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.

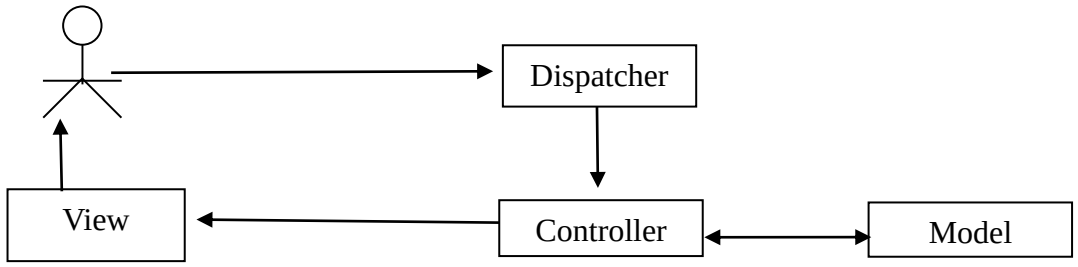
MVC est l'une des premières approches pour décrire et mettre en œuvre des architectures basées sur la notion de responsabilité. L'idée était de bien séparer les données, la présentation et les traitements des événements utilisateur. Il en résulte les trois parties :

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

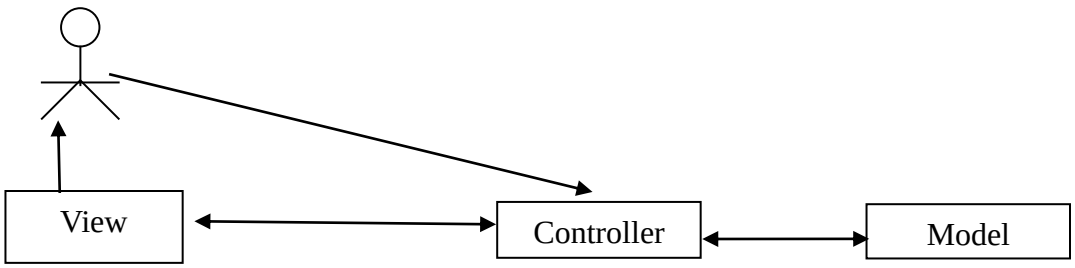
Bien que développé à l'origine pour les applications "standard", le MVC a été largement adopté comme une architecture dans le monde des applications WEB. Plusieurs « systèmes Web » commerciaux et non commerciaux ont mises en œuvre ce modèle. L'interprétation de ce modèle, principalement dans la façon 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 5h59mm59s a pour résultat 6h0mm0s ; et retirer 5 secondes à 7h00mn2s a pour résultat 6h59mm58s

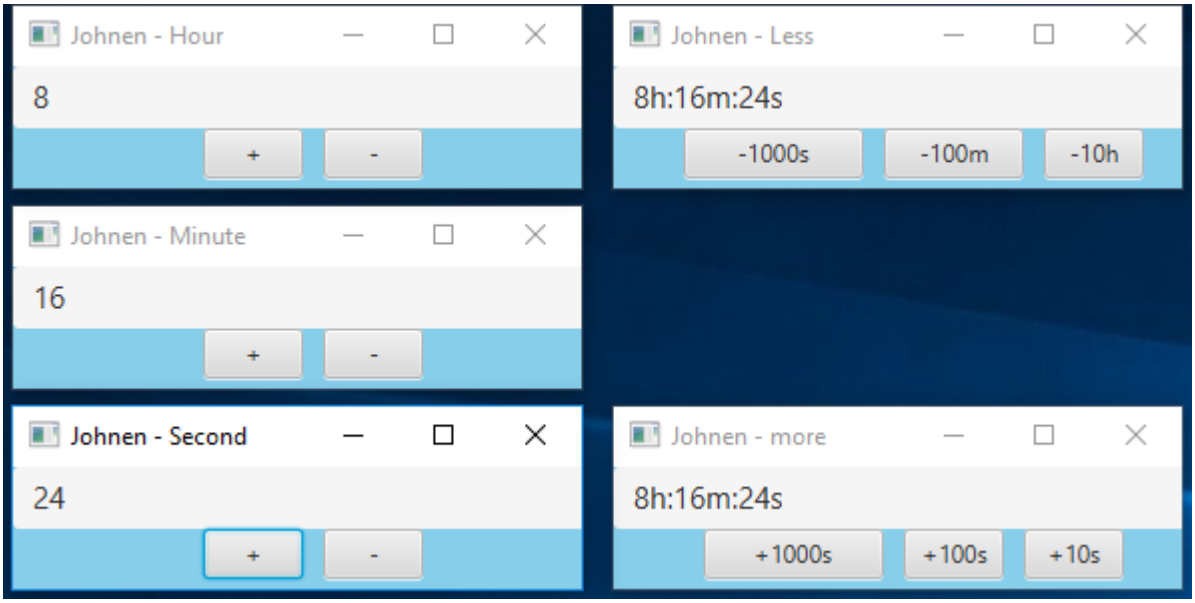
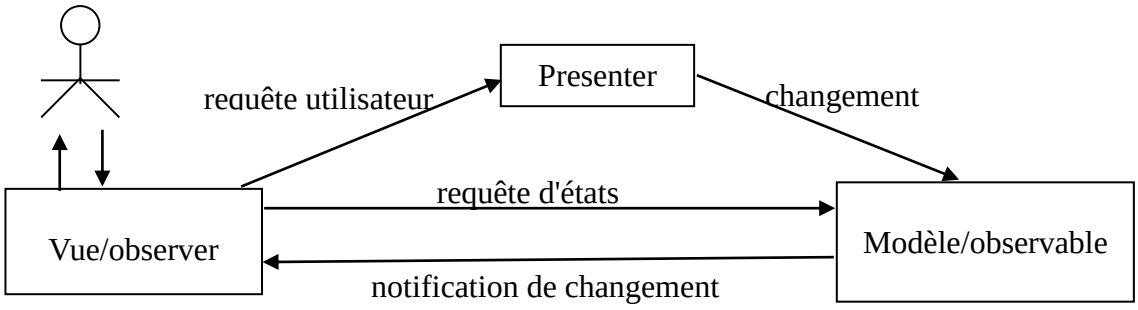


Figure : 5 interfaces graphiques de la même horloge

L'architecture logicielle sera une variante du MVC, nommé MVP. Plus précisément, l'architecture correspondra au schéma suivant.



```
public class ClockModel {
      private final IntegerProperty hour = new SimpleIntegerProperty(0);
      private final IntegerProperty minute = new SimpleIntegerProperty(0);
      private final IntegerProperty second = new SimpleIntegerProperty(0);
      public static final int MAX_HEURE = 24;
      public static final int MAX_MINSEC = 60;
      public static final int MIN = 0;

      public ClockModel(int hour, int minute, int second) {
            this.hour.set(hour);this.minute.set(minute) ;
            this.second.set(second);   }

      public int getHour() {
            System.out.println("ClockModel - getHour");  return hour.get();}

      public int getMinute() {
          System.out.println("ClockModel - getMinute"); return minute.get();}

     public int getSecond()     {
         System.out.println("ClockModel - getSecond"); return second.get(); }

      public void addHourObserver(ChangeListener<Number> cl) {
                        hour.addListener(cl);}
      public void addMinuteObserver(ChangeListener<Number> cl) {
                                  minute.addListener(cl);}
      public void addSecondObserver(ChangeListener<Number> cl) {
                                  second.addListener(cl);}

//  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 à écrire */}
}
```

Questions

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

Après lecture du code sur les pages suivantes

2. Ecrivez le code de la classe C*lockControllerM* en utilisant le patron de conception décorateur. La classe C*lockControllerM* gère directement les mises à jour des minutes. Les mises à jour des heures et des secondes sont déléguées.

```
public interface IClockController                    {
      public void setHour(int hour);
      public void incHour(int hour);
      public void setMinute(int minute);
      public void incMinute(int minute);
      public void setSecond(int seconde);
      public void incSecond(int seconde) ;           }
```

```
public abstract class AClockController implements IClockController   {
       protected ClockModel myModel;

       public AClockController (ClockModel model) { this.myModel = model; }
       public ClockModel getModel() { return myModel; }                      }


// ClockControllerEmpty gère aucune mise à jour.
public class ClockControllerEmpty extends AClockController {
       public ClockControllerEmpty(ClockModel model) { super(model); }
       @Override
       public void setHour(int hour) { }
       @Override
      public void incHour(int hour) { }
       @Override
        public void setMinute(int minute) { }
      @Override
      public void incMinute(int minute) { }
       @Override
      public void setSecond(int second) { }
       @Override
      public void incSecond(int second) { }                                 }


// ClockControllerDecorator délègue toutes les mises à jours à l'objet décoré.
//C'est un décorateur qui ne fait rien.
//Cette classe implémente le patron de conception « décorateur » :
public class ClockControllerDecorator extends AClockController {
       protected AClockController myController;

       public ClockControllerDecorator(AClockController controller) {
                    super(controller.getModel());
                    myController = controller; }

       public ClockControllerDecorator(ClockModel model) {
                    super(model);
                    this.myController = this;                    }

       @Override
       public void setHour(int hour) { myController.setHour(hour);  }
      @Override
      public void incHour(int hour) { myController.incHour(hour);  }
      @Override
      public void setMinute(int minute) { myController.setMinute(minute); }
      @Override
      public void incMinute(int minute) { myController.incMinute(minute); }
       @Override
      public void setSecond(int second) { myController.setSecond(second); }
       @Override
      public void incSecond(int second) { myController.incSecond(second); }  }
```

```
// ClockControllerH gère directement les mises à jour des heures,
// les mises à jours des minutes et des secondes sont déléguées.
/Cette classe implémente le patron de conception « décorateur »
public class ClockControllerH extends ClockControllerDecorator {

      public ClockControllerH(AClockController controller) { super(controller); }

      @Override
      public void setHour(int hour){
            int h = hour % ClockModel.MAX_HOUR;
            if (h < ClockModel.MIN_TIME) h = h + ClockModel.MAX_HOUR;
            try {  myController.getModel().setHour(h);}
            catch (Exception e) {System.out.println(e);}    }

     @Override
     public void incHour(int hour){
                  int hh = myController.getModel().getHour();
                  this.setHour(hh+hour); }                    }
```

Questions

3. A la lecture du code suivant, que gèrent les objets *cc1*, *cc2*, *cc3*, *cc4*, *cc5 et cc6* ?

```
ClockModel model1, model2;
model1 = new ClockModel();
model2 = new ClockModel();
AClockController cc1, cc2, cc3, cc4, cc5, cc6;
cc1 = new ClockControllerEmpty(model1);
cc2 = new ClockControllerH(cc1);
cc3 = new ClockControllerM(cc2);
cc4 = new ClockControllerEmpty(model2);
cc5 = new ClockControllerM(cc4);
cc6 = new ClockControllerH(cc5);
```

4. A la lecture du code suivant, que gère un objet de la classe ClockController ? Un objet de la classe *ClockControllerS* gère directement les mise à jour des secondes. Les mises à jour des heures et des minutes sont déléguées.

```
public class ClockController extends ClockControllerDecorator {

      public ClockController(ClockModel model) {
            super(model);
            AClockController c1 = new ClockControllerEmpty(model);
            AClockController c2 = new ClockControllerH(c1);
            AClockController c3 = new ClockControllerM(c2);
            myController = new ClockControllerS(c3);   }            }
```

5. Ecrivez le diagramme de classes correspondant à l'application vous devez utiliser le patron de conception Property-Observable/Property-Observer, ainsi que le patron de conception décorateur dans le cadre d'une architecture logicielle variante du MVC, nommé MVP

6. Elaborez le diagramme de communication associé au scénario suivant :
Après que l'application soit lancée, l'horloge a la valeur 8h :16m :24s, l'utilisateur augmente la valeur du nombre d'heures via le bouton « plus » de la fenêtre « Heure », les affichages sont mises à jour (il y a 3 fenêtres d'affichage) : 9h :16m :24s.

7. Complétez les commentaires des méthodes de la classe *ClockScenariosTest.java*.

```
public class ClockScenariosTest {
	private static int HMS_VALEUR = 10;
	private static int ZERO = 0;
	private static int UN = 1;
	private static int MAX_TESTS = 1000;
	private ClockModel model;
	private ClockController  controller;
	private 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();		}


	@Test  // test la méthode incHour(x) avec x positif
	public void scenariosIncHour() {
		/* 23h plus une heure = 0h */
		controller.setHour(ClockModel.MAX_HOUR  -  UN);
		TestClock.testIncHour(model, controller, UN);
		//

		TestClock.testIncHour(model, controller, UN);
		//

		TestClock.testIncHour(model, controller, ClockModel.MAX_HOUR);

		//tests aleatoires
		for (int i = 0 ; i < MAX_TESTS; i++) {
			TestClock.testIncHour(model, controller, rd.nextInt(MAX_TESTS));
		}  }

	@Test // test la méthode incHour(x) avec x  negatif
	public void scenariosDecHour() {
		/* 0h moins une heure = 23h */
		controller.setHour(ZERO);
		TestClock.testDecHour(model, controller, UN);
		//

		TestClock.testDecHour(model, controller, UN);
		//

		TestClock.testDecHour(model, controller, ClockModel.MAX_HOUR);
```

```
        //tests aleatoires
        for (int i = 0 ; i < MAX_TESTS; i++) {
              TestClock.testDecHour(model, controller, rd.nextInt(MAX_TESTS));
        }     }


@Test  // test la méthode incMinute(x) avec x positif
public void scenariosIncMinute() {
        // 10h:59m plus une minute = 11h:0m
        controller.setHour(HMS_VALEUR);
        controller.setMinute(ClockModel.MAX_MINSEC - UN);
        TestClock.testIncMinute(model, controller, UN);
        //

        TestClock.testIncMinute(model, controller, UN);
        //

        TestClock.testIncMinute(model, controller,
                        ClockModel.MAX_MINSEC*ClockModel.MAX_HOUR);

        //tests aleatoires
        for (int i = 0 ; i < MAX_TESTS; i++) {
              TestClock.testIncMinute(model, controller, rd.nextInt(MAX_TESTS));
        }     }


@Test  // test la méthode incMinute(x) avec x negatif
public void scenariosDecMinute() {
        // 10h:0m moins une minute = 9h:59m
        controller.setHour(HMS_VALEUR); controller.setMinute(ZERO);
        TestClock.testDecMinute(model, controller, UN);
        //

        TestClock.testDecMinute(model, controller, UN);
        controller.setHour(ZERO);
        controller.setMinute(ZERO);
        //

        TestClock.testDecMinute(model, controller, UN);
        //

        TestClock.testDecMinute(model,
               controller,ClockModel.MAX_MINSEC*ClockModel.MAX_HOUR);

        //tests aleatoires
        for (int i = 0 ; i < MAX_TESTS; i++) {
              TestClock.testDecMinute(model, controller, rd.nextInt(MAX_TESTS));
        }   }
```

```
	@Test  // test la méthode incSecond(x) avec x positif
	public void scenariosIncSecond() {
		// 10m:59s plus une seconde = 11m:0s
		controller.setMinute(HMS_VALEUR);
		controller.setSecond(ClockModel.MAX_MINSEC - UN);
		TestClock.testIncSecond(model, controller, UN);
		//

		TestClock.testIncSecond(model, controller, UN);
		//

		controller.setHour(HMS_VALEUR);
		controller.setMinute(ClockModel.MAX_MINSEC - UN);
		controller.setSecond(ClockModel.MAX_MINSEC - UN);
		TestClock.testIncSecond(model, controller, UN);
		//

		controller.setHour(ClockModel.MAX_HOUR - UN);
		controller.setMinute(ClockModel.MAX_MINSEC - UN);
		controller.setSecond(ClockModel.MAX_MINSEC - UN);
		TestClock.testIncSecond(model, controller, UN);
		//

		TestClock.testIncSecond(model, controller,
			ClockModel.MAX_MINSEC*ClockModel.MAX_MINSEC*ClockModel.MAX_HOUR);  }


	@Test  // test la méthode incSecond(x) avec x  negatif
	public void scenariosDecSecond() {
		// 10m:0s moins une seconde = 9m:59s
		controller.setMinute(HMS_VALEUR); controller.setSecond(ZERO);
		TestClock.testDecSecond(model, controller, UN);
		//

		TestClock.testDecSecond(model, controller, UN);
		//

		controller.setHour(HMS_VALEUR); controller.setMinute(ZERO);
		controller.setSecond(ZERO);
		TestClock.testDecSecond(model, controller, UN);
		//

		controller.setHour(ZERO); controller.setMinute(ZERO);
		 controller.setSecond(ZERO);
		TestClock.testDecSecond(model, controller, UN);
		//

		TestClock.testDecSecond(model, controller,
			ClockModel.MAX_MINSEC*ClockModel.MAX_MINSEC*ClockModel.MAX_HOUR);
		//

		TestClock.testDecSecond(model, controller,
			ClockModel.MAX_MINSEC*ClockModel.MAX_MINSEC*(ClockModel.MAX_HOUR-UN));
		//

		TestClock.testDecSecond(model, controller,
				ClockModel.MAX_MINSEC*ClockModel.MAX_MINSEC); }
}
```

A rendre – Règles de présentation - un document ne suivant pas ces règles ne sera pas corrigé -
Sur toutes le feuilles, il faut votre nom, le nom du groupe ; il y a au plus un diagramme par feuille ; chaque diagramme est nommé ; les diagrammes sont lisibles et le titre des fenêtres inclus votre nom

- le diagramme de classes cohérent avec le code des tests, l'architecture MVP.
- le diagramme de communication « Après que l'application soit lancée, l'horloge a la valeur 8h :16m :24s, l'utilisateur augmente la valeur du nombre d'heures via le bouton « plus » de la fenêtre « Heure », les affichages sont mises à jour (il y a 3 fenêtres d'affichage) : 9h :16m :24s». 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 du paquet clockTest) et surtout il doit réaliser les fonctionnalités attendues.

  Dans Moodle, vous trouverez le projet Eclipse JEE à compléter.