TD5 : 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.

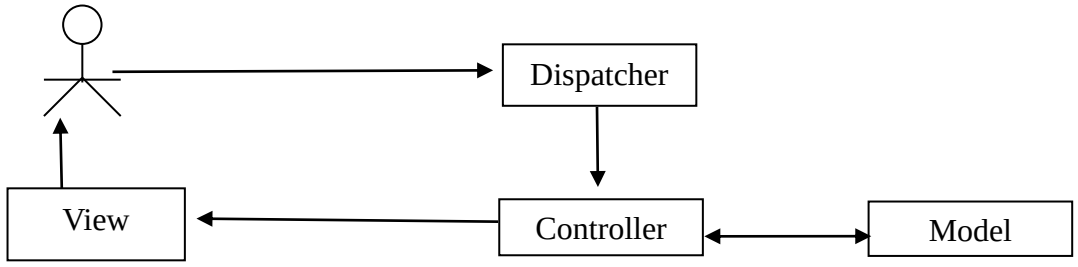
MVC est l'une des premières approches pour décrire et mettre en œuvre des architectures basées sur la notion de responsabilité. L'idée était de bien séparer les données, la présentation et les traitements des événements utilisateur. Il en résulte les trois parties :

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

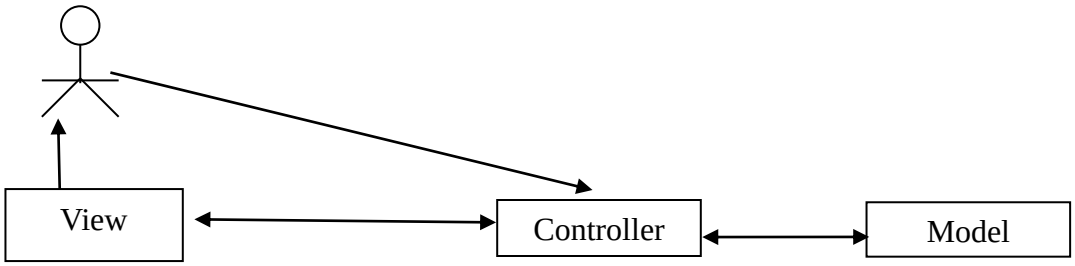
Bien que développé à l'origine pour les applications "standard", le MVC a été largement adopté comme une architecture dans le monde des applications WEB. Plusieurs « systèmes Web » commercials et non commercials ont mises en œuvre ce modèle. L'interprétation de ce modèle, principalement dans la façon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

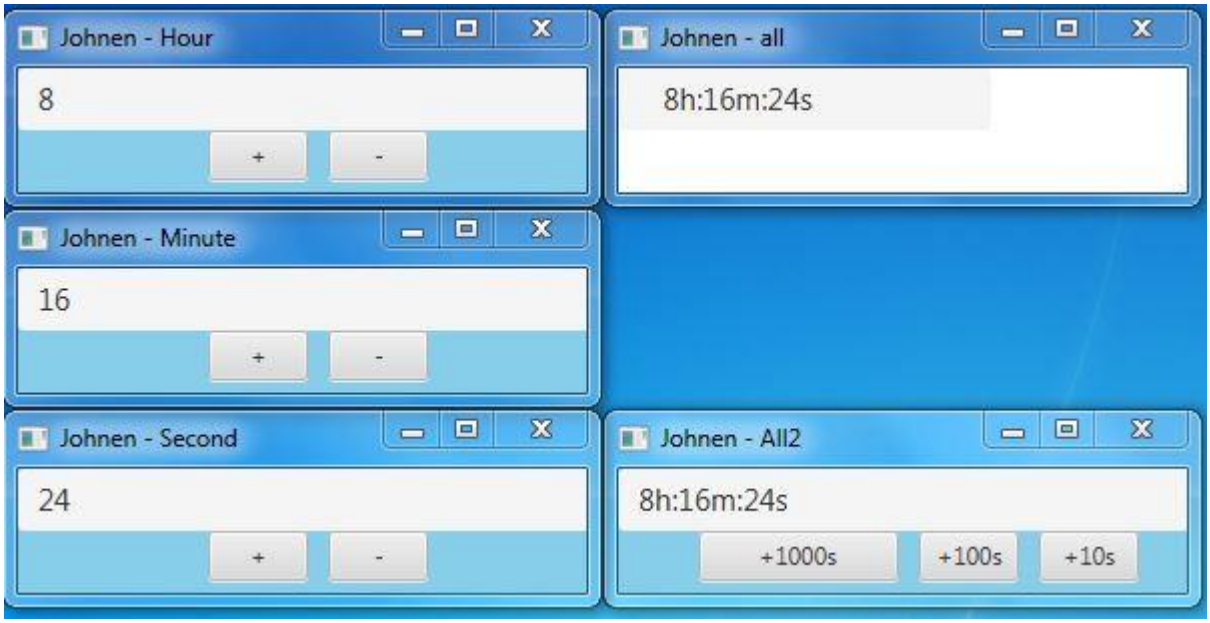


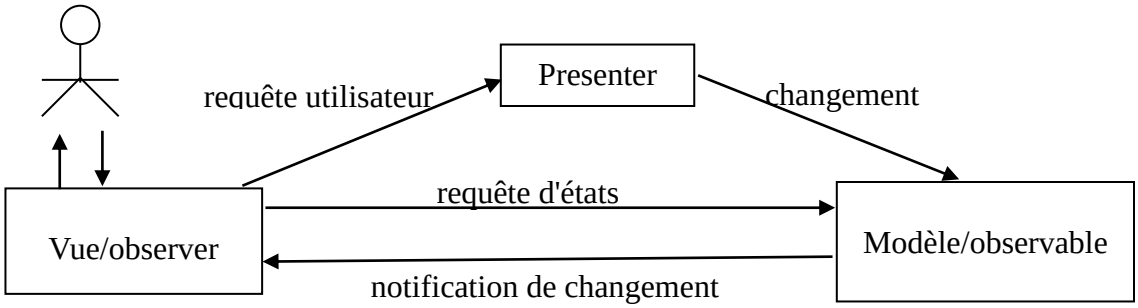
Figure : 5 interfaces graphiques du même minuteur

```
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;
      int hour=0;
      int minute=0;
      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 m) 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, le minuteur a la valeur 8h16m24s, l'utilisateur augmente la valeur du nombre de secondes via le bouton « plus » de fenêtre « Seconde», les affichages sont mis à jour.
4. Implémentez en JavaFX l'application que vous avez conçue (diagramme de classes). Votre code doit « passer » les tests suivants (fichier déposé sous le moodle du cours M3105) :

```
public class ClockTests {
	static int HMS_VALEUR = 10;
	static int ZERO = 0;
	static int UN = 1;
	static int DEUX = 2;
	static int NEG_UN = -1;
	ClockModel model;
	ClockControler  controler;
	Random rd;

	// initialise les variables model controleur.
	//annotation "@Before" : initial() est executé les tests
	@Before
	public void initial() {
		model = new ClockModel(ZERO,ZERO,ZERO);
		controler = new ClockControler(model);
		rd = new Random();
	}

// tester les méthodes setHour(), getHour(), setMinute(), getMinute(), setSecond() et
// getSecond()
	@Test
	public void testSetEtGet(){ /* code omis */ }


	// tester la méthode incHour(x) avec x = 1 et avec x=-1;
```

```java
// tester la méthode incHour(x) avec x choisi aléatoirement
@Test
public void testIncEtDecHeure(){ /* code omis */ }

// tester la méthode incMinute(x) avec x = 1 et avec x=-1;
// tester la méthode incMinute(x) avec x choisi aléatoirement
@Test
public void testIncEtDecMinute(){ /* code omis */ }


// tester la méthode incSecond(x) avec x = 1;
@Test
public void testIncSeconde() {
	// 10m:59s plus une seconde donne 11m:0s
	controler.setMinute(HMS_VALEUR);
	controler.setSecond(ClockModel.MAX_MINSEC - UN);
	controler.incSecond(UN);
	assertEquals(HMS_VALEUR + UN, model.getMinute());
	assertEquals(ZERO, model.getSecond());
	// 11m:0s plus une seconde donne 11m:1s
	controler.incSecond(UN);
	assertEquals(HMS_VALEUR + UN, model.getMinute());
	assertEquals(UN, model.getSecond());
	// 10h:59m:59s plus une seconde donne 11h:0m:0s
	controler.setHour(HMS_VALEUR);
	controler.setMinute(ClockModel.MAX_MINSEC - UN);
	controler.setSecond(ClockModel.MAX_MINSEC - UN);
	controler.incSecond(UN);
	assertEquals(HMS_VALEUR + UN, model.getHour());
	assertEquals(ZERO, model.getMinute());
	assertEquals(ZERO, model.getSecond());
	// 23h:59m:59s plus une seconde donne 0h:0m:0s
	controler.setHour(ClockModel.MAX_HOUR - UN);
	controler.setMinute(ClockModel.MAX_MINSEC - UN);
	controler.setSecond(ClockModel.MAX_MINSEC - UN);
	controler.incSecond(UN);
	assertEquals(ZERO, model.getHour());
	assertEquals(ZERO, model.getMinute());
	assertEquals(ZERO, model.getSecond());
	}

// tester la méthode incSecond(x) avec x = -1;
@Test
public void testDecSeconde() {
	// 10m:0s moins une seconde donne 9m:59s
	controler.setMinute(HMS_VALEUR); controler.setSecond(ZERO);
	controler.incSecond(NEG_UN);
	assertEquals(HMS_VALEUR-UN, model.getMinute());
	assertEquals(ClockModel.MAX_MINSEC - UN, model.getSecond());
	// 9m:59s moins une seconde donne 9m:58s
	controler.incSecond(NEG_UN);
	assertEquals(HMS_VALEUR - UN, model.getMinute());
	assertEquals(ClockModel.MAX_MINSEC - DEUX, model.getSecond());
	// 10h:0m:0s moins une seconde donne 9h:59m:59s
	controler.setHour(HMS_VALEUR); controler.setMinute(ZERO);
	controler.setSecond(ZERO);
	controler.incSecond(NEG_UN);
	assertEquals(HMS_VALEUR - UN, model.getHour());
	assertEquals(ClockModel.MAX_MINSEC - UN, model.getMinute());
```

```
            assertEquals(ClockModel.MAX_MINSEC  - UN, model.getSecond());
            // 0h:0m:0s moins une seconde donne 23h:59m:59s
            controler.setHour(ZERO); controler.setMinute(ZERO);
            controler.setSecond(ZERO);
            controler.incSecond(NEG_UN);
            assertEquals(ClockModel.MAX_HEURE - UN, model.getHour());
            assertEquals(ClockModel.MAX_MINSEC - UN, model.getMinute());
            assertEquals(ClockModel.MAX_MINSEC - UN, model.getSecond());
      }

      // tester la méthode incSecond(x) avec x choisi aléatoirement
      @Test
      public void testIncSecondeQuelconque() {
            int ajout, s, m, h, sec, min, heure ;
            for (int i = 0 ; i < MAX_TESTS; i++) {
                  s = model.getSecond();
                  m = model.getMinute();
                  h = model.getHour();
                  ajout = rd.nextInt(MAX_TESTS);
                  controler.incSecond(ajout);
                  sec = (s+ajout)%ClockModel.MAX_MINSEC;
                  min  = (m + (s+ajout)/ClockModel.MAX_MINSEC)%ClockModel.MAX_MINSEC;
                  heure = (h +
                              (m +
                                          (s+ajout)/ClockModel.MAX_MINSEC
                                          ) /ClockModel.MAX_MINSEC
                              )
                              %ClockModel.MAX_HOUR;
                  assertEquals(model.getSecond(),sec);
                  assertEquals(model.getMinute(),min);
                  assertEquals(model.getHour(),heure);
            }
      }

}
```

A rendre :

- le diagramme de classes cohérent avec le code des tests*;*
- le code du minuteur (interface utilisateur programmée avec JavaFX) correspondant au diagramme de classes qui fonctionne correctement : votre code doit passer tous les tests et surtout il doit réaliser les fonctionnalités attendues;
- le diagramme de communication cohérent avec votre diagramme de classes et votre code.

Optionnel – créer un Thread (aussi nommé processus léger ou 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.