

**Cours**  
**Programmation Orientée Objet 2**  
**Pour**  
**ING 2**

**Chap 04:**  
**Interfaces Graphiques**

MEKAHLIA Fatma Zohra LAKRID  
Maître de Conférences Classe B

Laboratoire de Modélisation, Vérification et Evaluation des Performances des systèmes complexes (MOVEP)  
Bureau 123

# PLAN

- Généralités sur les interfaces graphiques.
- Composants des interfaces graphiques.
- Les packages AWT et Swing.
- Classes de base.
- Création et affichage d'une fenêtre.
- Placer des composants dans une fenêtre .
- Création de Jar exécutable.
- Gestion des événements.
- Le modèle MVC.

# MVC ... Le design pattern : l'architecture Modèle- Vue-Contrôleur

# DESIGN PATTERN

- les design patterns (patrons de conception en français) ?
- Ce sont des solutions générales et réutilisables d'un problème récurrent et considérées comme des “bonnes pratiques”.
- Les patrons de conception sont une boîte à outils permettant de résoudre des problèmes classiques de la conception de logiciels. Ils définissent un langage commun pour aider votre équipe à communiquer plus efficacement.

# DESIGN PATTERN

- Les patrons de conception diffèrent par leur complexité, leur niveau de détails et l'échelle à laquelle ils peuvent être mis en œuvre.
- Permettent de résoudre des problèmes courants (par exemple : la conception d'une interface graphique).

# MVC: MODÈLE-VUE-CONTRÔLEUR

- Pour Modèle-Vue-Contrôleur ou en anglais Model-View-Controller.
- Le problème : la conception d'interface graphique et la programmation client/serveur.
- Solution datant de la fin des années 70 avec le développement des premières interfaces graphiques et indépendante des langages de programmation.
- C'est une manière de structurer une application graphique.

# MVC: MODÈLE-VUE-CONTRÔLEUR

En effet, MVC est un modèle de conception pour le développement d'applications logicielles qui sépare le modèle de données, l'interface utilisateur et la logique de contrôle.

**Son principe:** Organiser son architecture et son code en séparant trois rôles :

# MVC: MODÈLE-VUE-CONTRÔLEUR

- le modèle : la logique interne du programme, la gestion des données, les calculs, etc.
- la vue : l'affichage pour l'utilisateur final, tous les aspects graphiques
- le contrôleur : la gestion des événements graphiques lancés par l'utilisateur et le lien entre la vue et le modèle.

**Pourquoi ?** Organisation, modularité, maintenabilité, séparation des compétences et expertises entre les membres de l'équipe.

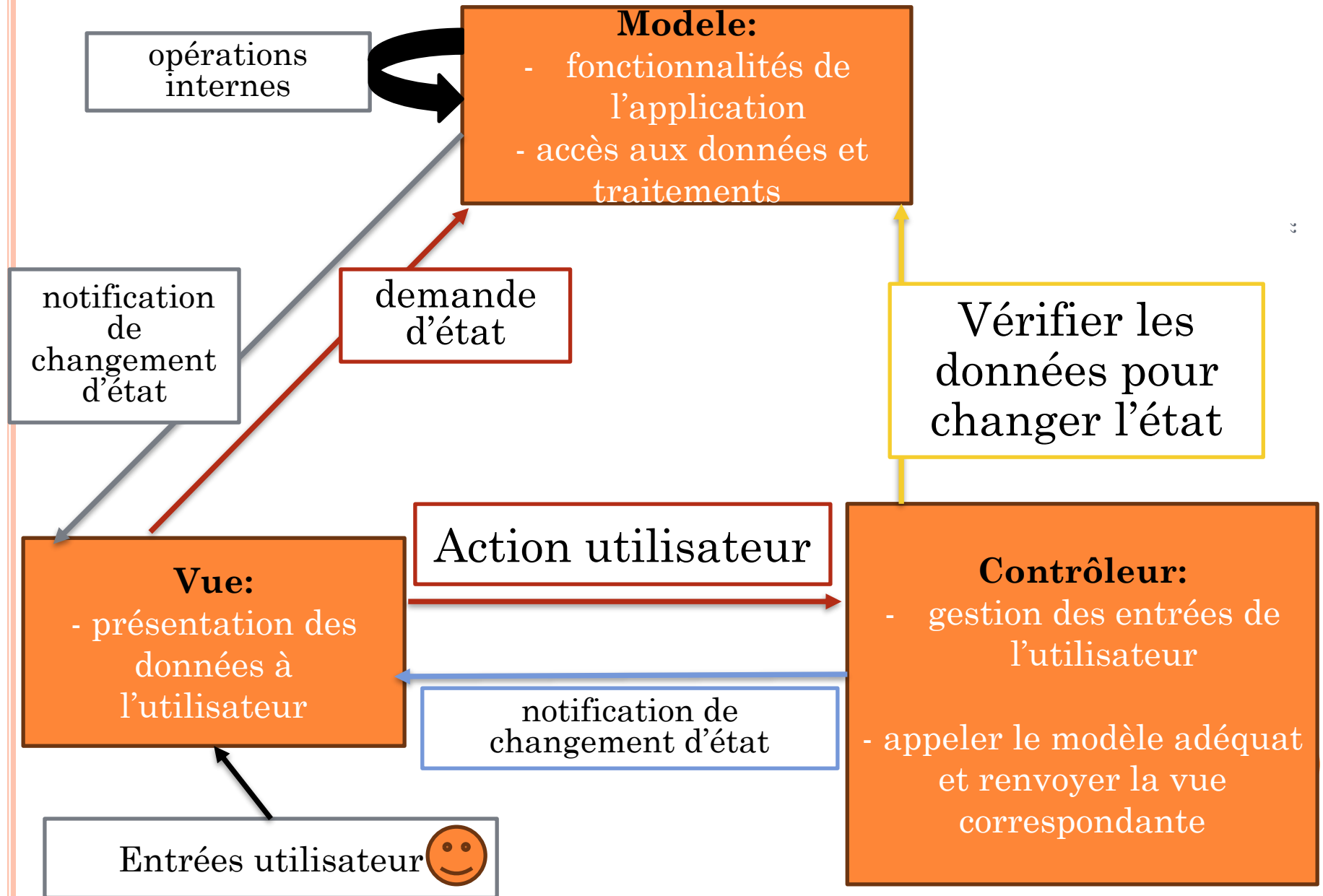


# MVC: MODÈLE-VUE-CONTRÔLEUR

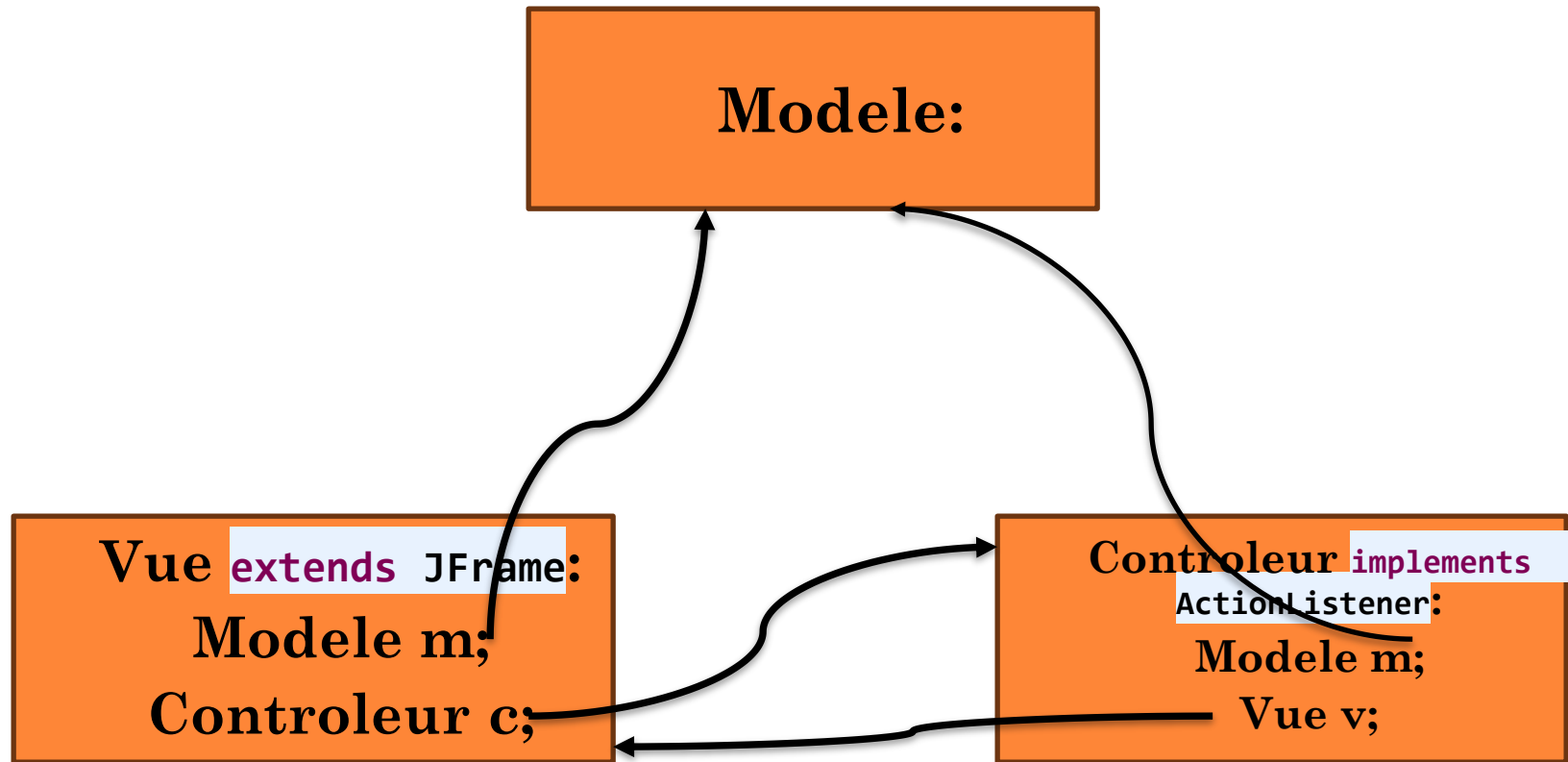
Dans une application structurée en MVC on trouve les étapes suivantes:

- L'utilisateur effectue une action sur l'application (un bouton par exemple),
- Le contrôleur capte l'action et contrôle la cohérence des données ainsi que leurs transformations au modèle. Il peut demander aussi à la vue de changer l'état.
- Le modèle reçoit les données et change l'état ( la valeur d'une variable qui change par exemple).
- Le modèle notifie la vue qu'il faut se mettre à jour. En conséquence, l'affichage de la vue est modifié.

# FLUX D'INFORMATION ENTRE LES COMPOSANTES



# RÉFÉRENCES ENTRE COMPOSANTS



# FLUX D'INFORMATION ENTRE LES COMPOSANTES

- **Vue:** c'est l'IHM qui représente l'état du modèle et ce que l'utilisateur a sous les yeux. Peut être:
  - ✓ Une application graphique Swing, AWT pour Java, Form pour C#.
  - ✓ Une page web.
  - ✓ Une console Windows ou encore un terminal Linux. etc

## L'INTÉRÊT EST DE:

- **Séparer les responsabilités,**
- **Simplifier la maintenance.**

# FLUX D'INFORMATION ENTRE LES COMPOSANTES

- **Modèle:** c'est là que se trouvent les données:
  - ✓ Il décrit les données manipulées par l'application,
  - ✓ Définit les méthodes d'accès aux données,
  - ✓ Fournit les traitements applicables aux données,
  - ✓ Gère les interactions avec la base de données, etc.
- **Contrôleur:** cet objet permet de faire le lien entre la vue et le modèle en répondant sur les actions utilisateurs intervenant sur la vue:
  - ✓ Mettre à jour la vue ou le modèle.
  - ✓ Il n'effectue aucun traitement, ne modifie aucune donnée, il analyse la requête du client pour appeler le modèle adéquat et renvoi la vue correspondante à la demande.

# EXEMPLE : GESTION D'UN POINT DU PLAN

- Créer une application qui permet à l'utilisateur d'**entrer les coordonnées du point.**
- L'application doit : **Afficher les coordonnées du point.**

# EXEMPLE : GESTION D'UN POINT DU PLAN

- `public class MauvaiseClassePoint {`
- `private float x, y;`
- `public MauvaiseClassePoint(float x, float y) {`
- `this.x = x; this.y = y; }`
- `public float getAbscisse () { return this.x;}`
- `public float getOrdonnee() { return this.y;}`
- `public void saisirPoint() { /*...*/ }`
- `public void afficherPoint(){ /*...*/ }`
- `public void activerVuePoint(){ /*...*/ }`
- `public void saisirAbscisse(){ /*...*/ }`
- `public void saisirOrdonnee(){ /*...*/ }`
- `}`



# EXEMPLE : GESTION D'UN POINT DU PLAN

```
○ public class MauvaiseClassePoint {  
○ private float x, y;  
  
○ public MauvaiseClassePoint(float x, float y) {  
○ this.x = x; this.y = y; }  
  
○ public float getAbscisse () { return this.x;}  
○ public float getOrdonnee() { return this.y;}  
  
○ public void saisirPoint() { /*...*/}  
○ public void afficherPoint(){ /*...*/}  
○ public void activerVuePoint(){ /*...*/}  
○ public float saisirAbscisse(){ /*...*/}  
○ public float saisirOrdonnee(){ /*...*/}  
○ }
```

Gestion  
d'un point =

**Modèle**

Gestion  
d'interactions  
avec  
l'utilisateur  
=  
**Vue**

# EXEMPLE : GESTION D'UN POINT DU PLAN

*saisirPoint()* correspond à une requête d'utilisateur qui est proposée par la vue

- `public void saisirPoint() {`

- `float unX = this.saisirAbscisse();`

- `float unY = this.saisirOrdonnee();`

- `This.modifierPoint(unX, unY);`

- `}`

Gérer par la vue  
(interactions avec  
l'utilisateur)

L'interprétation  
relève du  
*contrôleur*

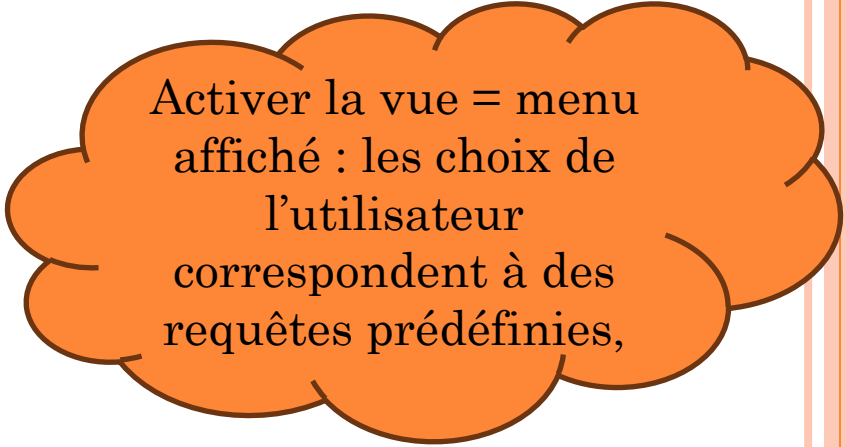
Instruction s'adressant au  
*modèle* pour affecter les  
coordonnées saisies au Point  
courant

# LA CLASSE APPLICATION QUI ACTIVE LA VUE

```
○ public class Application {  
  
○ public static void main (String[] args){  
○     MauvaiseClassePoint p = new MauvaiseClassePoint(2,3);  
○     p.activerVuePoint();  
○ } }
```

```
***** MENU POINT *****
```

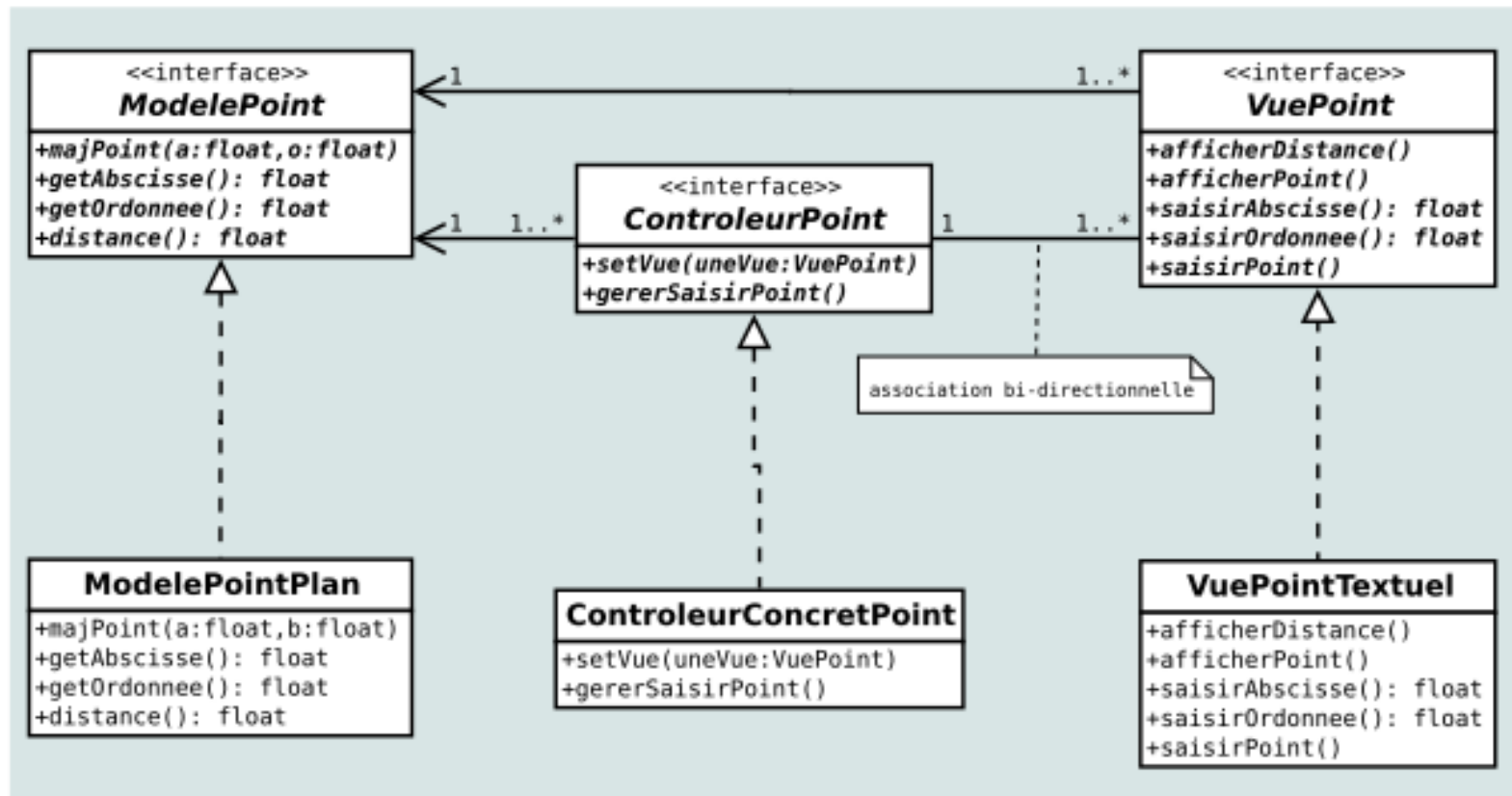
```
1 - entrer/modifier le point  
2 - sortir du programme
```



Activer la vue = menu  
affiché : les choix de  
l'utilisateur  
correspondent à des  
requêtes prédéfinies,

**Constat: la classe Application n'interagit qu'avec la vue**

# UTILISATION ET RÉALISATION DE MVC



il peut y avoir plusieurs vues et plusieurs contrôleurs associés au même modèle  
(chaque vue est en générale associée à un contrôleur)

# UTILISATION ET RÉALISATION DE MVC

## Classes:

- `ModelePointPlan;`
- `VuePointTextuel;`
- `ControleurConcretPoint;`

- `public class ModelPointPlan implements ModelePoint{`
- `private float x,y;`
- `public ModelPointPlan(){}`
- `public ModelPointPlan(float x,float y){`
- `this.x=x;`
- `this.y=y;`
- `}`
- `public float getAbscisse(){`
- `return this.x;`
- `}`
- `public float getOrdonnee(){`
- `return this.y;`
- `}`
- `public void modifierPoint(float ab, float or) {`
- `this.x=ab;`
- `this.y=or; }`
- `}`

```

○ import java.util.Scanner;
○ public class VuePointTextuel implements VuePoint {
○     private ControleurConcretPoint controleur;
○     private ModelPointPlan model;
○
○     public VuePointTextuel(ControleurConcretPoint co, ModelPointPlan mo){
○         this.controleur=co; this.model=mo; }
○     public void activerVuePoint(){
○         saisirPoint();
○         afficherPoint();}
○     public void saisirPoint() {
○         System.out.println ("donner Les coordonnées d'un point");
○         this.controleur.gererSaisirPoint();}
○     public float saisirAbscisse(){
○         Scanner s = new Scanner(System.in);
○         return s.nextFloat();}
○     public float saisirOrdonnee(){
○         Scanner s = new Scanner(System.in);
○         return s.nextFloat();}
○     public void afficherPoint(){
○         System.out.println("abscisse = " + this.model.getAbscisse());
○         System.out.println("ordonnee = " + this.model.getOrdonnee());
○     }
○ }

```

Association avec le contrôleur et le modèle pour afficher les données qui se trouve dans le model

La vue délègue au contrôleur l'interprétation de la requête

La vue interroge le modèle pour afficher certaines données

```

○ public class ControleurConcretPoint implements ControleurPoint {
○     private ModelPointPlan model;
○     private VuePointTextuel vue;
○
○     public ControleurConcretPoint(ModelPointPlan unModelPoint){
○         this.model = unModelPoint;
○     }
○
○     public void setVue (VuePointTextuel uneVuePoint){
○         this.vue = uneVuePoint;
○     }
○     public void gererSaisirPoint (){
○         float abs = this.vue.saisirAbscisse();
○         float ord = this.vue.saisirOrdonnee();
○
○         this.model.modifierPoint(abs, ord);
○     }
○ }

```

Si plusieurs vues sont contrôlées alors déclarez un `ArrayList< VuePoint>`

Pour changer la vue devant être contrôlée

Interprétation de la requête `saisirPoint()` de la vue

S'adresse à la vue pour que l'utilisateur entre l'abscisse et l'ordonnée

S'adresse au modèle pour enregistrer les nouvelles coordonnées



- `public class Application {`

- `public static void main(String[] args) {`

- `ModelPointPlan m = new ModelPointPlan();`

- `ControleurConcretPoint c = new ControleurConcretPoint(m);`

- `VuePointTextuel v = new VuePointTextuel(c,m);`

- `c.setVue(v);`

- 

- `v.activerVuePoint();`

- `}`

- `}`

On crée le contrôleur et  
on lui associe le modèle  
déjà crée

Association de la  
vue au controleur

On crée une et on  
lui associé le  
controleur et le  
modèle

Activation de la vue  
qui attend les  
requêtes de  
l'utilisateur

<terminated> Application (5) [Java Application] C:\Program Files\Java\jre1.8.0\_251\bin\javaw.exe (21 avr. 202

donner les coordonnées d'un point

2

3

abscisse = 2.0

ordonnee = 3.0

# AVANTAGES ET INCONVÉNIENTS DE MVC

## Avantages

- Séparation des responsabilités.
- Le modèle peut être partagé parmi divers vues et contrôleurs.

## Inconvénients

- L'architecture peut augmenter la complexité du système.

**The END ... Thank you 😊**