Modèle-Vue-Contrôleur

Thierry CHAMPION

Modèle-Vue-Contrôleur

Définitions

- L ’architecture Model-View-Controller a pour objectif d ’organiser une application interactive en séparant :
  - les données
  - la représentation des données
  - le comportement de l ’application
- Créé dans la fin des années 1970 par Trygve Reenskaug au Xerox PARC
- Applications aux GUI (Graphical User Interfaces)
- Première version en 1980 utilise une sous classe pour chaque vue à adapter au modèle

Modèle-Vue-Contrôleur

Structures de MVC

- Le **modèle** représente la structure des données dans l ’application (état) et les opérations spécifiques sur ces données.

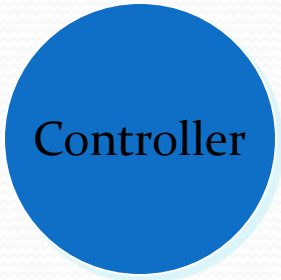
- Une **Vue** présente les données sous une certaine forme à l ’utilisateur, suivant un contexte d’exploitation.

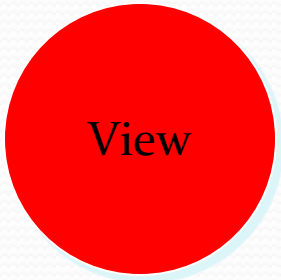
- Un **Controller** traduit les interactions utilisateur par des appels de méthodes (comportement) sur le modèle et sélectionne la vue appropriée basée sur l’état du modèle.

Modèle-Vue-Contrôleur

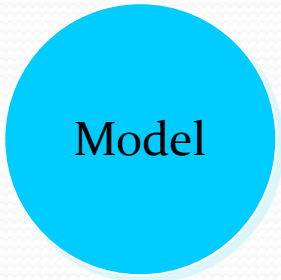
Structures de MVC

Controllers gèrent les entrées de l'utilisateur





Views présentent l'information à l'utilisateur



Models implémentent les fonctionnalités et l'état de l'application

Modèle-Vue-Contrôleur

Structures de MVC

Avantages :

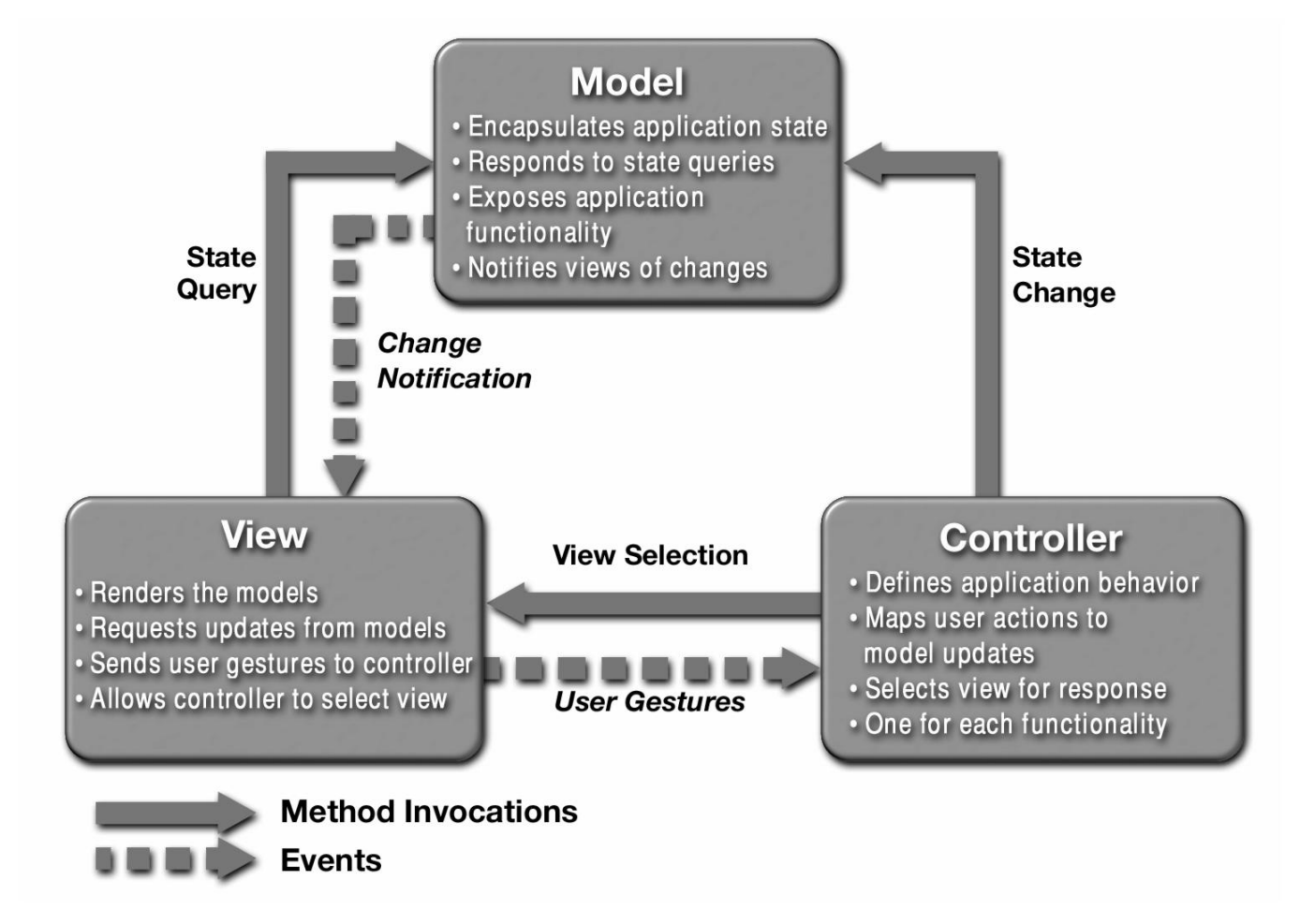
- Structure Orientée-Objet propre,
- Vues Multiples d'un même modèle,
- Vues Synchronized,
- views et controllers inter-changeables,
- Look and Feel modifiable,
- Framework Potentiel.

Inconvénients :

- Complexité accrue
- View/Controller fortement liés au Modèle

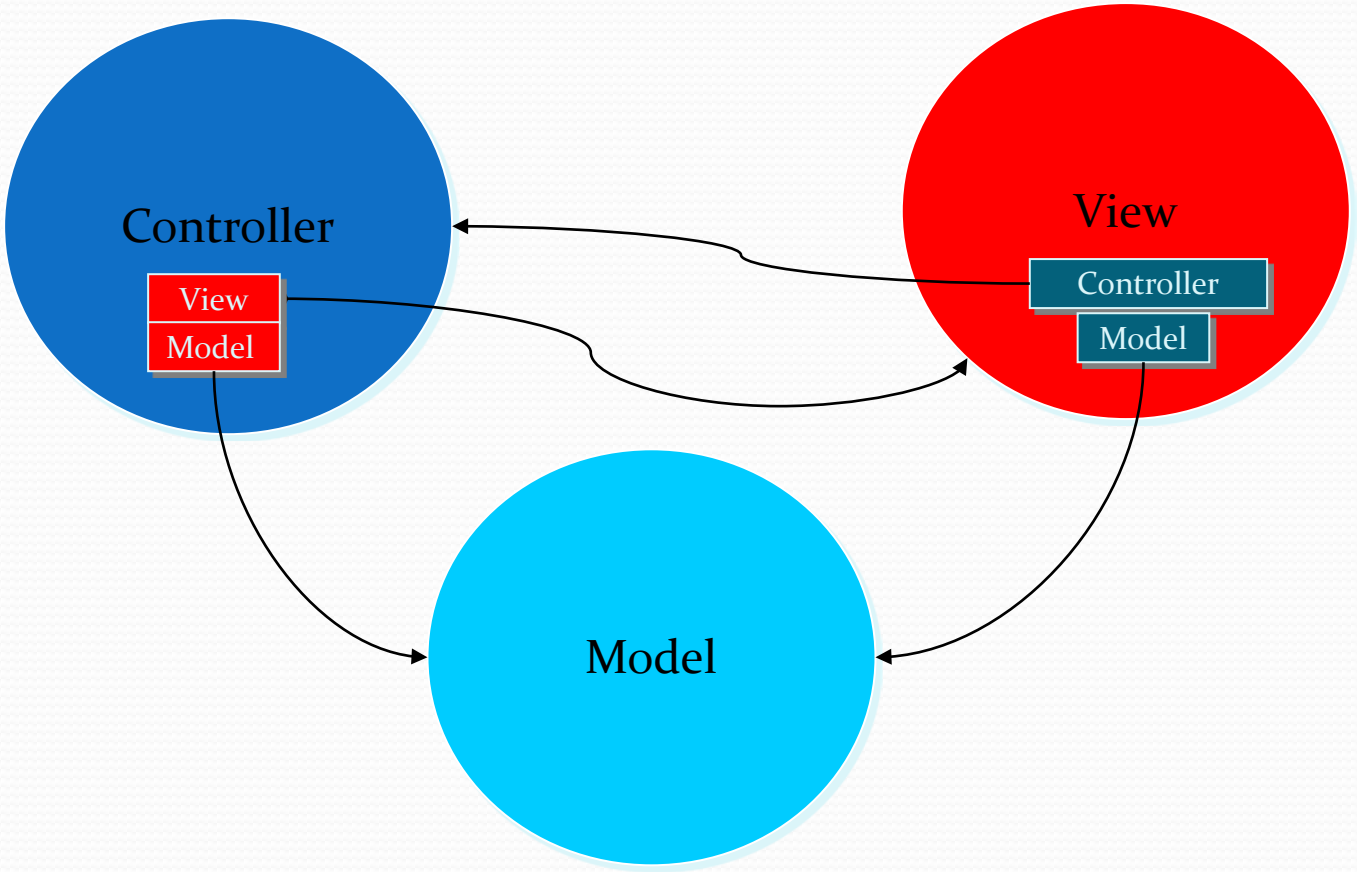
Modèle-Vue-Contrôleur

Architecture du MVC



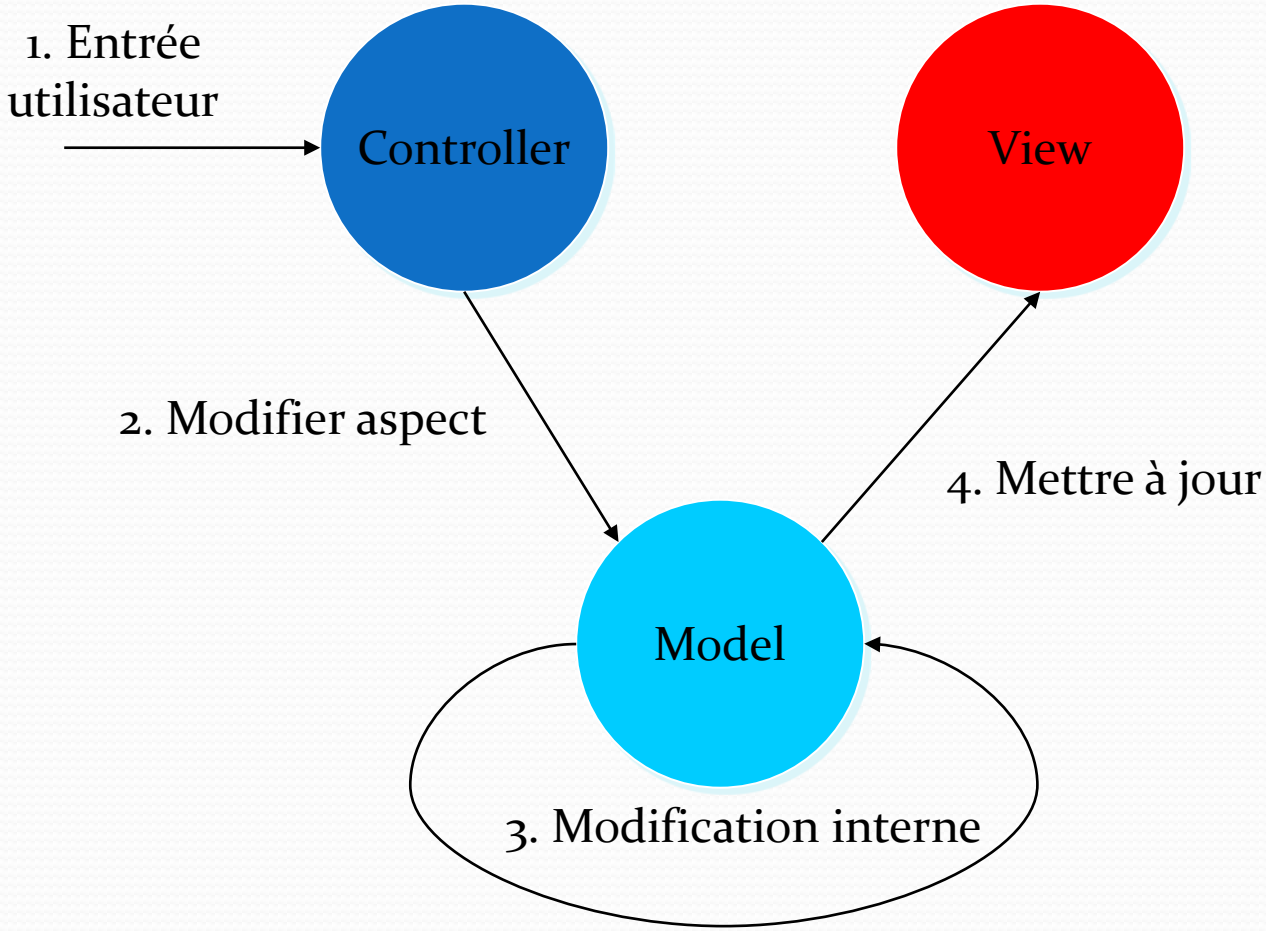
Modèle-Vue-Contrôleur

Architecture du MVC : références croisées



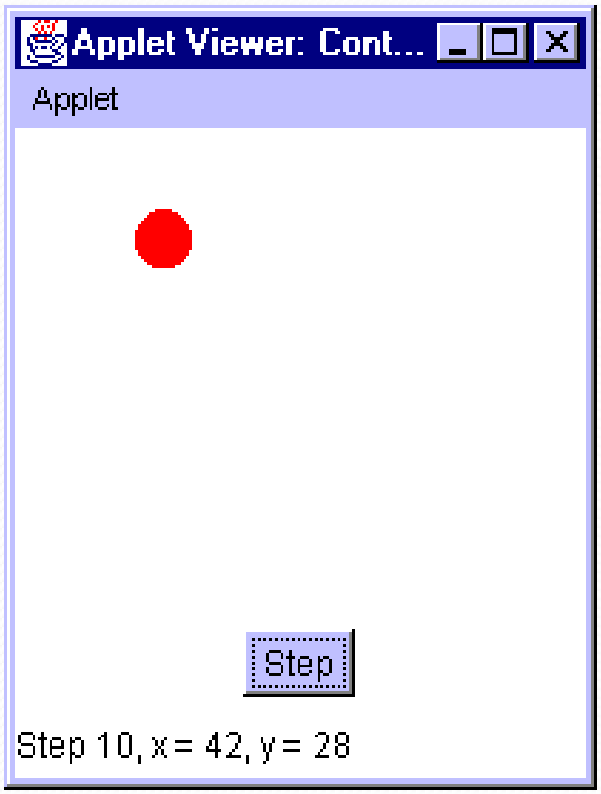
Modèle-Vue-Contrôleur

Architecture du MVC : exemple de fonctionnement



Modèle-Vue-Contrôleur

Exemple de conception MVC : Applet de la balle rebondissante



- Chaque clic sur le bouton Step fait un peu avancer la balle,
- Le numéro de clic et la position de la balle sont affichés dans la barre de statuts.

Modèle-Vue-Contrôleur

Applet de la balle rebondissante : Model

- L'applet de la balle montre une balle rebondissante dans une fenêtre.
- Le modèle va contrôler la position et le mouvement de la balle.
- Dans cet exemple, le modèle doit connaître la taille de la fenêtre
  - Ainsi il peut savoir quand la balle doit rebondir
- Le modèle n'a pas besoin de connaître autre chose de l'interface graphique

Modèle-Vue-Contrôleur

Observer et Observable

- Le package java.util contient l'interface Observer et la classe Observable qui vont être utiles pour implanter le pattern MVC :
  - Une instance de Observable est un objet qui peut être « observé ».
  - Une instance de Observer est « notifié » quand un objet qu'il est en train d'observer annonce un changement.

- Analogie possible :
  - Un Button agit comme un Observable,
  - Un ActionListener agit comme un Observer,
  - L'ActionListener doit être attaché au Button pour pouvoir l'observer.

- Autre Analogie :
  - Un Observable est comme un bulletin météo,
  - Un Observer est comme quelqu'un qui lit ou écoute ce bulletin météo.

Modèle-Vue-Contrôleur

Observer et Observable

- Une instance de Observable est un objet qui peut être « observé ».
- Une instance de Observer est « notifié » quand un objet qu'il est en train d'observer annonce un changement.

  - Quand un Observable veut annoncer qu'il a changé son état, il doit exécuter :

    setChanged() ;

    notyfyObservers( ) ; ou notifyObservers(arg) ;

    - arg est de type Object (donc n'importe quel objet)

  - L'objet Observable ne sait pas et ne s'occupe pas de qui est entrain de l'observer.

  - Mais tous les objets Observer doivent au préalable avoir été attachés à l'objet Observable avec :

    myObservable.addObserver(myObserver) ;

Modèle-Vue-Contrôleur

Observer

- Observer est une interface.
- Une classe implémentant l'interface Observer doit définir la méthode :

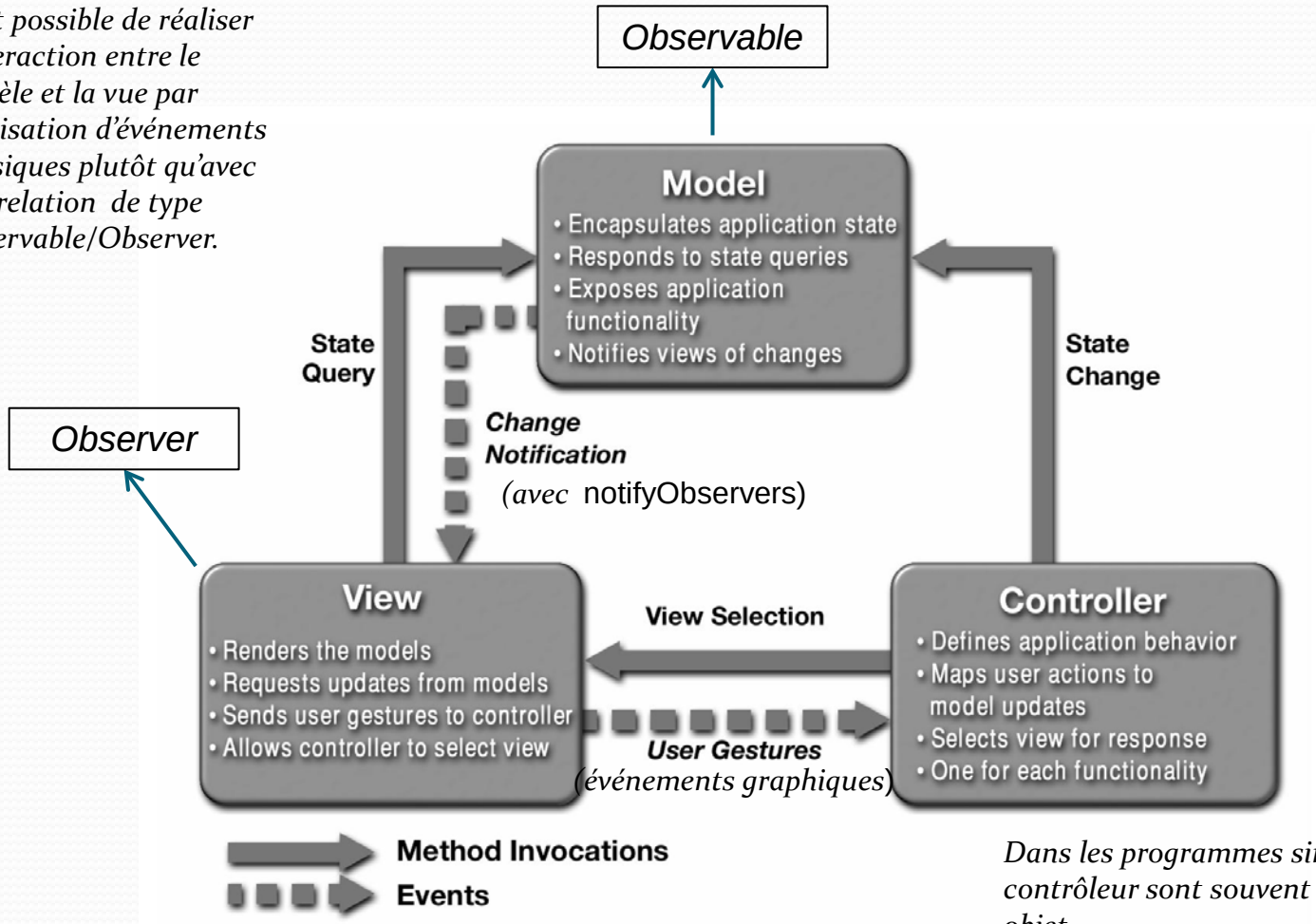
```
public void update(Observable obs, Object arg)
```

  - Cette méthode est invoquée lorsque un objet Observable, observée par l'objet Observer en question, exécute la méthode addNotify() ou addNotify(arg),
  - L'argument obs est une référence sur l'objet Observable à l'origine de la notification,
  - Si l'objet Observable a exécuté addNotify(), le paramètre arg est égal à null, sinon il prend la valeur de l'argument passé à addNotify(arg).

Modèle-Vue-Contrôleur

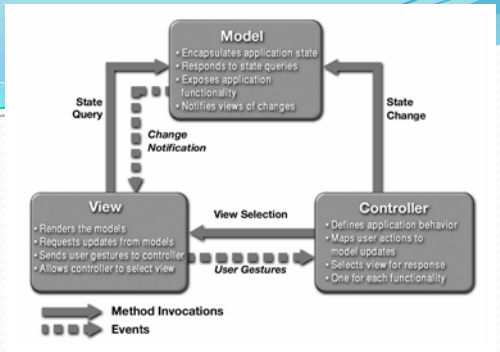
Architecture du MVC : Le modèle est observé par la vue, le Controller écoute les événements de l'interface

Il est possible de réaliser l'interaction entre le modèle et la vue par l'utilisation d'événements classiques plutôt qu'avec une relation de type Observable/Observer.



Dans les programmes simples la vue et le contrôleur sont souvent représentés par un seul objet.

Modèle-Vue-Contrôleur



Applet de la balle rebondissante : Model

- Le modèle va contrôler la position et le mouvement de la balle.
  - Dans cet exemple, le modèle doit donc connaître la taille de la fenêtre
    - Ainsi il peut savoir quand la balle doit rebondir
  - Le modèle n'a pas besoin de connaître autre chose de l'interface graphique (en dehors de ce qu'il représente lui-même, c'est-à-dire la balle)

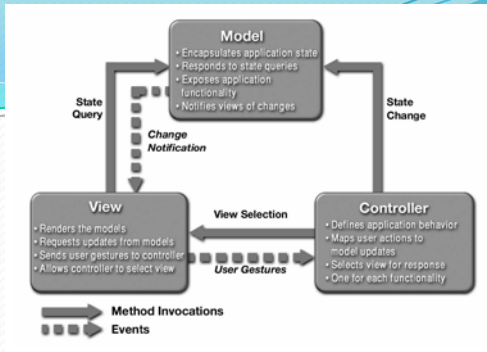
- Le modèle doit :
  - Etre un Observable (hériter de…) pour pouvoir être observé par la vue
  - Définir la position initiale de la balle et savoir gérer ses mouvements
  - Fournir des méthodes d'accès à la vue pour que celle-ci puisse connaître son état (donc la position de la balle)
  - Fournir des méthodes de changement au contrôleur pour que celui-ci puisse lui transmettre les actions de l'utiilisateur

Modèle-Vue-Contrôleur

Applet de la balle rebondissante : Model

```
import java.util.Observable;

public class Model extends Observable {
   final int BALL_SIZE = 20 ;
   int xPosition = 0 ; int yPosition = 0 ;
   int xLimit, yLimit ;
   int xDelta = 6 ; int yDelta = 4 ;
   int stepNumber = 0 ;

   // methodes pour le controller
   public void start( int width, int height) {
      xlimit = width – BALL_SIZE ; ylimit = height – BALL_SIZE ;
      stepNumber = 0 ;
      setChanged() ; notifyObservers() ;
   }
   public void makeOneStep( ) {
      xPosition += xDelta ;
      if (xPosition < 0 || xPosition >= xLimit) { xDelta = -xDelta ; xPosition += xDelta ; }
      yPosition += yDelta ;
      if (yPosition < 0 || yPosition >= yLimit) { yDelta = -yDelta ; yPosition += yDelta ; }
      setChanged() ; notifyObservers() ;
   }

   // méthodes pour la vue
   public getXPosition( ) { return xPosition ; }
   public getYPosition( ) { return yPosition ; }
   public getBallSize( ) { return BALL_SIZE ; }
   public getStepNumber ( ) { return stepNumber ; }

} // end of Model class
```

Modèle-Vue-Contrôleur



Applet de la balle rebondissante : View

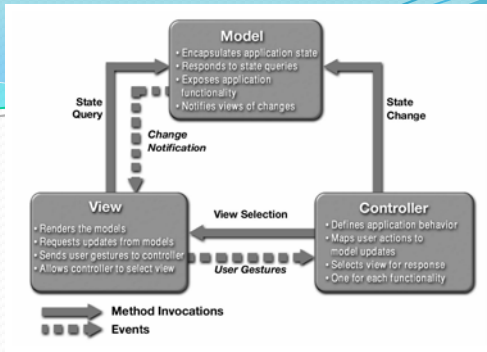
- La vue gère la représentation graphique pour l'utilisateur.
- La vue doit :
  - Créer et afficher les différents composants graphiques nécessaires,
  - Mettre à jour ses composants (affichage de la balle, barre de statut) lorsque le statut du modèle (la position de la balle) change :
    - Pour cela elle doit accéder au modèle pour récupérer les informations nécessaires (les coordonnées et la taille de la balle, ainsi que le numéro de « step »).
    - Elle doit aussi accéder au contrôleur qui dans le cas d'une Applet gère la barre de statut.

Modèle-Vue-Contrôleur

Applet de la balle rebondissante : View

```
import java.awt.* ; import java.awt.event.* ; import java.util.* ;

public class View extends Panel implements Observer {
   Controller controller ;
   Model model ;

    public View (Model model, Controller controller ) {
        this.model = model ; this.controller =controller ;
        JPanel buttonPanel = new JPanel() ;
        JButton stepButton = new JButton("Step") ;
        stepButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                controller.stepEvent( ) ;
            }
        }) ;
        canvas = new Canvas( ) {
            public void paint(Graphics g) {
                g.setColor(Color.red) ;
                g.fillOval( model.getXPosition(), model.getYPosition(), model.getBallSize(), model.getBallSize()) ;
                controller.showStatus("Step " + (model.getStepNumber()) + ", x = " + model.getXPosition( )
                                                                        + ", y = " + model.getYPosition() ) ;
            }
        }) ;
        setLayout(new BorderLayout()) ;
        buttonPanel.add(stepButton) ;
        this.add(BorderLayout.SOUTH, buttonPanel);
        this.add(BorderLayout.CENTER, canvas);
    }
    public void update(Observable obs, Object arg) { repaint( ) ; }
}
```

Modèle-Vue-Contrôleur



Applet de la balle rebondissante : Controller

- Le contrôleur construit l'application et gère l'ensemble des interactions avec l'application.
- Le contrôleur doit :
  - Créer le modèle,
  - Créer la vue,
  - Donner à la vue un accès au modèle et au controleur,
  - Gérer les événements utilisateurs :
    - Clic sur le bouton Step => Demander au modèle d'avancer

Modèle-Vue-Contrôleur

Applet de la balle rebondissante : Controller

```
import java.applet.* ;
import java.awt.* ;
import java.awt.event.* ;
import java.util.* ;

public class Controller extends JApplet {
   Model model ;
   View view  ;

   public Controller () {
       // Création du modèle et de la vue etdes liaisons entre eux
        model = new Model( ) ;
        view = new View( model, this ) ;
        model.addObserver( view ) ;

       // affichage de la vue dans l'applet
        this.add ( view ) ;

       // lancement du modèle (provoque un affichage de la vue via le modèle
        model.start( vew.getSize( ).width, view.getSize( ).height ) ;
    }

    public stepEvent( ) {
        model.makeOneStep( ) ;
    }

    public void showStatus( String texte ) {
        super.showStatus(texte) ;
    }
}
```

Modèle-Vue-Contrôleur

Application de la balle rebondissante : Controller

```
import java.awt.* ;
import java.awt.event.* ;
import java.util.* ;

public class Controller extends Frame {
   Model model ;
   View view  ;
   Label label ;

   public Controller () {
       // Création du modèle et de la vue etdes liaisons entre eux
        model = new Model( ) ;
        view = new View( model, this ) ;
        model.addObserver( view ) ;

       // affichage de la vue dans l'application, avec ajout d'une barre de statut
        this.setLayout( new BorderLayout() ) ;
        this.add ( view , BorderLayout.CENTER) ;
        this.add ( label = new Label() , BorderLayout.SOUTH) ;

       // lancement du modèle (provoque un affichage de la vue via le modèle) puis afficage de l'application
        model.start( vew.getSize( ).width, view.getSize( ).height ) ;
        this.pack() ; this.setVisible(true) ;
    }

    public stepEvent( ) { model.makeOneStep( ) ; }

    public void showStatus( String texte ) { label.setText(texte) ; }
}
```

Modèle-Vue-Contrôleur

Points clés

- Le modèle contient la partie « intelligente » du programme
  - Il ne doit pas contenir d'entrée sortie (rôle du contrôleur),
  - La communication avec le modèle se fait exclusivement avec des méthodes (encapsulation),
  - Cette approche procure le maximum de flexibilité pour l'utilisation du modèle (possibilité de faire évoluer la vue, les interactions utilisateurs, … , sans changer l'intelligence du programme).
- Le contrôleur organise le programme et procure les entrées au modèle.
- La vue affiche ce qui se passe dans le modèle
  - Elle ne doit jamais afficher ce qui doit arriver dans le modèle avant que cela ne soit fait
  - Par exemple si on doit enregistrer un fichier , la vue ne doit pas indiquer elle-même que le fichier a été enregistré, elle doit juste dire ce que le modèle lui a indiqué.
- Le contrôleur et la vue sont souvent combinés, en particulier dans les petits programmes