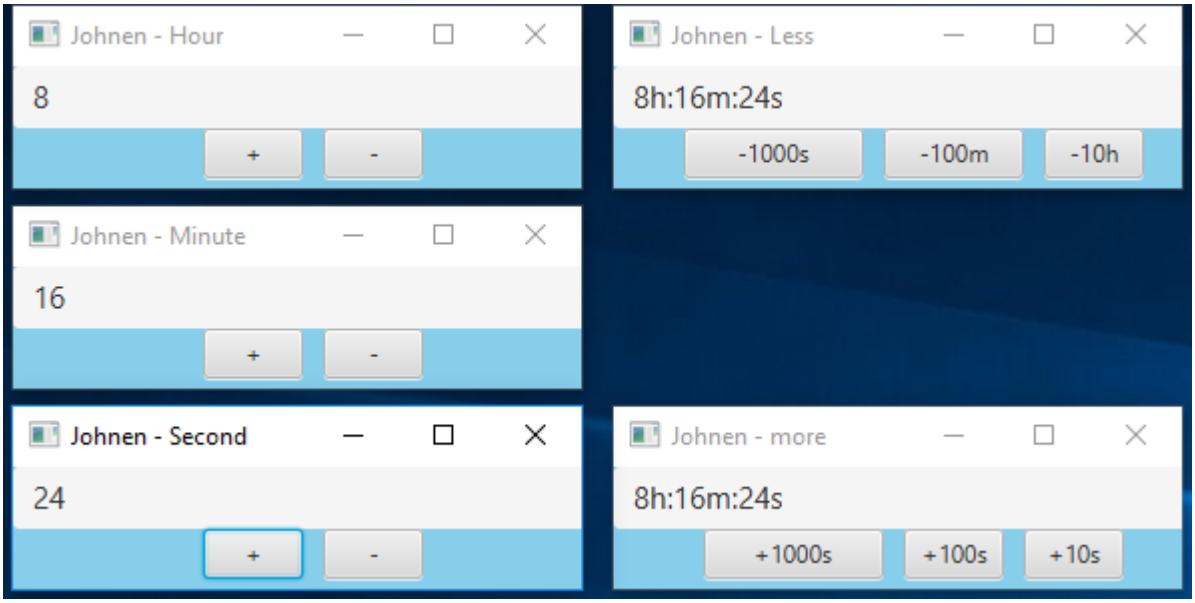
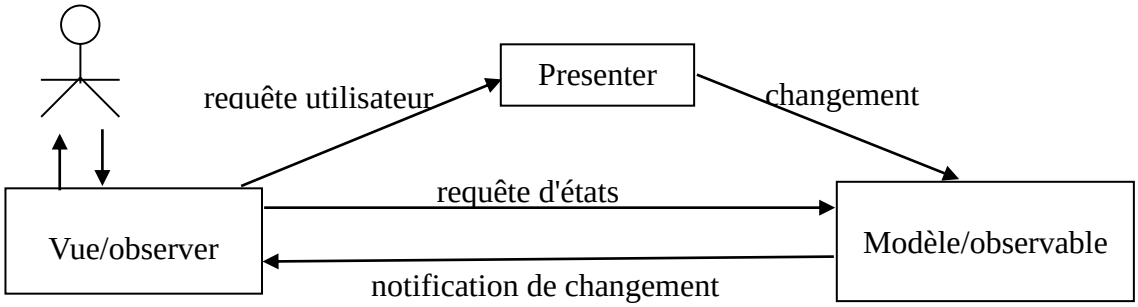


Figure : 5 interfaces graphiques de la même horloge

```
public class ClockModel extends Observable {
      public static final int MAX_HOUR=24;
      public static final int MAX_MINSEC=60;
      public static final int MIN_TIME=0;
      private int hour=0;
      private int minute=0;
      private int second = 0;

      public ClockModel() { }
      public ClockModel(int h, int m, int s) {
                                    hour = h; minute = m; second = s;     }
      public int getHour() {return hour;}
      public int getMinute() {return minute;}
      public int getSecond() {return second;}

//  ClockException est jetée si h n'est pas inclus entre 0 et 23 inclus.
      public void setHour(int h) throws ClockException {/*code à écrire */}

//  ClockException est jetée si m n'est pas inclus entre 0 et 59 inclus.
      public void setMinute(int min) throws ClockException {/*code à écrire
*/}
//  ClockException est jetée si s n'est pas inclus entre 0 et 59 inclus.
     public void setSecond(int s) throws ClockException {/*code a écrire */}}
```

Questions

1. Ecrivez le code des méthodes *setHour*, *setMinute*, et *setSecond*.
2. 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.



3. Elaborez le diagramme de communication associé au scénario suivant :
   Après que l'application soit lancée, l'horloge a la valeur 8h16min24s, l'utilisateur augmente la valeur du nombre de secondes via le bouton « plus » de la fenêtre « Seconde», les affichages sont mises à jour.
4. Implémentez en JavaFX l'application que vous avez conçue (diagramme de classes). Votre code doit « passer » les tests (fichier déposé sous le moodle du cours M3105).

   Voici un extrait du fichier « clockTests » :

```
package tests;

import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;

import java.util.Random;

import clockController.ClockController;
import clockModel.ClockModel;

public class clockTests {
	static int HMS_VALEUR = 10;
	static int ZERO = 0;
	static int UN = 1;
	static int DEUX = 2;
	static int MAX_TESTS = 1000;
	ClockModel model;
	ClockController  controller;
	Random rd;

	// initialisation des variables model controller.
	//annotation "@Before" : la méthode initial() est exécutée avant les tests
	@Before
	public void initial() {
		model = new ClockModel(ZERO,ZERO,ZERO);
		controller = new ClockController(model);
		rd = new Random()  }
```

```
 // debit est un entier positif ou null
private void testDecSecond(int debit) {
	int  s, m, h, sec, min, heure, reportMin = 0, reportHeure = 0;
	s = model.getSecond();
	m = model.getMinute();
	h = model.getHour();
	controller.incSecond(-debit);
	sec = (s-debit) % ClockModel.MAX_MINSEC;
	if (sec <0) {sec+=ClockModel.MAX_MINSEC; reportMin  = -1;}
	min  = (m + reportMin +
		(s - debit)/ClockModel.MAX_MINSEC)%ClockModel.MAX_MINSEC;
	if (min<0) { min+=ClockModel.MAX_MINSEC; reportHeure = -1; }
	heure = (h + reportHeure +
			(m + reportMin +
					(s-debit)/ClockModel.MAX_MINSEC
			 ) /ClockModel.MAX_MINSEC
		) % ClockModel.MAX_HOUR;
	If (heure < 0) heure  += ClockModel.MAX_HOUR;
	assertEquals(sec, model.getSecond());
	assertEquals(min, model.getMinute());
	assertEquals(heure, model.getHour());
}



// test la méthode incSecond(x) avec x positif
@Test
public void testsIncSecond() {
	// 10min:59s plus une Second = 11min:0s
	controller.setMinute(HMS_VALEUR);
	controller.setSecond(ClockModel.MAX_MINSEC - UN);
	testIncSecond(UN);
	// 11min:0s plus une Seconde = 11min:1s
	testInce(UN);
	// 10h:59min:59s plus une seconde = 11h:0min:0s
	controller.setHour(HMS_VALEUR);
	controller.setMinute(ClockModel.MAX_MINSEC - UN);
	controller.setSecond(ClockModel.MAX_MINSEC - UN);
	testIncSecond(UN);
	// 23h:59min:59s plus une seconde = 0h:0min:0s
	controller.setHour(ClockModel.MAX_HOUR - UN);
	controller.setMinute(ClockModel.MAX_MINSEC - UN);
	controller.setSecond(ClockModel.MAX_MINSEC - UN);
	testIncSecond(UN);
	// 0h:0min:0s plus 24*3600 secondes  = 0h:0min:0s
	testIncSecond(ClockModel.MAX_MINSEC*ClockModel.MAX_MINSEC*ClockModel.MAX_HOUR);
	//tests aleatoires
	for (int i = 0 ; i < MAX_TESTS; i++) {
		testIncSecond(rd.nextInt(MAX_TESTS));
	}
}
```

```
    // test la méthode incSecond(x) avec x  negatif
    @Test
    public void testsDecSecond() {
          // 10min:0s moins une seconde = 9min:59s
          controller.setMinute(HMS_VALEUR); controller.setSecond(ZERO);
          testDecSecond(UN);
          // 9min:59s moins une seconde = 9min:58s
          testDecSecond(UN);
          // 10h:0min:0s moins une seconde = 9h:59min:59s
          controller.setHour(HMS_VALEUR); controller.setMinute(ZERO);
          controller.setSecond(ZERO);
          testDecSecond(UN);
          // 0h:0min:0s moins une seconde = 23h:59min:59s
          controller.setHour(ZERO); controller.setMinute(ZERO);
          controller.setSecond(ZERO);
          testDecSecond(UN);
          // 23h:59min:59s  moins 24*3600 secondes  = 23h:59min:59s
          testDecSecond(ClockModel.MAX_MINSEC*ClockModel.MAX_MINSEC*ClockModel.MAX_HOUR);
          // 23h:59min:59s  moins 23*3600 secondes  = 0h:59min:59s
          testDecSecond(ClockModel.MAX_MINSEC*ClockModel.MAX_MINSEC*(ClockModel.MAX_HOUR-UN));
          // 0h:59min:59s  moins 3600 secondes  = 23h0min:1s
          testDecSecond(ClockModel.MAX_MINSEC*ClockModel.MAX_MINSEC*(ClockModel.MAX_HOUR-UN));
          //tests aleatoires
          for (int i = 0 ; i < MAX_TESTS; i++) {
                testDecSecond(rd.nextInt(MAX_TESTS));
          }
    }


}
```

A rendre – Règles de présentation : un document ne suivant pas ces règles ne sera pas corrigé
Sur toutes le feuilles, il faut vos noms, le nom du groupe ; il y a au plus un diagramme par feuille ; chaque diagramme est nommé ; les diagrammes sont lisibles.

- le diagramme de classes cohérent avec le code des tests.

- le diagramme de communication « Après que l'application soit lancée, l'horloge a la valeur 8h16min24s, l'utilisateur augmente la valeur du nombre de secondes via le bouton « plus » de la fenêtre « Seconde», les affichages sur les 5 fenêtres sont mises à jour ».
  le diagramme de communication doit être cohérent avec votre diagramme de classes et votre code.

- le code du horloge (interface utilisateur programmée avec JavaFX) correspondant au diagramme de classes qui fonctionne correctement : votre code doit passer tous les tests (sans aucune modification dans la classe « ClockTests ») et surtout il doit réaliser les fonctionnalités attendues.

Optionnel – créer un Thread (aussi nommé processus léger ou fil d'exécution) qui incrémente toutes les 5 secondes le Horloge. 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.