

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

”

Gestion des événements

Gestion des événements

- Le principal objectif d'une application graphique est la programmation événementielle c'est-à-dire l'utilisateur peut déclencher des événements et réagir à ce qui se passe dans la fenêtre.
- Il utilise le clavier et la souris pour intervenir sur le déroulement du programme.
- Le système d'exploitation engendre des événements à partir des actions de l'utilisateur.
- Le programme doit lier des traitements à ces événements.

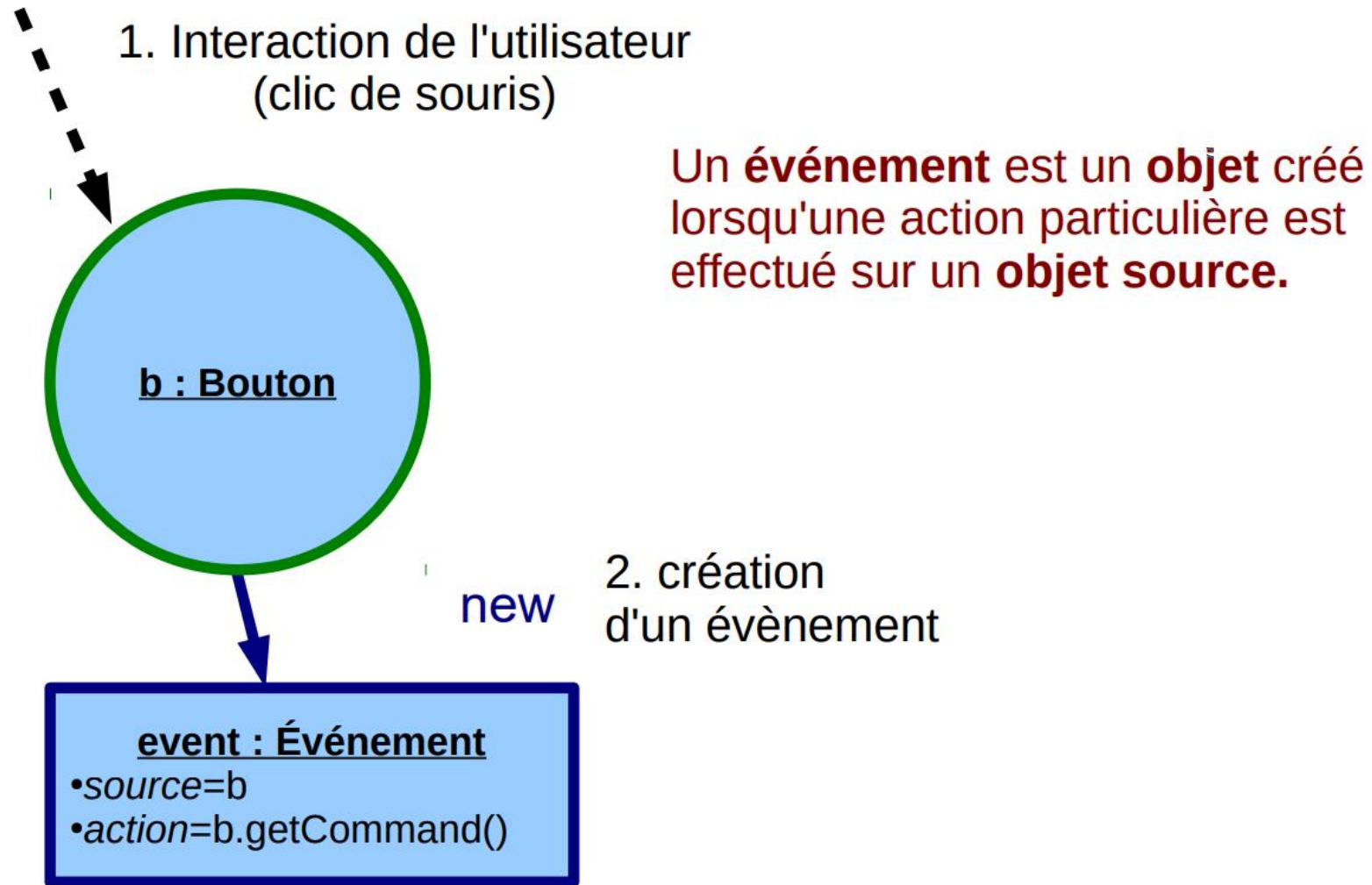
Gestion des événements

- Des événements provoqués par la souris ou le clavier peuvent concerner un composant graphique et le faire réagir. Si on veut que le composant soit sensible à certaines catégories d'événements, il faut le demander explicitement en mettant un objet à l'écoute de l'événement (un Listener) et en indiquant ce qui doit être fait lorsque l'événement survient.

Exemples d'événements

- appui sur un bouton de souris ou une touche du clavier.
- relâchement du bouton de souris ou de la touche. `
- déplacer le pointeur de souris.
- clic de souris: clic sur un bouton par exemple.
- choisir un élément dans une liste.
- modifier le texte d'une zone de saisie.

Événements / Listener



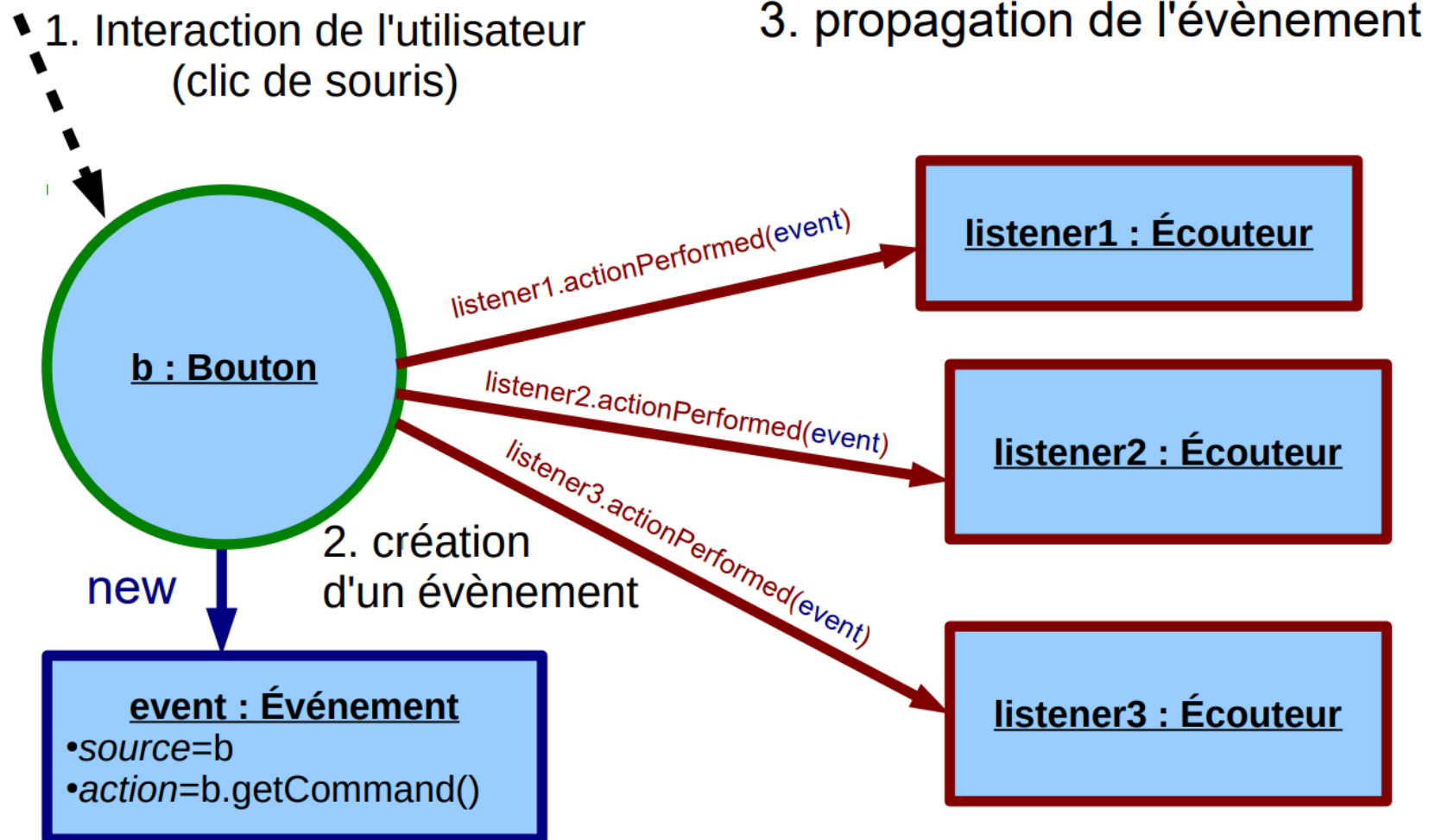
Événements / Listener

- Un composant qui crée des événements est appelé **source** (Exemple: bouton Ajouter de notre calculatrice).
- Le composant **source délègue** le traitement de l'événement au composant **auditeur (listener)**, c'est le composant qui traite l'événement.
- Chacun des composants graphiques a ses écouteurs (*listeners*)

Événements / Listener

- Un composant peut avoir plusieurs écouteurs (par exemple, 2 écouteurs pour les clics de souris et un autre pour les frappes de touches du clavier).
- Un écouteur peut écouter plusieurs composants.

Événements / Listener



Un écouteur est un objet réagissant aux évènements d'un objet source. Pour qu'un objet puisse écouter un objet source, il doit s'enregistrer auprès de celui ci

Événements / Listener

- **Question:** Quel message sera envoyé par le composant à ses écouteurs pour les prévenir que l'événement qui les intéresse est arrivé ?
- **Réponse :** à chaque type d'écouteur correspond une interface que doit implémenter la classe de l'écouteur .

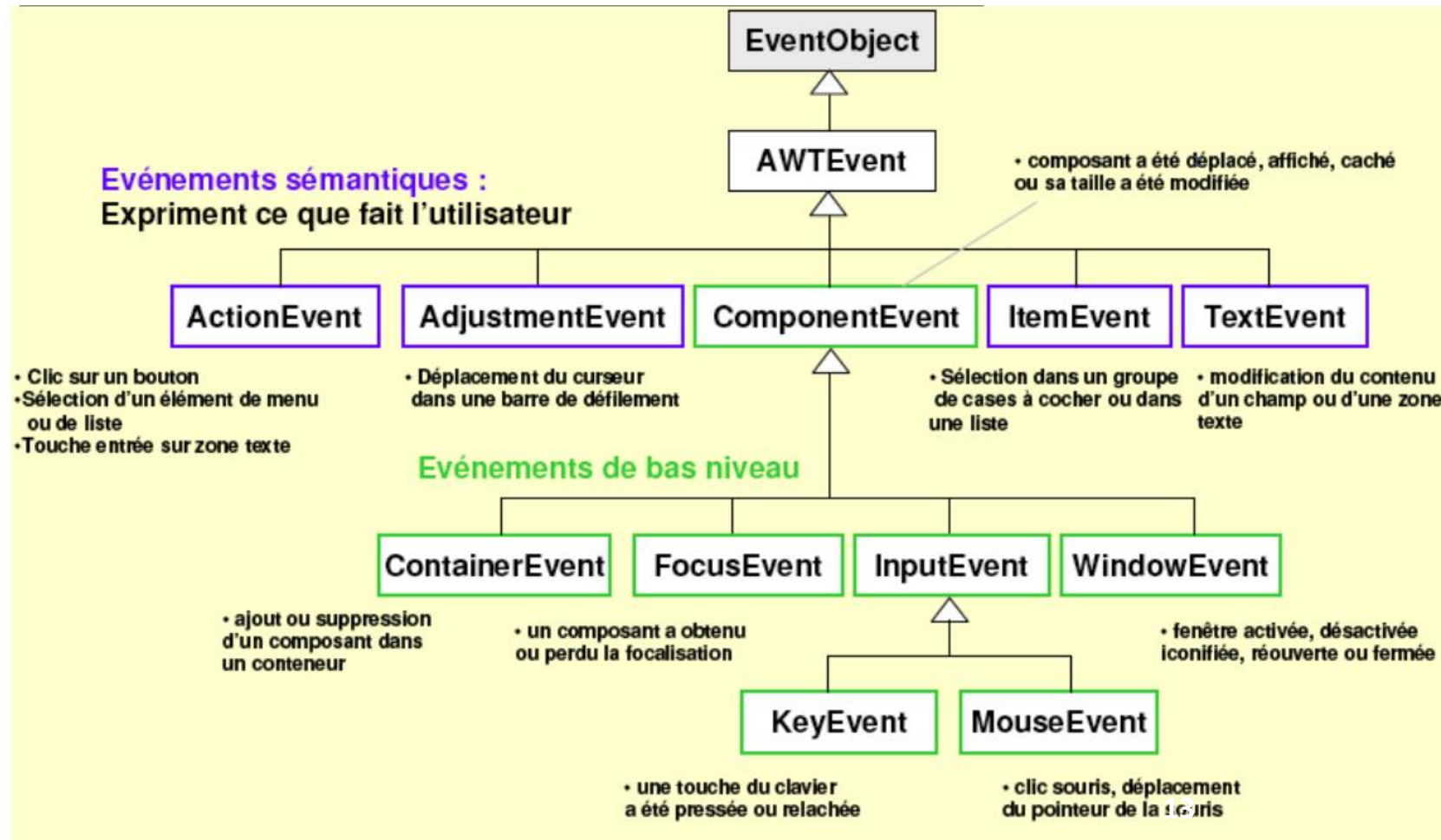
Événements / Listener

- Tout **événement** hérite de la classe **EventObject**. Les différents types d'événements sont représentés par des classes différentes
- Tout **listener** correspond à une interface qui hérite de **EventListener**.
- Toute classe désirant recevoir des notifications d'un type d'événement donné devra implémenter l'interface correspondante :
 - ActionEvent ActionListener
 - MouseEvent MouseListener
 - KeyEvent KeyListener
 - WindowEvent WindowListener
 - ...

Classes d'événements / Listener

- **Les sources :**
 - Boutons : JButton, JRadioButton, JCheckBox, JToggleButton
 - Menus : JMenuItem, JMenu, JRadioButtonMenuItem, JCheckBoxMenuItem
 - Texte : JTextField
 - ...
- **Les listeners:**
 - Il faut implémenter l'interface qui correspond au type de l'événement !!

Une partie de la hiérarchie des événements AWT



Classe : **ActionEvent**

- Cette classe décrit des événements qui vont le plus souvent déclencher un traitement (une *action*) :
 - clic sur un bouton.
 - return dans une zone de saisie de texte.
 - choix dans un menu.
- Ces événements sont très fréquemment utilisés et ils sont très simples à traiter.

Interface: **ActionListener**

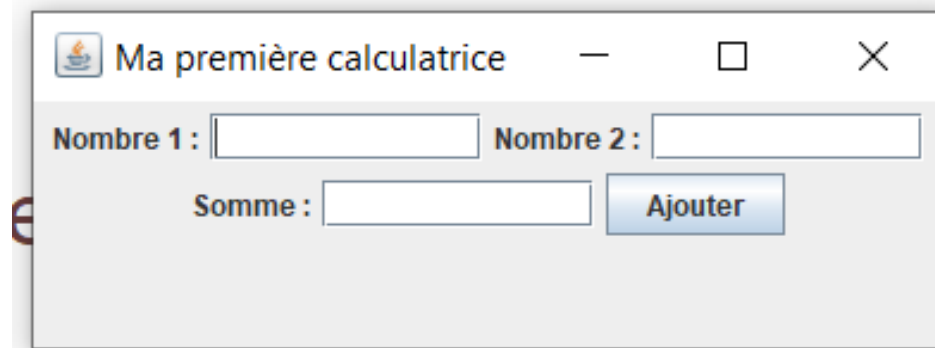
- Un objet *ecouteur intéressé par les événements de type « action »* (classe **ActionEvent**) **doit** appartenir à une classe qui implémente l'interface **java.awt.event.ActionListener**

Exemples d'utilisation : bouton

- Exemple d'un **bouton** :
 - Un bouton est un élément graphique sur lequel l'utilisateur peut cliquer pour déclencher une action.
 - Le bouton ne fait rien tant que l'utilisateur n'a pas cliqué dessus.
 - Lors d'un clique un événement est créé ... reste à le traiter !

Ecouter un bouton

- Le bouton « **Ajouter** » de notre calculatrice de TP4 (partie 1) n'a été pas encore prêt à réagir afin d'additionner les deux nombres (voir projet CalculatriceVer1).



Ecouter un bouton

- Si on veut que le bouton **Ajouter** soit sensible au clic et affiche la somme des deux nombres, il faut le demander explicitement en **ajoutant un objet à l'écoute de l'événement** (un Listener) et en indiquant ce qui doit être fait lorsque l'événement survient.

Comment Activer le bouton Ajouter

- Créer une sous-classe qui implémente **l'interface ActionListener**.
- Implémenter la seule méthode abstraite de l'interface ActionListener (actionPerformed).
- L'implémentation de la méthode actionPerformed (ActionEvent e) sera exécuté suite au clic sur le bouton.
- Faire une liaison entre notre bouton **Ajouter** et **l'objet** de la sous-classe créée, en utilisant la methode **addActionListener (Object o)** sur le bouton.

Sous-classe: EventAjouter implements ActionListener

```
• /*****Sous-classe*****/
• class EventAjouter implements ActionListener{
• @Override
• public void actionPerformed(ActionEvent arg0) {
• // TODO Auto-generated method stub
• try{
• int num1, num2, somme;
• num1=Integer.parseInt(entrée1.getText());
• num2=Integer.parseInt(entrée2.getText());

• somme=num1+num2;
• resultat.setText(Integer.toString(somme));
• }
• catch (Exception e){
• JOptionPane.showMessageDialog(null, "entrer un nombre");}
• }}
• /***Liaison entre le bouton Ajouter et l'objet de la sous-classe***/
• lancer.addActionListener(new EventAjouter());
```

Interface: MouseListener

- L'interface MouseListener prend en compte:
 - L'appui ou le relâchement du bouton de la souris dans l'espace du composant,
 - l'entrée ou la sortie du curseur de la souris de cet espace, ou un clic sur le composant.
- A noter qu'un clic génère aussi les événements appui et relâchement.
- Cinq méthodes de l'interface MouseListener traitent ces 5 événements :

Interface: MouseListener

- void mouseClicked (MouseEvent evt) : clic sur l'espace du composant.
- void mouseEntered (MouseEvent evt) : le curseur entre dans l'espace du composant.
- void mouseExited (MouseEvent evt) : le curseur sort de l'espace du composant.
- void mousePressed (MouseEvent evt) : bouton de la souris appuyé sur le composant.
- void mouseReleased (MouseEvent evt) : bouton de la souris relâché sur le composant.

Interface: MouseListener

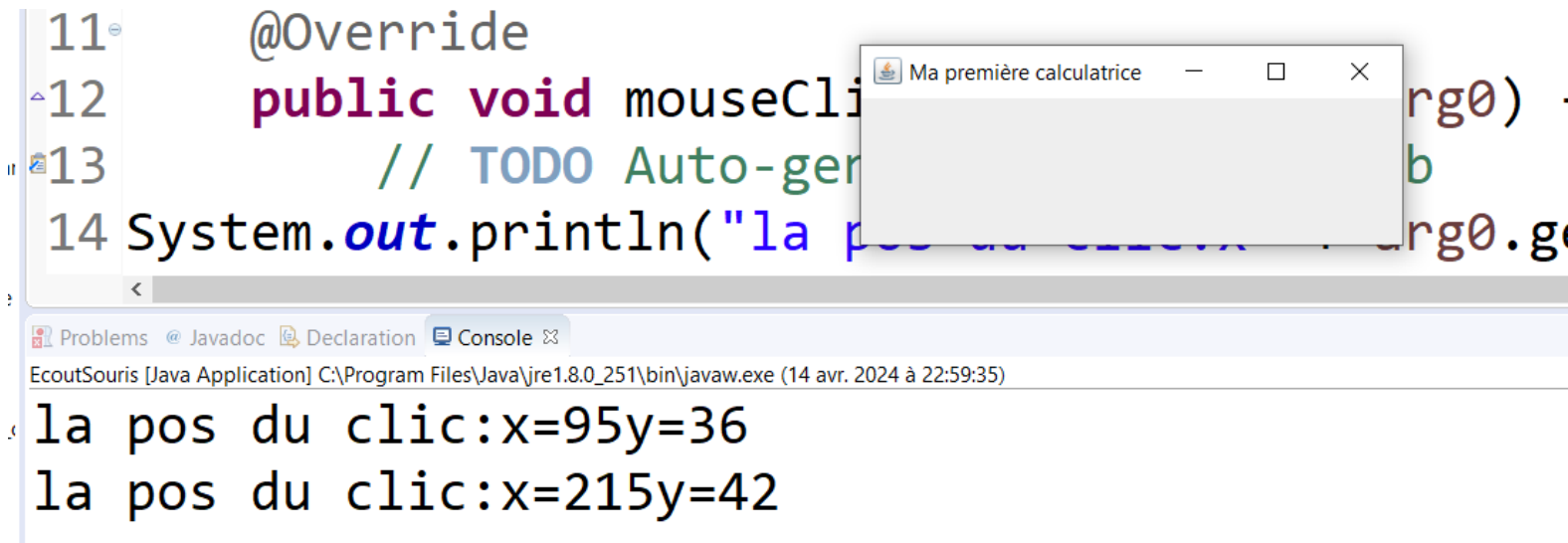
- `/** Un programme qui permet d'afficher la position (X,Y) de clic souris dans le panel ****/`
- `import java.awt.event.MouseEvent;`
- `import java.awt.event.MouseListener;`
- `import javax.swing.JFrame;`
- `import javax.swing.JPanel;`
- `public class EcoutSouris implements MouseListener {`
- `@Override`
- `public void mouseClicked(MouseEvent arg0) {`
- `// TODO Auto-generated method stub`
- `System.out.println("La pos du clic : x="+ arg0.getX()+"y="+arg0.getY());`
- `}`
- `@Override`
- `... etc`

Interface: MouseListener

- `public static void main (String [] args){`
- `JFrame fenetre= new JFrame("Ma première calculatrice");`
- `fenetre.setSize(400,150);`
- `fenetre.setVisible(true);`
- `fenetre.setLocationRelativeTo(null);`
- `JPanel p = new JPanel ();`
- `fenetre.add(p);`
- `EcoutSouris ecout = new EcoutSouris ();`
- `p.addMouseListener (ecout);`
- `}`

Interface: MouseListener

```
11 @Override
12 public void mouseClicked(MouseEvent e) {
13     // TODO Auto-generated method stub
14     System.out.println("la pos du clic:x=" + e.getX() + "y=" + e.getY());
15 }
```



Ma première calculatrice

la pos du clic:x=95y=36
la pos du clic:x=215y=42

L'interface `MouseEventListener`

- `MouseEventListener` prend en compte les déplacements de la souris quand le curseur se trouve dans l'espace du composant ; il gère deux événements : le déplacement du curseur sans appui sur le bouton, et le déplacement avec bouton de la souris constamment appuyé. Deux méthodes de l'interface `MouseEventListener` traitent ces 2 événements :
- **`void mouseMoved (MouseEvent evt)`** : survol du composant, bouton non appuyé.
- **`void mouseDragged (MouseEvent evt)`** : survol du composant, bouton appuyé.

“

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