

Séance 8

Modèle-vue-contrôleur (MVC) Interfaces : innovations et évolutions

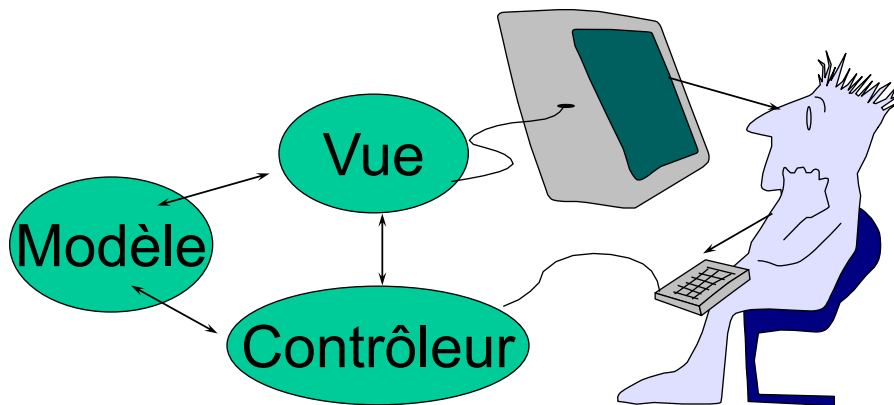
INF 753



Contenu:

- Modèle-vue-contrôleur (MVC)
- Interfaces : innovations et évolution récente

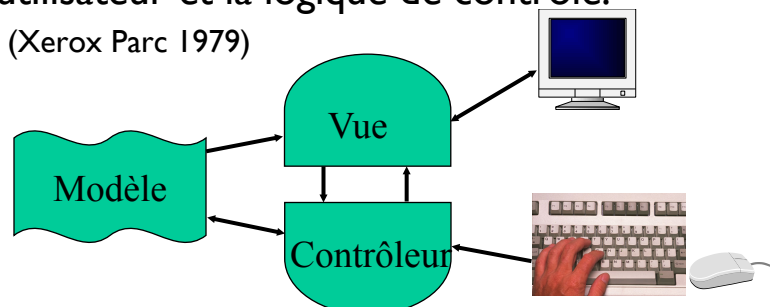
Modèle-vue-contrôleur (MVC)



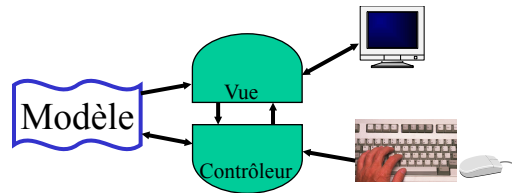
Modèle-vue-contrôleur

- Méthode de conception (*design pattern*) pour le développement d'applications logicielles qui sépare la logique du code en 3 parties : le modèle de données, l'interface utilisateur et la logique de contrôle.

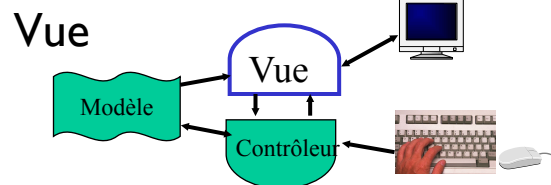
(Xerox Parc 1979)



Modèle

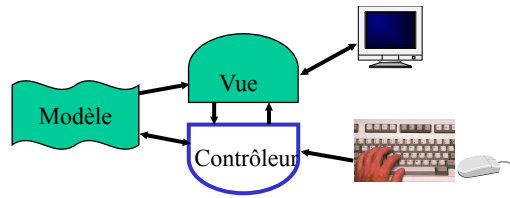


- L'information brute que manipule l'application
- Les données représentant les objets (BD)
- *Contrôleur manipule le modèle*
- *Modèle met à jour la vue*



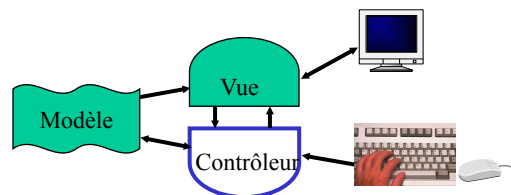
- Implémente l'affichage (la présentation visuelle) du modèle
- Possibilité de vues multiples sur un même modèle (e.g. vue graphique et vue numérique)
- Lorsque le modèle change, informer les vues
- *Vue mise à jour par le modèle*
- *Vue perçue par l'utilisateur*
- *Contrôleur agit sur la vue*

Contrôleur



- Reçoit tous les événements d'entrées en provenance de l'utilisateur
- Décide de leur sens et de quoi en faire (la logique à associer à tel clic sur un bouton, tel élément sélectionné dans une liste)
- *Utilisateur utilise le contrôleur*
- *Contrôleur manipule le modèle*
- *Contrôleur agit sur la vue*

Communication du contrôleur



- Le contrôleur communique avec la vue
 - détermine les objets qui ont été manipulés
 - Ex: quel objet a été sélectionné par un clic de souris
- Appelle les méthodes du modèle pour procéder aux changements
 - le modèle fait les changements et notifie les vues pour leurs mises à jour

Pourquoi le MVC?

Dans les applications simples, modèle, vue et contrôleur sont souvent mis dans une même classe ou dans des variables globales.

Cette approche est à éviter dans des applications plus complexes*. Pourquoi?

Avantages du MVC

- Un modèle peut avoir plus d'une vue
 - Chacune différente et nécessitant une mise à jour lors des changements dans le modèle (ex. types d'affichage des fichiers dans Windows; vue graphique ou numérique de données comptables; etc.)
 - Différents modes d'accès aux données (Web/mobile)

* Plusieurs applications Web notamment (PHP, JSP) combinent des éléments de présentation et de traitement

Pourquoi le MVC?

(suite)

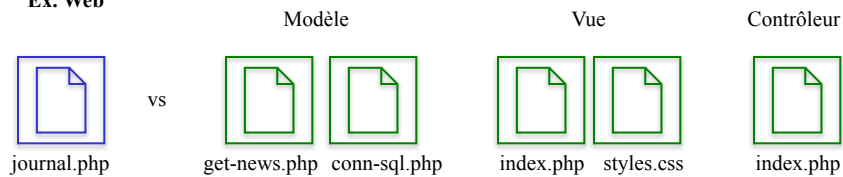
- La séparation facilite la maintenance
 - Facile d'ajouter une nouvelle vue dans le futur
 - L'ajout d'informations dans le modèle permet aux anciennes vues de continuer à fonctionner
 - Facile de changer une vue dans le futur (Ex.: *skins*, internationalisation, accessibilité)
- Un MVC permet à des équipes de développer en parallèle
 - Designers / infographistes travaillent l'interface pendant que programmeurs travaillent le modèle
- MVC supporte bien une approche préconisée aujourd'hui :
Réutilisez pour concevoir et concevez pour réutiliser

Pourquoi le MVC?

Inconvénients du MVC

- Complexe pour les petites applications (temps et rentabilité)
- 3 boîtes noires = plus de travail en amont
 - Nécessité de tester chaque élément
 - Bien concevoir les interactions entre modèle et vue et être attentionné aux détails
 - Débogage parfois plus difficile dû à la séparation MVC
- Multiplication du nombre de composants à gérer (fichiers)

Ex. Web



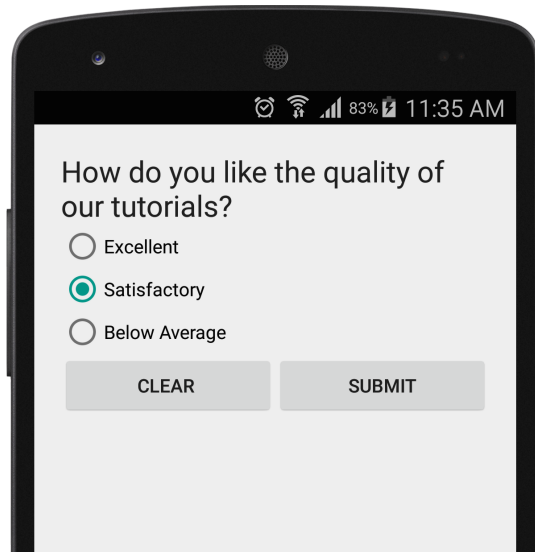
Widgets

- Le MVC s'observe notamment lorsque nous disposons de *widgets* (composantes d'interfaces réutilisables) dans un développement
- Avantages:
 - Réutilisation d'un effort de développement précédent (code, débogage, tests, itération...)
 - Cohérence externe (compatibilité avec ce qui se fait ailleurs)
- Désavantages:
 - Contraintes de design pour le concepteur
 - Dicte le style de menus, d'interactions
 - Peuvent être incorrectement utilisés

<http://www.axure.com/support/download-widget-libraries>

<http://jqueryui.com/demos/>

Widgets



How do you like the quality of our tutorials?

☐ Excellent

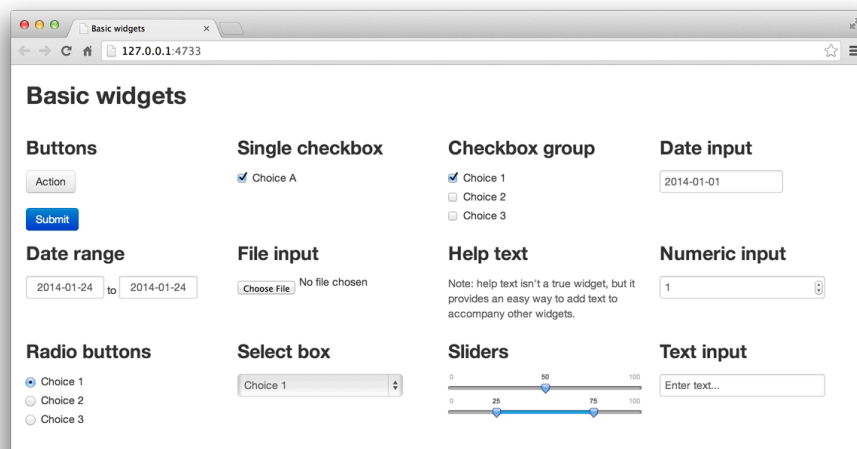
☒ Satisfactory

☐ Below Average

CLEAR SUBMIT

Android

Widgets



Basic widgets

Buttons

Action

Submit

Date range

2014-01-24 to 2014-01-24

Radio buttons

☒ Choice 1

☐ Choice 2

☐ Choice 3

Single checkbox

☒ Choice A

File input

Choose File No file chosen

Select box

Choice 1

Checkbox group

☒ Choice 1

☐ Choice 2

☐ Choice 3

Help text

Note: help text isn't a true widget, but it provides an easy way to add text to accompany other widgets.

Sliders

0 50 100

0 25 75 100

Date input

2014-01-01

Numeric input

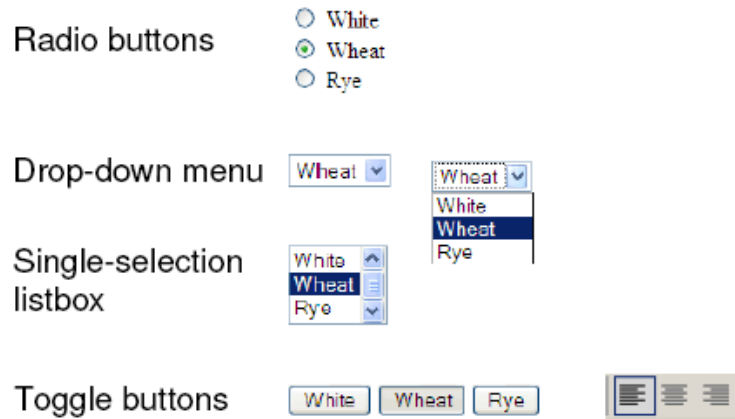
1

Text input

Enter text...

Shiny app

Widgets : Sélection unique et choix multiples



Widgets : Sélection unique et choix multiples

- Boutons radios : sélection rapide
- *Toggle* : Occupent plus d'espace = Cibles faciles à atteindre (loi de Fitts). Utiles pour les changements de mode, ou les barres d'outils
- Menu déroulant : compact / accès clavier. Mais nécessitent au moins 2 clics et éventuellement une barre de défilement

Sélection multiple : lequel choisiriez-vous?

N checkboxes

☒ Lettuce
☐ Tomato
☒ Onion
☐ Pickle

Multiple-selection
listbox

Lettuce
Tomato
Onion
Pickle

Two listboxes

Chosen Toppings: All Toppings:

Lettuce Lettuce
Onion Tomato
 Onion
 Pickle

<- Add Remove->

- Boutons radios et cases à cocher :
aligner verticalement

Pay Period

☒ Hourly
☐ Weekly
☐ Salary

Pay Period

☒ Hourly ☐ Weekly ☐ Salary

- Prévoir des libellés descriptifs, clairs, court, groupés, encadrés (avec un libellé pour le groupement).
- Limiter les options à 6 choix pour les boutons radios et 10 pour les cases à cocher. Au-delà, considérer une liste.

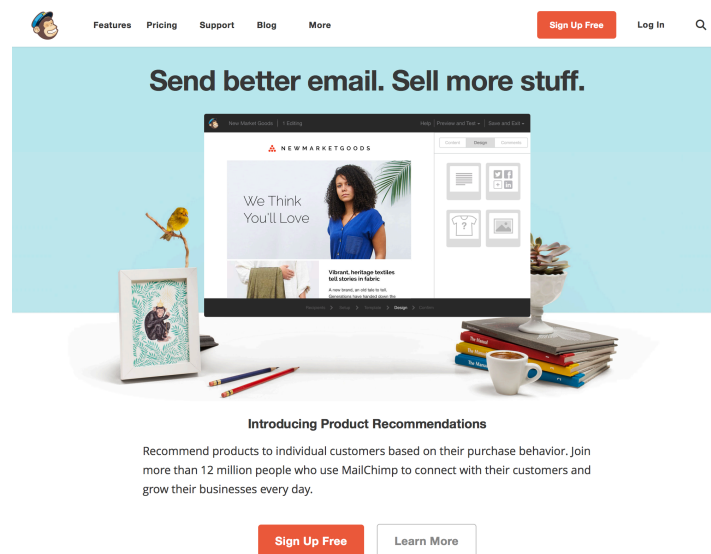
Boutons radios et cases à cocher

Choisir le meilleur ordre d'affichage des options

Comment ?

- Par fréquence : placer les options les plus fréquentes en haut
- Par tâche : s'il y a un ordre habituel dans lequel les parties d'une tâche sont exécutées.
- Logiquement : s'il existe un ordre logique (par exemple, une liste de dates)
- Par ordre alphabétique : utiliser cette méthode uniquement si les libellés s'apparient à la façon dont les utilisateurs considèrent les options.

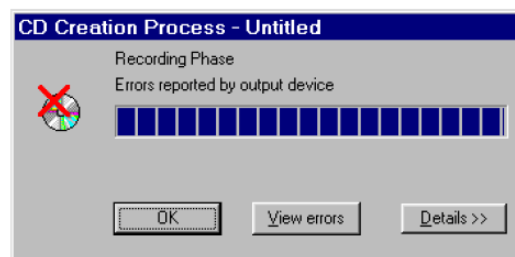
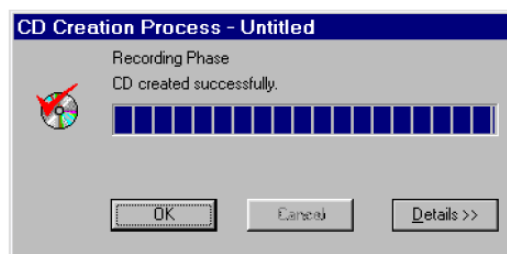
Squint test pour évaluer la perception des messages



Squint test pour évaluer la perception des messages



Squint test pour évaluer la perception des messages



Cadriciels web selon le MVC

[Ruby on Rails](#) (framework Web)

[ASP.NET](#) (Microsoft, Visual Studio)

Zend Framework (framework PHP)

CakePHP (framework PHP)

Symfony (framework PHP)

Interfaces : innovations et évolution récente

La souris : espèce en voie de disparition?

- Appareils mobiles / Écrans tactiles / Assistants vocaux
 - Téléphones / Tablettes / Phablettes / Montres connectées...
 - Apple iOS (iPhone / iPad / Apple Watch) + Siri (Google Home)
 - Microsoft (Windows 10 / Windows Phone) + Cortana
 - Google Android (*smartphones* / Blackberry/ HTC Tablet)
- www.dontclick.it
- [Minority report](#) (Film 2002)
- [The sixth sense](#)
- [‘Imaginary’ interface could replace real thing](#)
- www.leapmotion.com

Interfaces : innovations et évolution récente

- Microsoft

- <http://www.microsoft.com/microsoft-surface-hub/>
- mashable.com/2015/04/29/microsoft-demos-how-windows-10-apps-work-with-hololens-and-its-stunningly-beautiful/

- Google

- <http://obsession.nouvelobs.com/high-tech/20150119.OBS0293/pourquoi-google-retire-ses-google-glass-de-la-vente.html> (Lunettes Google / Google Glass)
- <http://www.businessinsider.com/google-soli-finger-control-technology-2015-5>
- <http://www.wired.com/2016/05/device-turns-arm-touchpad-heres-works/>

A day made of glass

- <https://www.youtube.com/watch?v=jZkHpNnXLB0>

- *No interface* / interfaces sans contact

- Kinect : contrôle sans manette
- Reconnaissance des sentiments (vidéo Ubisoft séance 6)
- Apple HomePod (haut-parleur + Siri) / Amazon Echo (haut-parleur + Alexa)

- Objets connectés

- Vêtements connectés / données partagées (ex. avec notre médecin)

- Intelligence artificielle et *big data*

- Chatbots / assistants virtuels

Annexe

- Programmation selon MVC

Le MVC illustré en Java

Le modèle (ButtonModel)

- Représente l'état du dialogue
 - Bouton pressé, armé ou sélectionné
- Définit les méthodes (routines) pouvant agir sur cet état ou pouvant la manipuler (l'interroger par exemple)
 - isPressed, setPressed, SetArmed...
- Offre la possibilité d'enregistrer des contrôleurs d'action et de changement qui peuvent être actionnés en cas de nécessité

La Vue (ButtonUI)

- Sait rafraîchir la partie de l'interface touchée par un changement
 - Paint()
- Il communique avec le contrôleur des événements

Le contrôleur (ButtonUIListener)

- Définit des méthodes qui lui permettent
 - D'écouter, d'interpréter les événements qui surviennent dans la vue, et d'appeler les routines appropriées du modèle
 - D'écouter les changements survenus dans le modèle et demander un rafraîchissement à la vue.

Lien entre Vue et Contrôleur

- La vue doit connaître les contrôleurs d'événements qui lui sont connectés
- L'enregistrement des contrôleurs se fait via les méthodes `installUI` et `uninstallUI` définies dans la classe `ButtonUI`

Lien entre Vue et Contrôleur

- Suite à la notification d'un changement dans le modèle (message `stateChange` en provenance du `Model`), le contrôleur sait demander à la vue de se rafraîchir en lui envoyant le message `paint()`
- Voir la méthode `stateChange` définie dans la classe `ButtonUIListener`

Lien entre Contrôleur et le Modèle

- Le contrôleur doit interpréter et activer les routines appropriées du modèle
 - Voir les méthodes: *mouseDragged*, *MousePressed*, *MouseReleased*...
- Le contrôleur doit traiter la notification de changement envoyée par le modèle. Ainsi, il implémente la méthode
 - *stateChanged* qui lui permet de demander un rafraîchissement à la vue

Lien entre Contrôleur et le Modèle

- Le modèle doit connaître les contrôleurs d'actions et de changement. Il offre la possibilité de les connecter ou de les déconnecter
 - Voir les méthodes *removeChangeListener*, *addChangeListener*, *addActionListener* et *removeActionListener*
- Il offre aussi des méthodes qui permettent de notifier les contrôleurs appropriés suite aux changements
 - Voir la méthode *FireChangeEvent*

```

import java.awt.*;
import java.awt.event.*;
import java.text.*;

class CalcTest {
    public static void main(String args[]) {
        CalcModel calc = new CalcModel();
        /* Le Modèle crée sa Vue et son Contrôleur */
        calc.myView.pack();
        /*ajuste la fenêtre de la calculatrice à son contenu*/
        calc.myView.setVisible(true);
        /* la rend visible à l'écran */
    }
}

class CalcModel {
    static final int MAXLENGTH = 10;
    static final char NOP = '\0';
    CalcView myView;
    double x=0,y=0;
    char op=NOP;
    boolean insert = false;
    StringBuffer bufX;

    CalcModel() {
        bufX = new StringBuffer(MAXLENGTH);
        CalcController cc = new CalcController(this);
        myView = new CalcView(this,cc);
        changed();
        /* Met à jour la Vue en fonction de l'état du Modèle */
    }
}

```

```

/*Manipulation des registres x et y selon une logique de pile*/
double pop ()
{
    double r = x;
    x=y;
    y=0;
    return r;
}
void push (double v)
{
    y=x;
    x=v;
}
/* Idem pour le registre "opérateur" */
char popOp ()
{
    char r = op;
    op = NOP;
    return r;
}
void pushOp (char op)
{
    this.op = op; }
/* Effectue l'opération désirée sur les registres */
double execOp (char op, double x, double y)
{
    switch (op){
        case '+': return y+x;
        case '-': return y-x;
        case '*': return y*x;
        default : return y/x;
    }}

```

```

/* Traitement de la touche = */
void processEq(){
/* Si le buffer vient d'être édité, alors on transforme son contenu en
un flottant double précision (double) pour pouvoir faire des calculs */
if (insert) push(Double.valueOf(bufX.toString()).doubleValue());
insert = false;
/* S'il y a un opérateur en attente, alors
1- dépiler l'opérateur
2- dépiler les opérandes
3- effectuer l'opération
4- empiler le résultat
*/
if (op != NOP){
push(execOp(popOp(),pop(),pop()));
changed();
}}
/* Traitement des touches +,-,/,* */
void processBinOp(char op){
/* Effectue la même action qu'un = puis empile l'opérateur */
processEq();
pushOp(op);
}
/* Traitement des touches 0 à 9 et . */
void processDigit(char digit){
/* Si l'on n'est pas déjà en mode insertion, alors s'y mettre et effacer le buffer
*/
if (! insert) {
clearBuf();
insert=true;
}
/* ajoute au buffer le caractère qui vient d'être entré */
addBuf(digit);
changed();
}

```

```

/* Traitement de la touche C */
void clear (){
clearBuf();
insert=false;
x=y=0;
changed();
}
/* Effacement du buffer */
void clearBuf(){
/* le plus simple est d'en créer un nouveau */
bufX = new StringBuffer(MAXLENGTH);
}
/* Ajout d'un chiffre ou d'un point au buffer */
void addBuf(char d)
{
/* Si le buffer n'est pas plein */
if (bufX.length() < MAXLENGTH)
/* et si d n'est pas un point ou sinon n'est pas déjà présent */
if (d != '.' || bufX.toString().indexOf(d) == -1){
/* alors ajouter le caractère au buffer */
bufX.append(d);
}
}
/* Noter l'emploi de cette méthode "changed" dans les méthodes ci-dessus */
void changed(){
/* Le Modèle a changé : il rafraichit sa Vue */
myView.refresh();
}
}

```

```

class CalcDisplay extends Canvas {
    CalcModel myModel;
    int leftMargin = 5;
    DecimalFormat nf = new DecimalFormat();

    CalcDisplay(CalcModel model){
        myModel = model;
    }
    public void paint(Graphics g){
        int fh = g.getFontMetrics().getAscent();
        int dh = getSize().height;
        int y = (dh+fh)/2;

        if (myModel.insert)
            /* En mode insertion, on affiche le contenu du buffer */
            showBufX(g,myModel.bufX,y);
        else
            /* Autrement, on affiche le registre x */
            showX(g,myModel.x,y);
    }
    private void showBufX (Graphics g, StringBuffer bx, int y)
    {
        g.drawString(bx.toString(),leftMargin,y);
    }
    private void showX (Graphics g, double x, int y)
    {
        String pattern = "0.00";
        /* On utilise un format limité à 2 chiffres après la virgule */
        nf.applyPattern(pattern);
        g.drawString(nf.format(x),leftMargin,y);
    }
}

```

```

class CalcView extends Frame {
    CalcModel myModel;
    CalcController myController;
    CalcDisplay display;

    CalcView (CalcModel model, CalcController c) {
        myModel = model;
        myController = c;

        Panel keyPanel = new Panel();
        keyPanel.setLayout(new GridLayout(4,4));

        display = new CalcDisplay(myModel);
        display.setSize(132,28);

        Button key_0 = new Button("0");
        ...
        Button key_9 = new Button("9");
        Button key_point = new Button(".");
        Button key_eq = new Button("=");
        ...
        Button key_clear = new Button("C");

        key_0.addActionListener(myController);
        ...
        key_9.addActionListener(myController);
        key_point.addActionListener(myController);
        key_eq.addActionListener(myController);
        key_add.addActionListener(myController);
        ...
        key_clear.addActionListener(myController);
    }
}

```

```

keyPanel.add(key_7);
keyPanel.add(key_8);
keyPanel.add(key_9);
keyPanel.add(key_sub);
keyPanel.add(key_4);
keyPanel.add(key_5);
keyPanel.add(key_6);
keyPanel.add(key_add);
keyPanel.add(key_1);
keyPanel.add(key_2);
keyPanel.add(key_3);
keyPanel.add(key_mul);
keyPanel.add(key_0);
keyPanel.add(key_point);
keyPanel.add(key_eq);
keyPanel.add(key_div);

setLayout(new BorderLayout ());
add(display,BorderLayout.NORTH);
add(keyPanel,BorderLayout.CENTER);
add(key_clear,BorderLayout.SOUTH);
}

void refresh(){
    /* Le seul composant qui doit être re-dessiné quand le modèle change
    est la partie affichage de la calculatrice */
    display.repaint();
}
}

```

```

class CalcController implements ActionListener {
    CalcModel myModel;

    CalcController(CalcModel model){
        myModel = model;
    }

    public void actionPerformed(ActionEvent e){
        String name = ((Button)e.getSource()).getLabel();
        char sym = name.charAt(0);
        switch (sym){
            case '0' :
            case '1' :
            case '2' :
            case '3' :
            case '4' :
            case '5' :
            case '6' :
            case '7' :
            case '8' :
            case '9' :
            case '.' : myModel.processDigit(sym); break;
            case '+' :
            case '-' :
            case '*' :
            case '/' : myModel.processBinOp(sym); break;
            case '=' : myModel.processEq(); break;
            case 'C' : myModel.clear(); break;
        }
    }
}

```