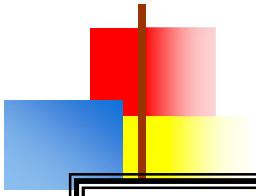
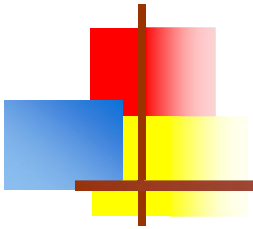


MODELES D'ARCHITECTURES LOGICIELLES

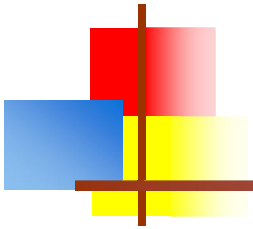
- ***Introduction*** *(objectif, indépendance du dialogue)*
- ***Modèle Langage (Seeheim, Arch)***
- ***Modèle Multi-Agents (MVC, PAC)***



Architecture Logicielle

- **Objectif: faire un système logiciel qui répond à la fois:**
  - aux exigences des **parties prenantes**
  - aux contraintes qui existent durant sa production

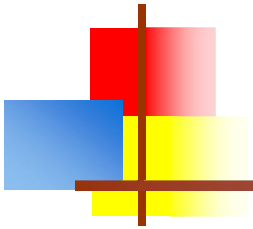
→ **Qui est viable industriellement**



Architecture Logicielle

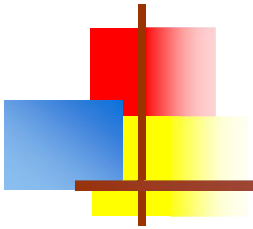
- **Partie prenante (stakeholder):**
  - une personne (ou un groupe) qui a un intérêt dans le système à créer

- ***Dans la conception*:**
  - la plupart du temps décrit avec des cas d'utilisation (**use cases**) avec description textuelle de chacun des cas
  - en nombre plutôt limité, faciles à appréhender.



Architecture Logicielle

- **Exigences système :**
  - décrit ce que le **système à développer** (la solution) doit faire et comment. Émanent des parties prenantes
  - **exemple**: utilisateur, développeur, architecte, administrateur, client, actionnaires …
- ***Dans la conception*:**
  - L'architecte logiciel interagit avec tous ces systèmes
  - Il s'assure que ce qui est construit et les choix qu'il fait sont en **adéquation** avec leurs besoins

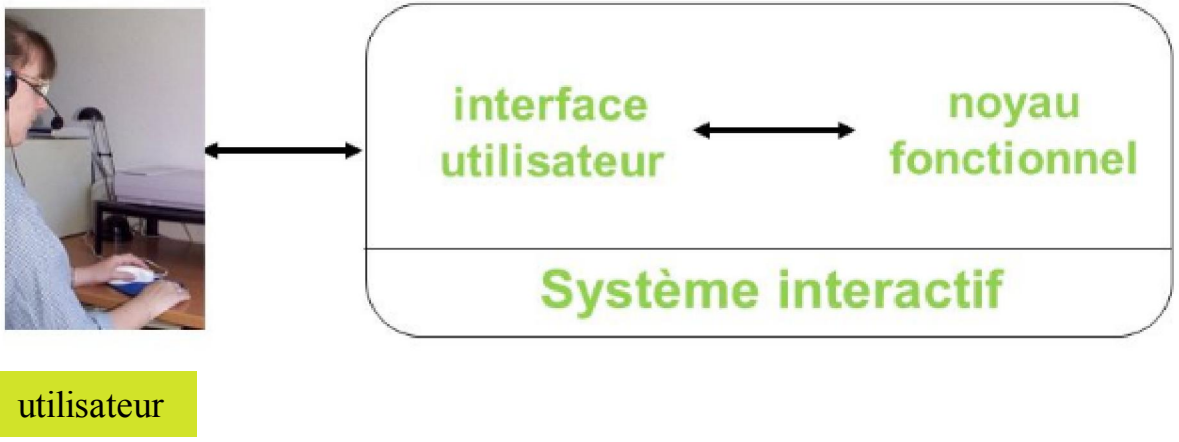


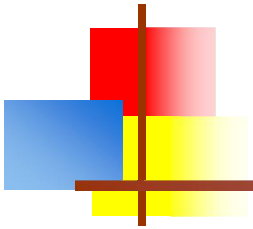
Architecture Logicielle

- **Architecture** : ensemble organisé de composants + des relations + des principes directeurs
- **Environnement** : participants (culture en qualité logicielle, outils, requis commercial…)
- **Finalité** d'une architecture
  - Communication (précision et non ambiguïté de la description)
  - rétro-conception d'un système existant
  - évaluation (selon des critères de qualité)

Modèles d'Architecture

- Tous les **modèles d'architecture** ont pour **principe:**
- Un **système interactif** comporte une partie **interface** et une partie **application pure** appelée ***noyau fonctionnel*** (considéré préexistant)





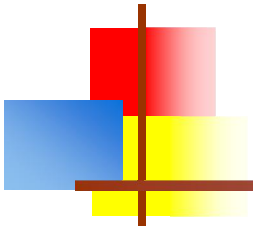
Indépendance (interface/ application)

- **Objectif:** réaliser **l'indépendance** entre :
  - le Code IHM (présentation)
  - le Code application (métier)

  Contrôleur de dialogue(3è composante)→Communication entre Application(métier) et Présentation (IHM)

  Exemple: **réutiliser l'Interface** relative à une Gestion de stock (produits alimentaires, cosmétiques, de pharmacie..)

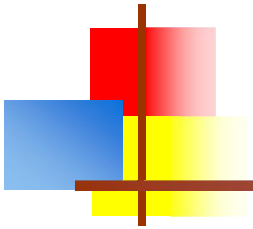
- L'indépendance augmente la **rapidité** pour modifier le **code** pendant le développement, puis au cours de la vie de **l'application**



Indépendance du Dialogue

- **La séparation entre l'interface et le noyau offre:**
  - Portabilité (device independant)
  - Réutilisabilité
  - Plusieurs Interfaces (flexibilité, plateformes, etc)
  - Personnalisation (designer, utilisateur)

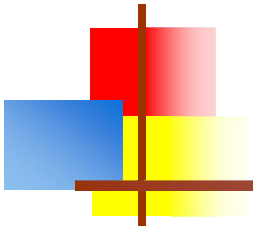
- Sans **l'indépendance**,**l'interface** (style d'interaction) et le **noyau** (composants fonctionnels) sont **intimement mêlés**
- Toute modification d'un **paramètre** de l'interface exige de parcourir la totalité du **code de l'application** pour faire des corrections



Indépendance du Dialogue

- **Exemple**: système dont l'affichage des messages d'erreur est dispersé partout dans le code

- Si on veut modifier le style des messages (ex: ***créer une boîte de dialogue à chaque fois qu'une erreur survient)***, au lieu d'afficher tous les messages à la suite d'une fenêtre d'état, on doit **revoir tout le code**



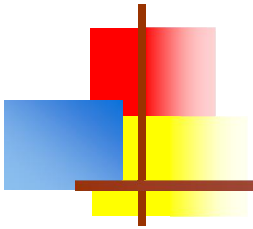
Indépendance du Dialogue

Exemple: affichage de message d'erreur en Java:

```
public void ShowError (string err_msg)
{  system.out.println ( "présence erreur n°" +
   err_msg); }
```

- Changement de l'implémentation de la fonction :

```
public void ShowError (string err_msg)
{  g.setColor (new Color (255,0,0));
   g.drawString ( x, y, "présence erreur n°" +
   err_msg);  }
```



Indépendance du Dialogue

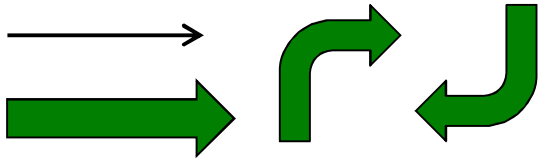
Exemple 2: paramétrage de fonctions graphiques:

public void **Dessin_triangle** (int x1, int y1, int x2, int y2, int x3, int y3)

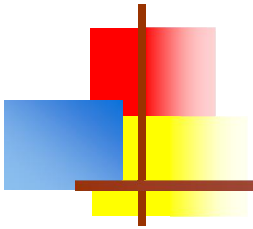
public void **Dessin_hexagone** (int x1, int y1, …., int x6, int y6) // polygone à 6 cotés

public void **Dessin_octagone** (int x1, int y1, …., int x8, int y8) // polygone à 8 cotés

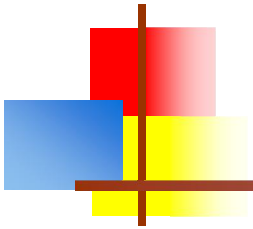
Public void **Dessin_Fleche** (…)



Exemple 3: réutilisation des templates (XML, php, …), des feuilles de style css pour faciliter le développement web



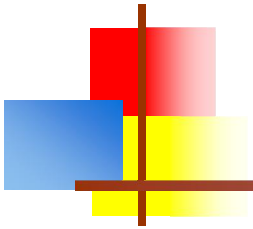
Modèle Langage



Modèle Langage

- S'inspire de l'analogie entre l'interaction homme-machine et le dialogue entre individus.
- L'utilisateur et le système communiquent avec un langage commun défini selon 3 niveaux (couches ou composantes): ***Sémantique, Syntaxe et Lexique***

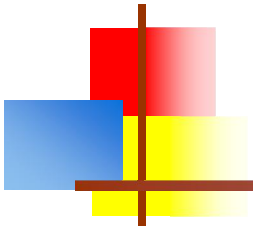
  - **Modèle Seeheim**
  - **Modèle Arch**



Niveau Sémantique

- Description précise des classes d'objets auxquelles l'utilisateur et le système font référence dans leur conversation
- A ce niveau, on s'intéresse à la signification des objets et les opérations sur ces objets et non la façon comment on manipule les objets

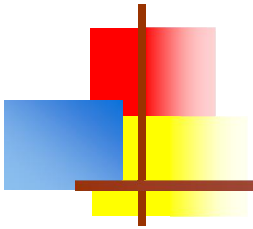
- Exemple: la température augmente la puissance du réchaud



Niveau Syntaxique

- Définit la construction des phrases du langage à partir des **éléments syntaxiques**.
- Choix du **style de dialogue** (menus, formulaires, langage de commande, langage naturel, manipulation directe, action/objet...), de la langue

- Ex: image d'un thermomètre, **agir sur un bouton pour augmenter la puissance d'un réchaud**.



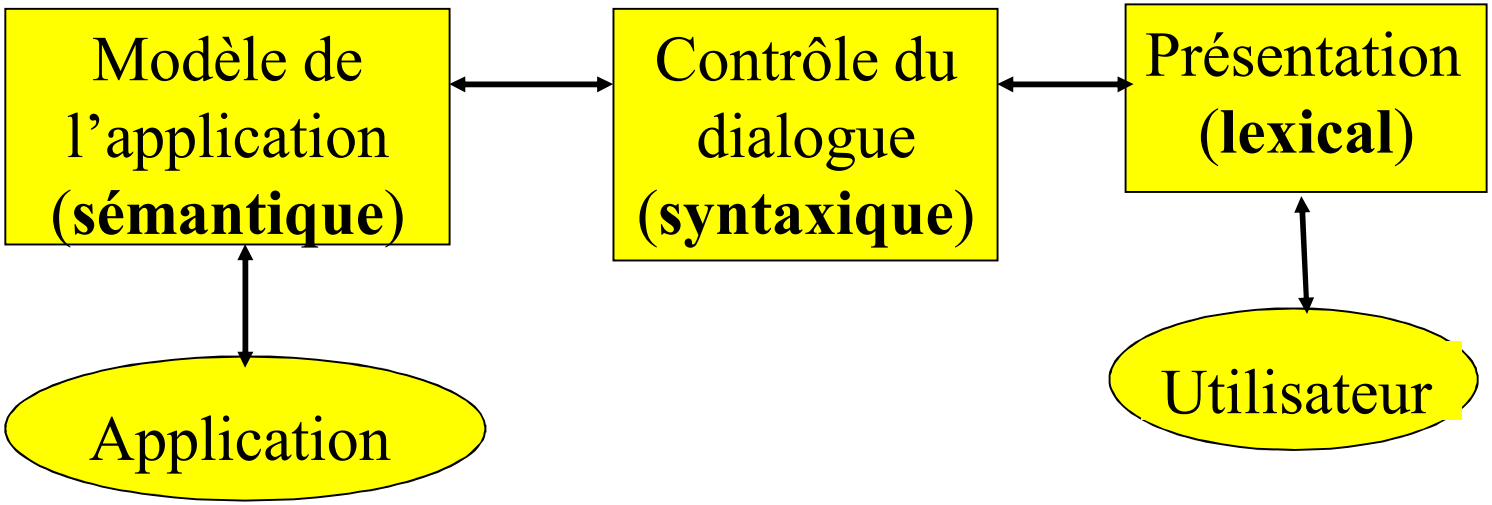
Niveau Lexical

- Définit le **vocabulaire** avec lequel sont formées les commandes, textes, et attributs d'affichage des objets: **couleur**, **fonte**, taille, touches fonctions, accélérateurs
- Domaine de la **représentation graphique** et la présentation **externe**

- **Exemple**: couleur et police de caractères du texte de sortie, taille,…

Modèle Seeheim

- Modèle en **Couches** (SIGGRAPH 1985)
- **Objectif**: faire la distinction entre la **sémantique** (traitements) et la **présentation** dans une application interactive



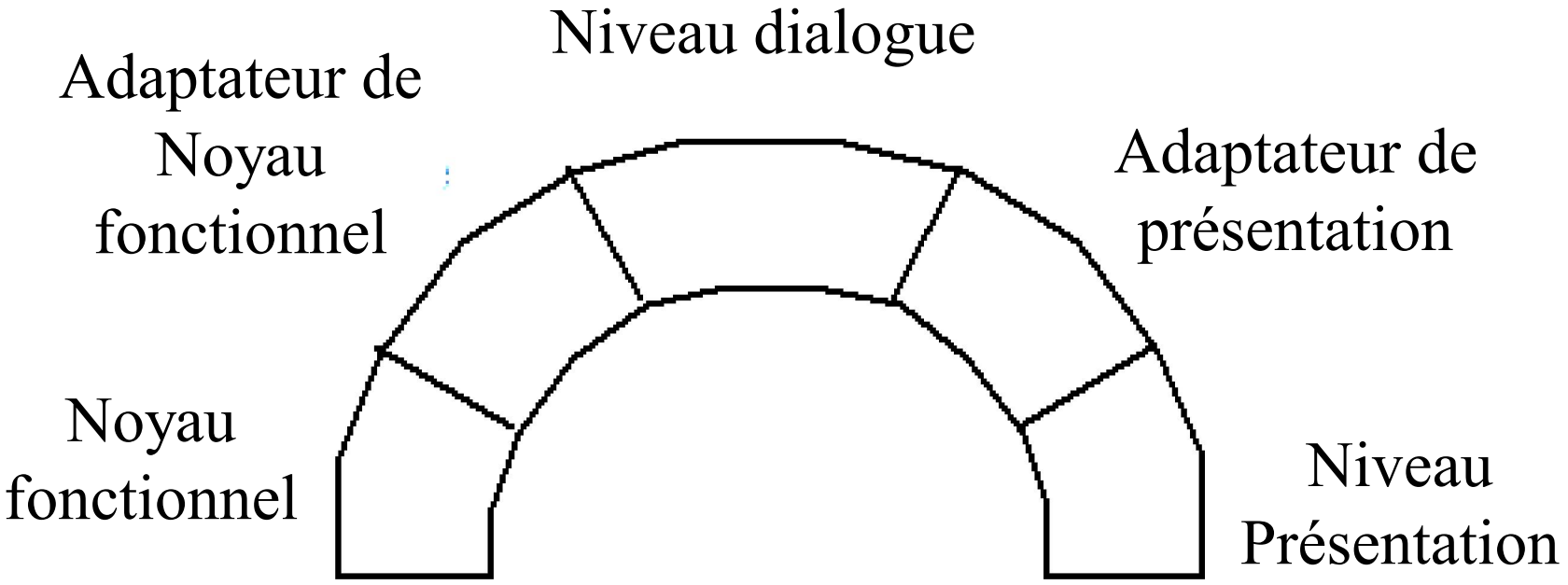
Modèle Seeheim

- **Présentation**
  - Définit l'image du système pour l'utilisateur (niv. lexical)
  - Gère la présentation et l'affichage des objets, ainsi que les événements physiques d'entrée/sortie

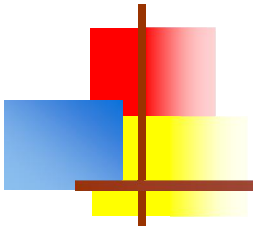
- **Contrôle du Dialogue** → Médiateur entre l'utilisateur et l'application (niveau syntaxique)
- **Modèle de l'application** → Noyau Fonctionnel (sémantique) de l'application

Modèle Arch/ Seeheim modifié

Plus de couches: appliquer l'abstraction à la présentation et l'application



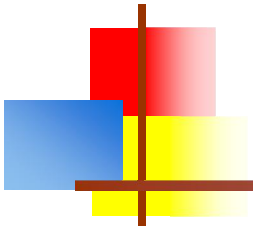
<u>Objectif</u>: accroître la portabilité de l'application et la présentation



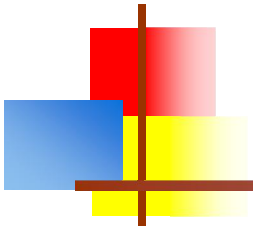
Modèles Langages

- **Avantages :**
- permettent une définition d'un cadre de pensée, une conception itérative des interfaces, à leur généricité et généralité d'utilisation

- **Inconvénients - limitations :**
- Traitement des formes centralisé (même pour les détails lexicaux …)
- Niveau d'abstraction des protocoles de communication entre les composants trop imprécis



MODELE MULTI-AGENT

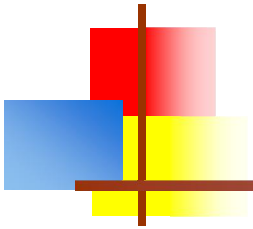


Modèle MULTI-AGENT

- Structure un système interactif en un ensemble **d'agents (objets)** spécialisés qui réagissent à des événements et produisent des événements

- Organisation **modulaire**, traitements exécutés en **parallèle** et **communication par événements**

- Remplace le modèle langage (séquentiel et dichotomique)

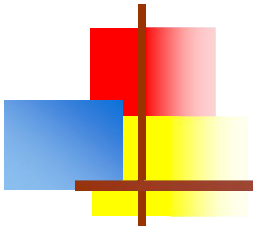


MODELE MVC

- **MVC** (Modèle, Vue, Contrôleur)
- **Smalltalk [Goldberg et Robson, 1981].**

- Un agent **MVC** = un ***modèle***, une ou plusieurs ***vues***, et un ou plusieurs ***contrôleurs***

- Permet de concevoir des interfaces graphiques modulaires en séparant clairement les trois composants d'éléments d'interfaces

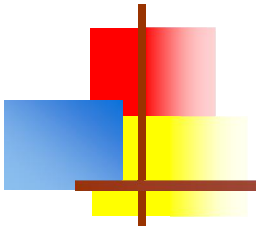


MODELE MVC

• **Modèle:** modélisation d'un **objet: données et comportement.** (exemple: variable booléenne pour l'état d'une case à cocher)

• **Vue:** représentation de l'état de l'objet à l'écran (interface avec l'utilisateur). Ex: une case à cocher est représentée par )

• **Contrôleur:** interprétation des entrées, gestion des événements (clavier souris), synchronisation (ex: que se passe-t-il si l'utilisateur clique sur une case à cocher)



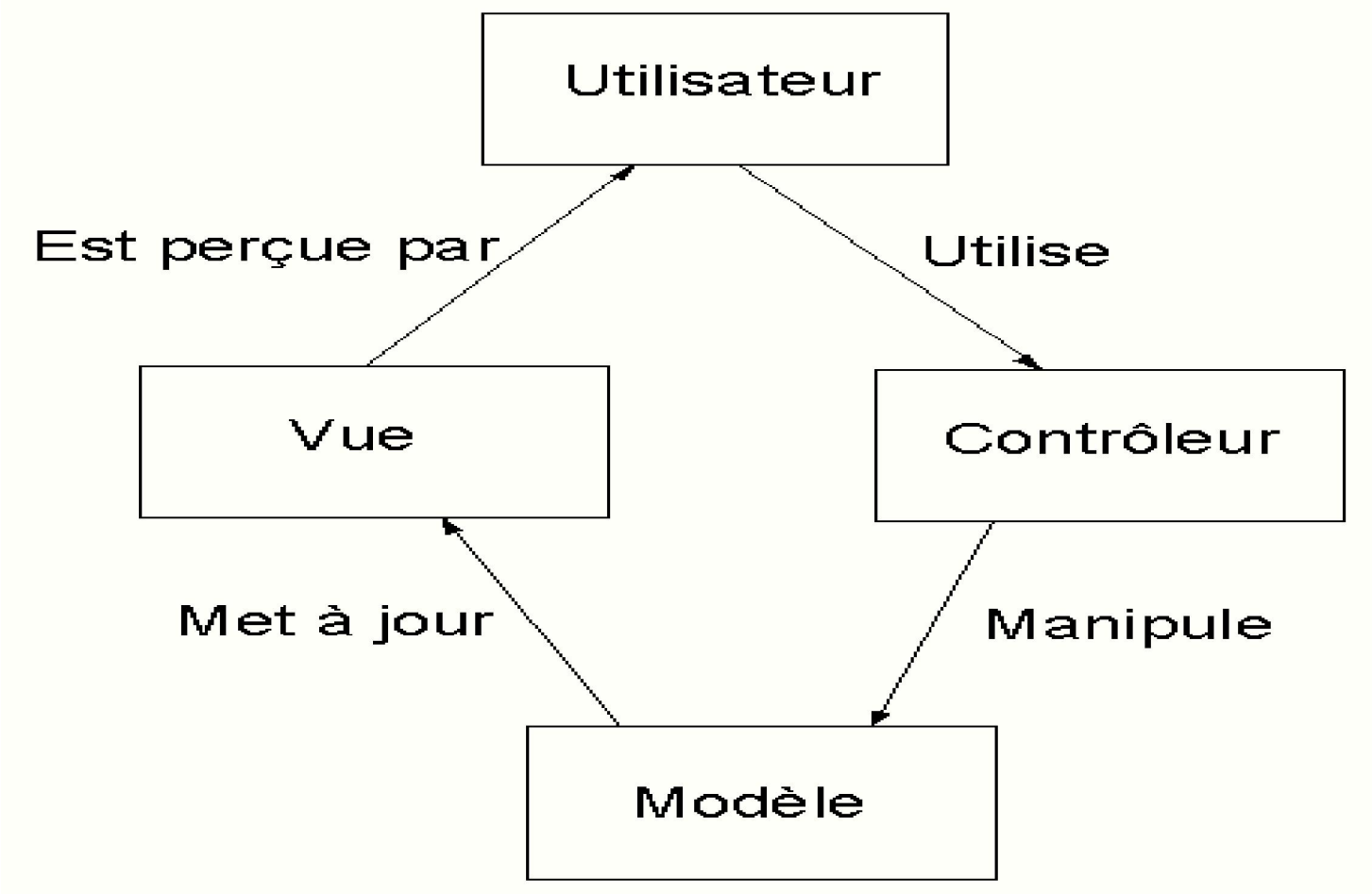
MODELE MVC

- MVC est utilisée par **Swing** pour rendre l'interface ouverte et configurable. Chaque objet Swing est associé à deux objets :

  • un **objet modèle**, qui traite de l'état du composant

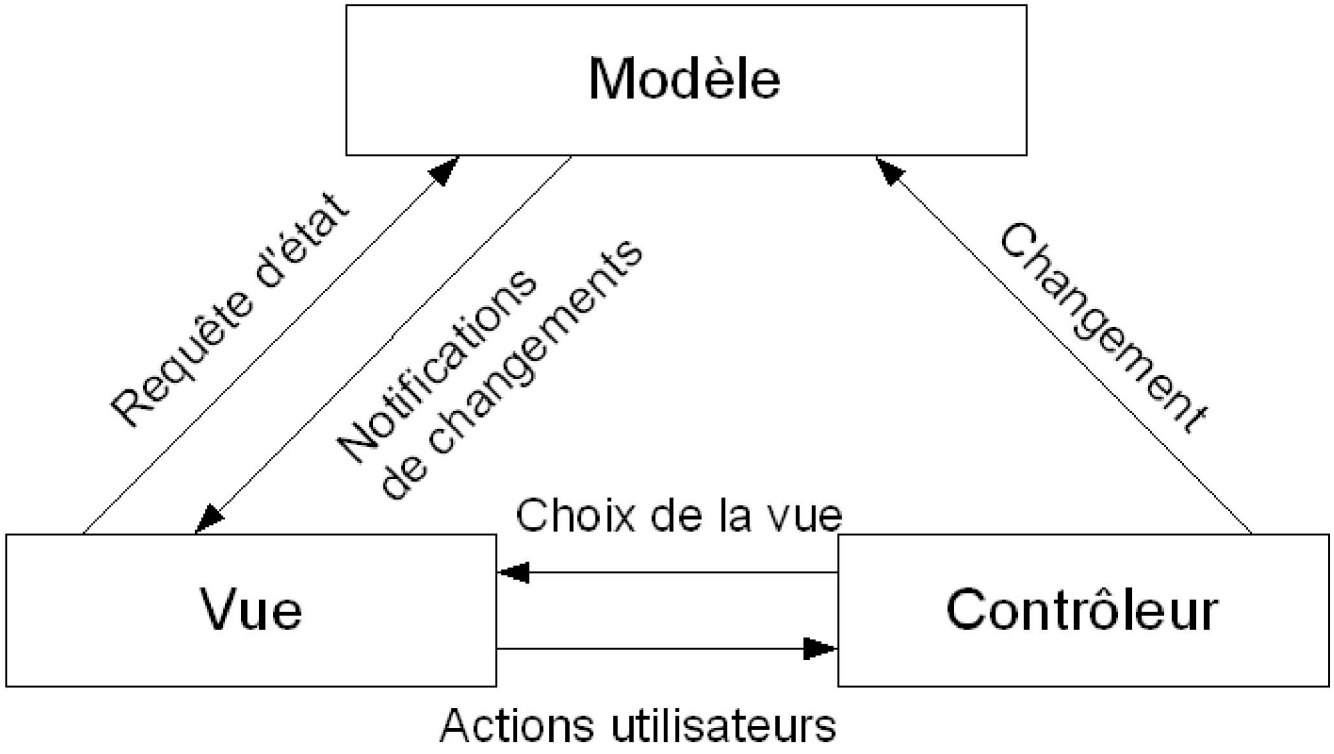
  • un **objet d'interface** qui gère la vue (le look) et le contrôle (le feel)

Architecture MVC

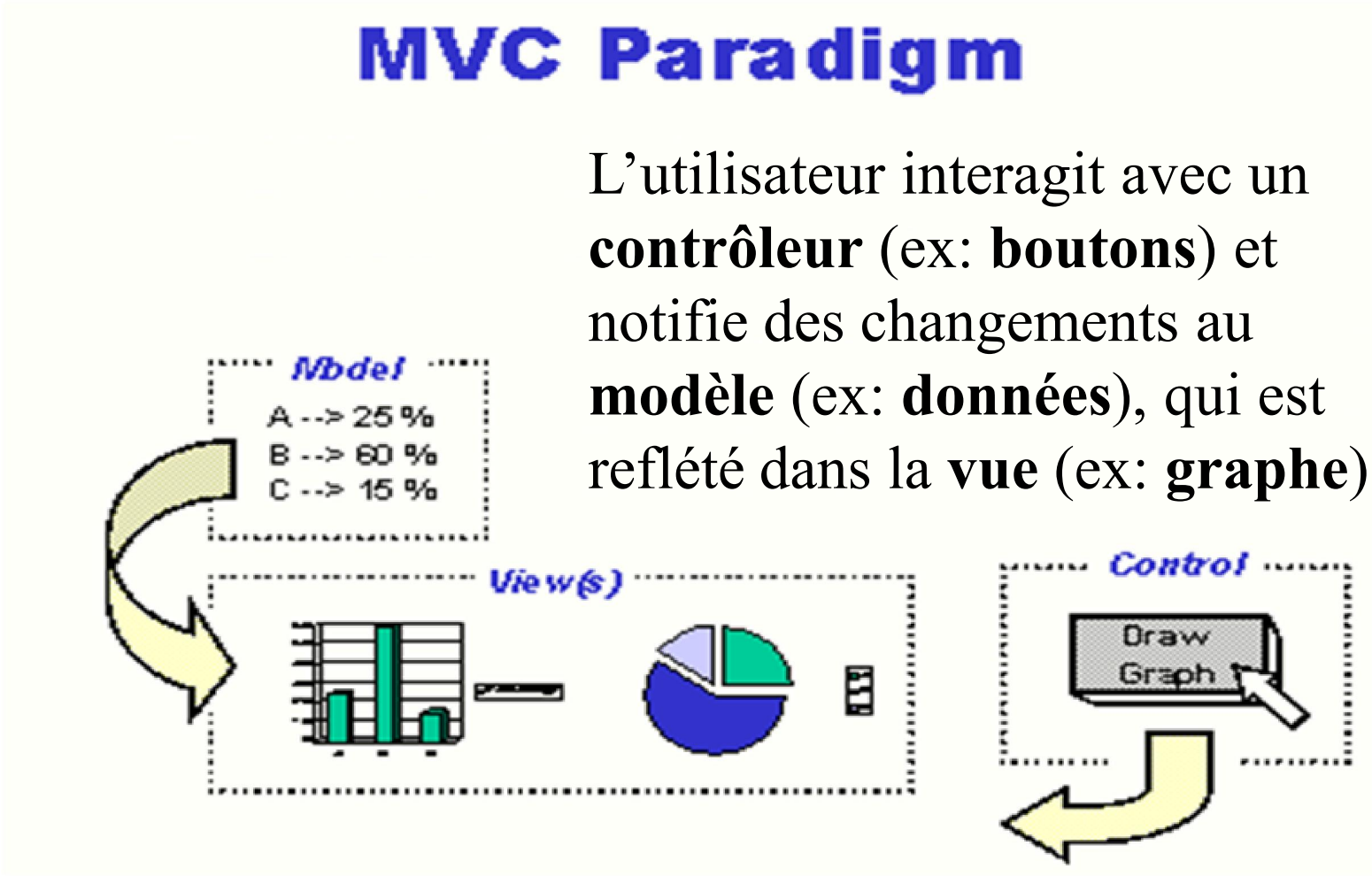


Architecture MVC

Le Contrôleur au centre de la communication

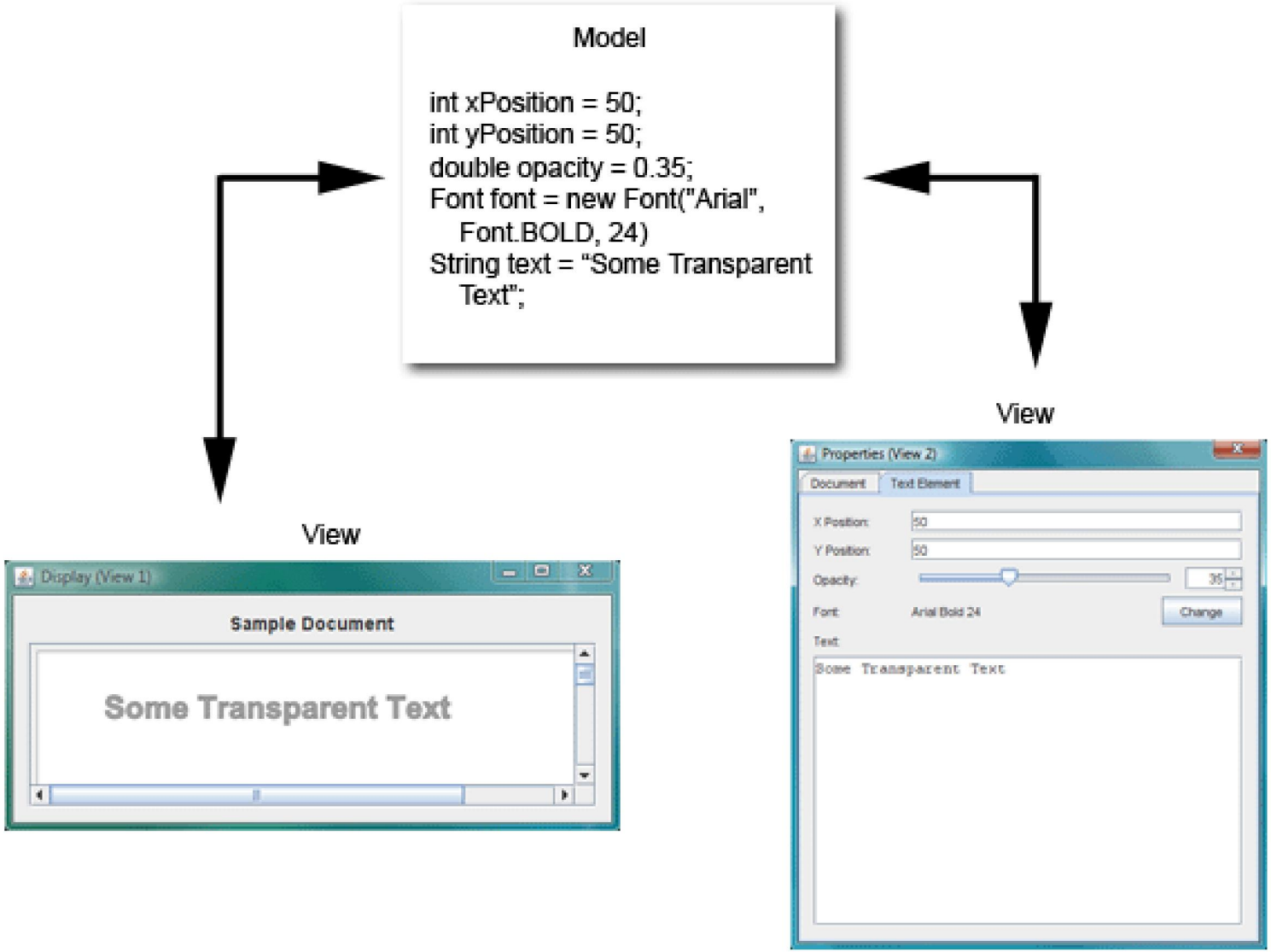


Exemple MVC

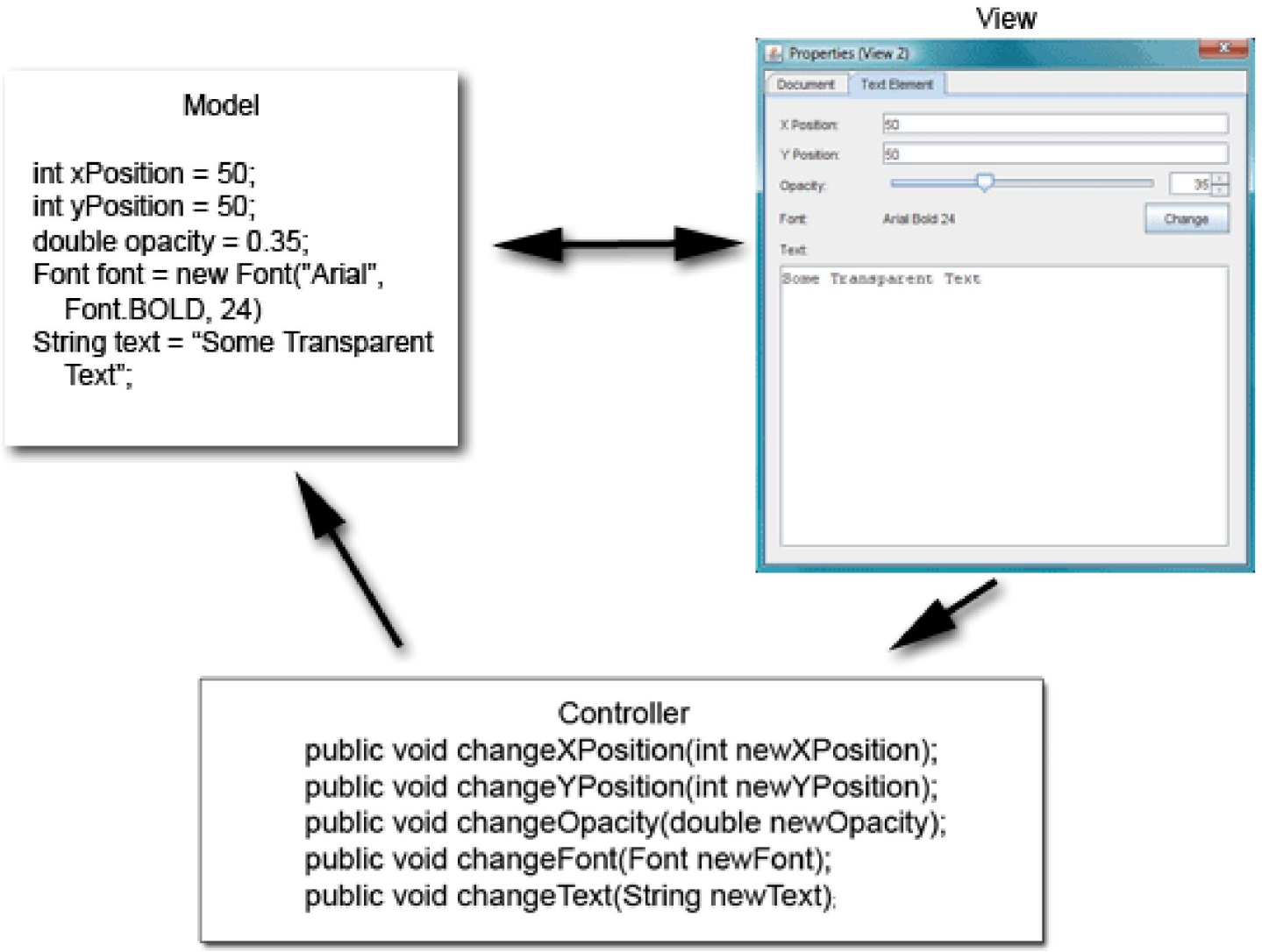


L'utilisateur interagit avec un **contrôleur** (ex: **boutons**) et notifie des changements au **modèle** (ex: **données**), qui est reflété dans la **vue** (ex: **graphe**)

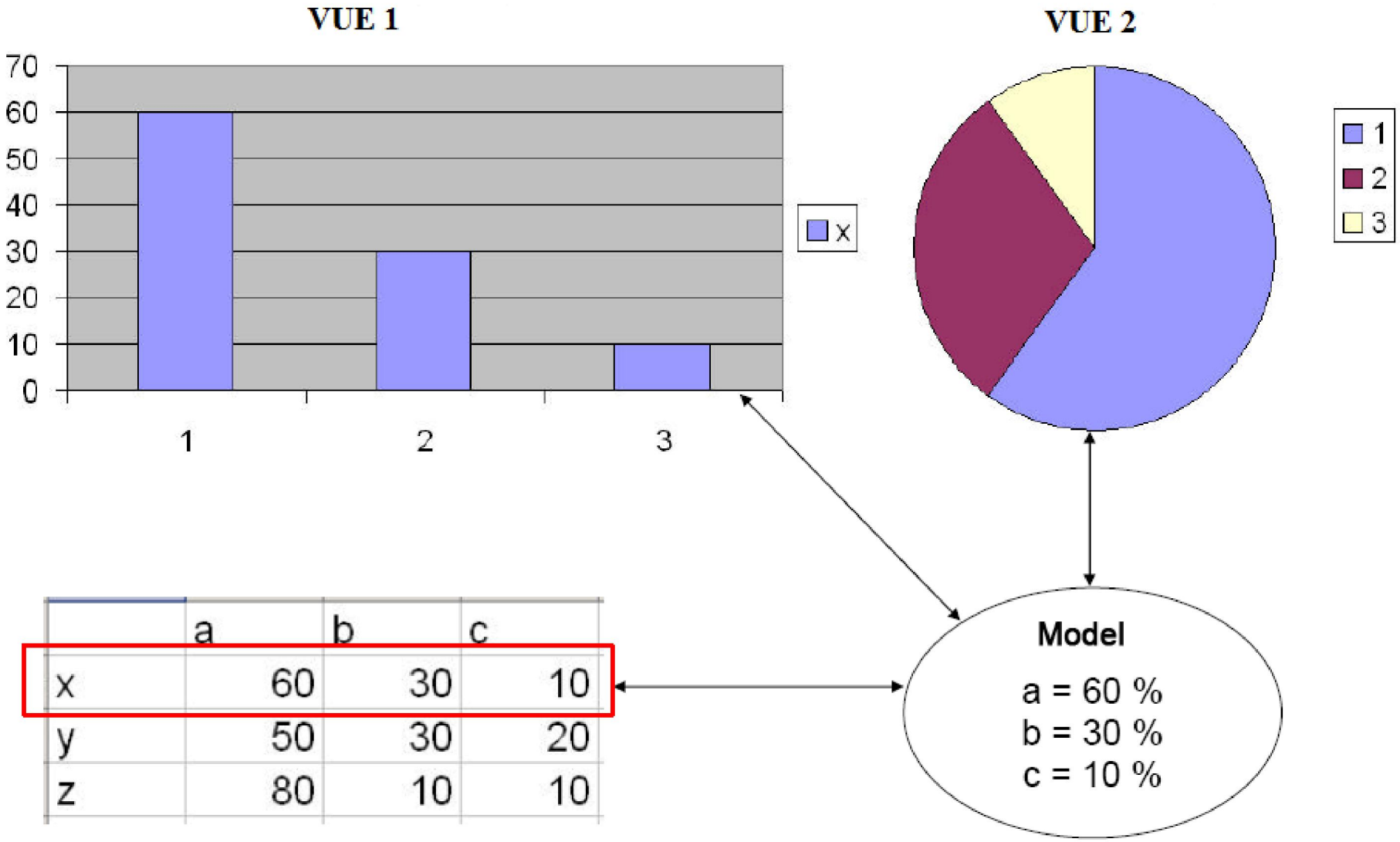
Plusieurs vues – 1 modèle



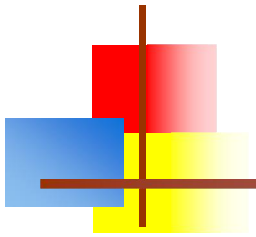
Exemple


View
Model
int xPosition = 50;
int yPosition = 50;
double opacity = 0.35;
Font font = new Font("Arial",
Font.BOLD, 24)
String text = "Some Transparent
Text";
Properties (View 2)
Document
Text Element
X Position:
50
Y Position:
50
Opacity:
35
Font:
Arial Bold 24
Change
Text:
Some Transparent Text
Controller
public void changeXPosition(int newXPosition);
public void changeYPosition(int newYPosition);
public void changeOpacity(double newOpacity);
public void changeFont(Font newFont);
public void changeText(String newText);

Exemple 2



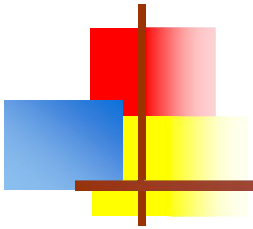
	a	b	c
x	60	30	10
y	50	30	20
z	80	10	10



MVC: Exemple Java …

Le programme “Inverseur”

- Dans ce programe le Contrôleur et la Vue sont combinés (il est difficile de les séparer)
- Le Modèle qui doit faire le calcul (inverser une chaîne de caractères) est mis dans une classe séparée

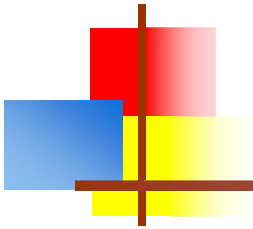


ReverserGUI.java

- Commencer par les instructions import, ensuite...

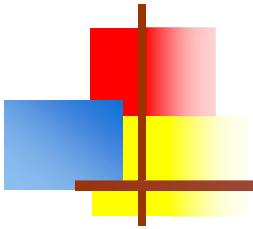
- public class **ReverserGUI** extends **Jframe** implements **ActionListener** {

```
    ReverserModel model = new ReverserModel();
    JTextField texte = new JTextField(40);
    JButton bouton = new JButton("Inverser");

    public static void main(String[] args) {
        ReverserGUI inverse = new ReverserGUI();
        inverse.create();

inverse.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE
);    }
```



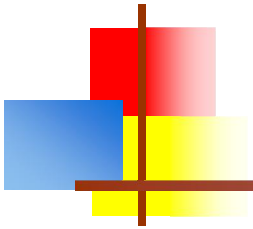
La méthode Create

```
    private void create() {
        setLayout(new GridLayout(2, 1));
        add(texte);
        add(bouton);

        button.addActionListener(new ActionListener()
{
 public void actionPerformed (ActionEvent arg0)

 {              String s = text.getText();
                s = model.reverse(s);
                text.setText(s);
            }
        } );
        pack();
        setVisible(true);      }
```



Le Modèle

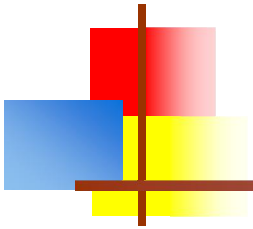
```
public class ReverserModel {

    public String reverse(String s) {

        StringBuilder builder = new StringBuilder(s);
        builder.reverse();
        return builder.toString();
    }
}
```



Le Modèle

- Le Modèle est la partie qui fait le travail --il *modélise* le problème actuel à résoudre

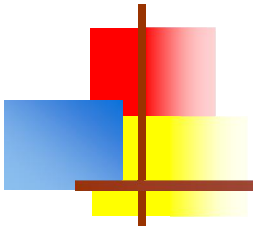
- **Le Modèle doit être indépendant du Contrôleur et la Vue**
  - **Mais il leur fournit des services (méthodes) à utiliser**



Le Contrôleur

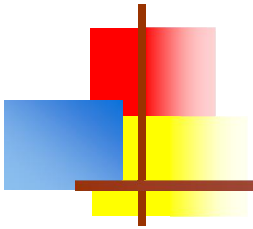
- Souvent, l'utilisateur est mis en contrôle à l'aide de composants de l'interface
  - dans ce cas, **l'Interface et le Contrôleur** sont les mêmes
- Le Contrôleur et le Modèle doivent être quasiment toujours séparés **(quoi faire versus comment le faire)**

- La conception du **Contrôleur dépend du Modèle**
- Le **Modèle** ne doit **pas dépendre** du **Contrôleur**



Combiner Contrôleur et Vue

- Parfois le Contrôleur et la Vue sont combinés, surtout dans des petits programmes
- Combiner le Contrôleur et la Vue est approprié s'ils sont trés dépendants entre eux.
- Le Modèle par contre doit être indépendant
- *Ne jamais* mélanger le code du **Modèle** avec le code du **GUI** (Graphics User Interface).



Séparation des concepts

- **L'indépendance** du code est l'idéal à chercher
- Le **Modèle** ne doit **pas être mélangé** avec le code du contrôle ou le code d'affichage
- La **Vue** doit représenter le Modèle comme il est réellement
- Le **Contrôleur** doit ***communiquer*** *avec* le Modèle et la Vue, et ne pas les *manipuler*
  - Le Contrôleur peut **modifier** des variables que le Modèle et la Vue peuvent lire