UNIVERSITÉ ABDELMALEK ESSAADI

FACULTÉ POLYDISCIPLINAIRE DE LARACHE

Département Informatique

FILIÈRE SMI

Module : M33

Semestre : S6

Intitulé : Programmation événementielle en JAVA

PARTIE I

ASSURER PAR : PR HAIMOUDI

ANNÉE UNIVERSITAIRE 2021/2022

Les concepts de la programmation événementielle

- le programme réagit en fonction d'événements utilisateurs, systèmes ou programmeurs.
- La façon de coder s'en ressent car le programmeur ne sait pas à l'avance dans quel ordre, certaines actions vont être effectuées.
- Le principe général est de recenser les différents événements à traiter et d'écrire pour chacun un gestionnaire d'événements.
- Dans ce cours on se restreindra à **la programmation événementielle appliquée aux interfaces graphiques**.
- la conception et la réalisation seront décomposés en trois parties :
  1. **Les différents objets graphiques vus par l'utilisateur.**
  2. **La disposition de ces objets sur l'écran.**
  3. **Les différents événements à traiter sur chaque objet graphique et les écouteurs associés**

Gestion des événements

- Notion d'événement.
- En Java, les événements n'ont pas une valeur physique mais logique.
- Par exemple, un clic souris ne correspond pas toujours à l'émission d'un événement de type MouseEvent.
- L'événement émis suite à un clic de souris dépend du composant sur lequel est fait ce clic.

Composant	Evénement émis
Canevas (*Canvas*)	MouseEvent
Liste (*List*)	ItemEvent
Bouton (*Button*), Menu (MenuItem)	ActionEvent

Gestion des événements

- La hiérarchie de classes *EventObject*

  Les classes de cette hiérarchie encapsulent les événements qui se produisent notamment dans une interface graphique.

  - **Clic souris**
  - **Frappe d'une touche**
  - **Changement de la taille d'une fenêtre**.
- Lorsque l'on développe une interface graphique, on a besoin d'écouter les messages que l'utilisateur envoie à l'interface, on doit permettre à l'interface de gérer les événements provoqués par l'utilisateur (clavier, souris) et par le système d'exploitation (horloge).
- Les événements que l'on va traiter vont générer des actions sur l'interface, on des traitements spécifiques.

Gestion des événements

- Quelques événements.

Il existe des évènements de bas niveau qui permettent par exemple de savoir si le bouton droit de la souris est enfoncé, relaché, si la souris a bougé, ou si une touche particulière est enfoncée ou relachée, si un composant vient d'être activé (Focus)...

De bas niveau :

FocusEvent : prise et perte de focus,

KeyEvent, MouseEvent : événements clavier ou souris.

il existe aussi des évènements de haut niveau , qui sont activées dés qu'il y a n'importe quel type d'action (clavier ou souris) sur un composant.

De haut niveau :

ActionEvent : action spécifique à un composant (clic bouton, sélection d'un menu...),

TextEvent : changement de valeur d'un champ de saisie.

Gestion des événements

- Informations contenues par les événements.

Les événements sont capables de fournir des informations concernant leur contexte d'apparition.

MouseEvent : position de la souris,

KeyEvent : valeur de la touche enfoncée, etc.

- La même action sur des composants différents ne provoque pas le même événement.

  Un clic gauche sur :un bouton,une liste,une zone de saisie ne provoque pas le même événement.

- Certains événements transportent des informations :

  Un caractère saisi au clavier provoque l'événement keyEvent qui transmet par son paramètre le caractère effectivement saisi au clavier

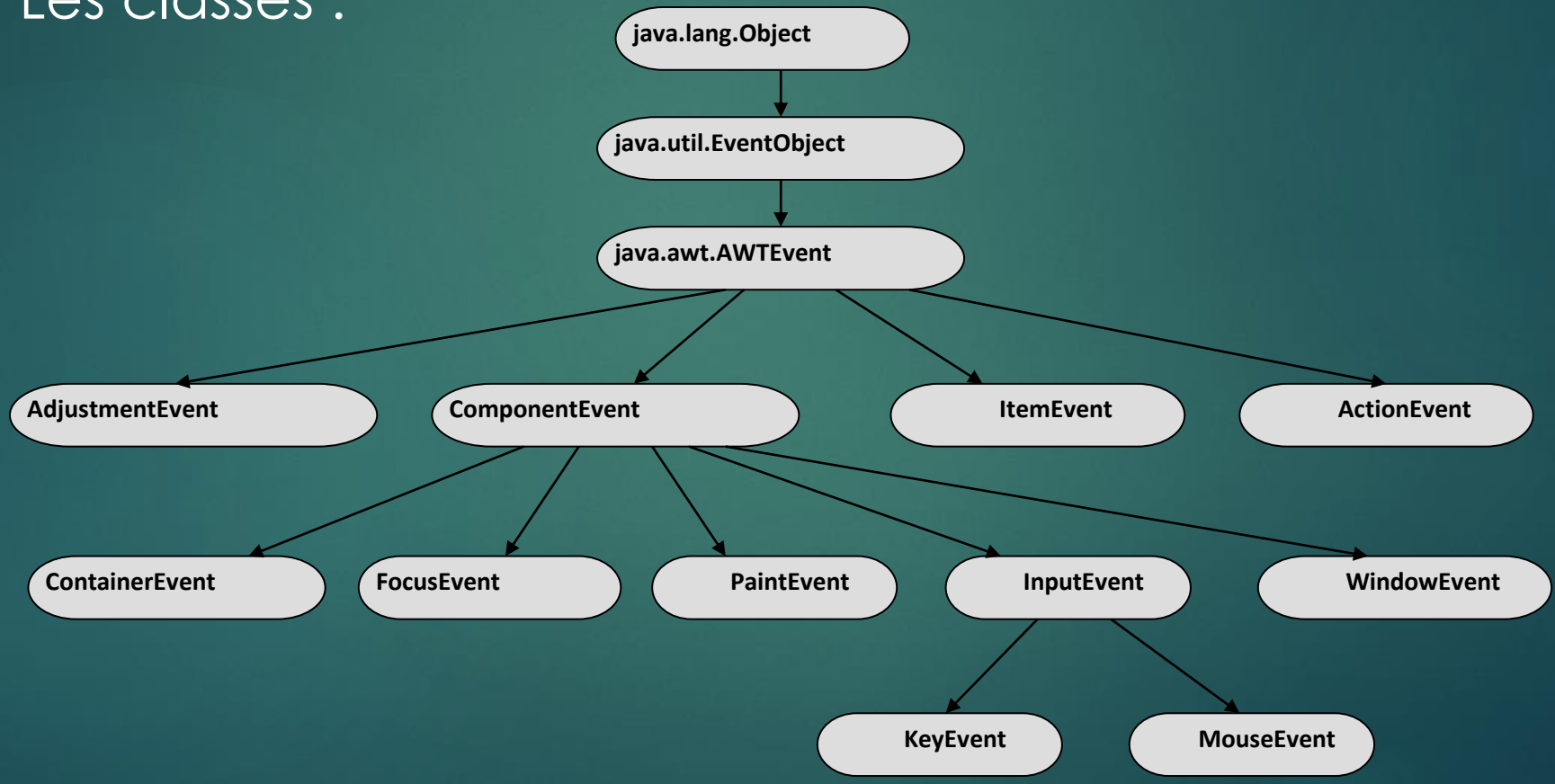
- Intercepter des événements

  Chaque composant ne gère pas forcément lui-même les événements qu'il génère. Il délègue cette gestion à des objets particuliers : les contrôleurs (appelés aussi listener).

Gestion des événements

- Les classes et Interfaces de gestion des évènements.

Les classes :

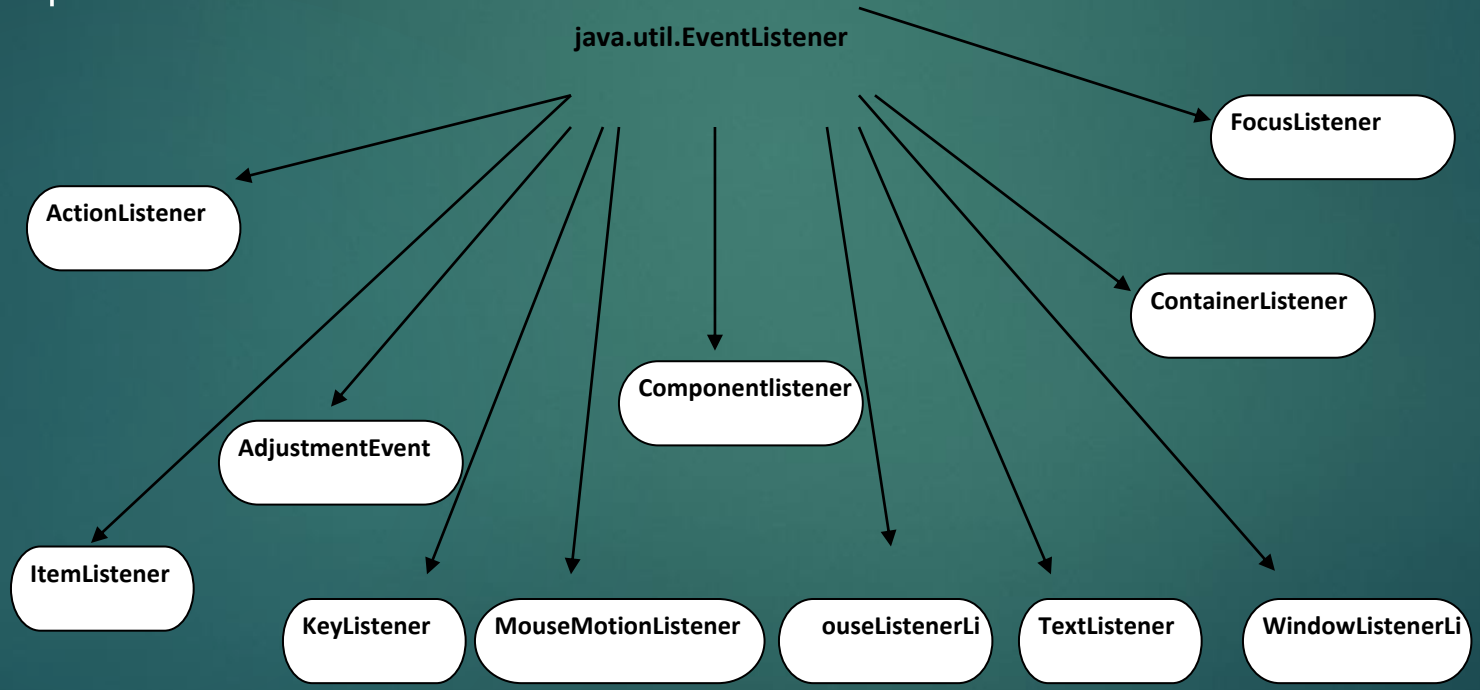


Gestion des événements

- Les classes et Interfaces de gestion des évènements.

Les interfaces de type EventListener :

En fonction des événements qu'ils gèrent, les contrôleurs doivent implémenter des interfaces de la hiérarchie EventListener.



Gestion des événements

- Les Interfaces bas Niveaux : Exemples

  l'interface MouseMotionListener gère le déplacement de la

```
public interface MouseMotionListener extends EventListener {
   // interface définissant les fonctions à associer au déplacement
de la souris
   public void mouseMoved(MouseEvent e);
     //  appelé lors du déplacement de la souris
   public void mouseDragged(MouseEvent e);
     //  appelé lors du déplacement de la souris, clic gauche
enfoncé
}
```

```
public interface MouseListener extends EventListener {
        public void mouseClicked(MouseEvent e);
        public void mouseEntered(MouseEvent e);
        public void mouseExited(MouseEvent e);
        public void mousePressed(MouseEvent e);
        public void mouseReleased(MouseEvent e);
}
```

Gestion des événements

- **Les Interfaces haut Niveaux : Exemple**

l'interface ActionListener gère tout type d'opération sur le composant

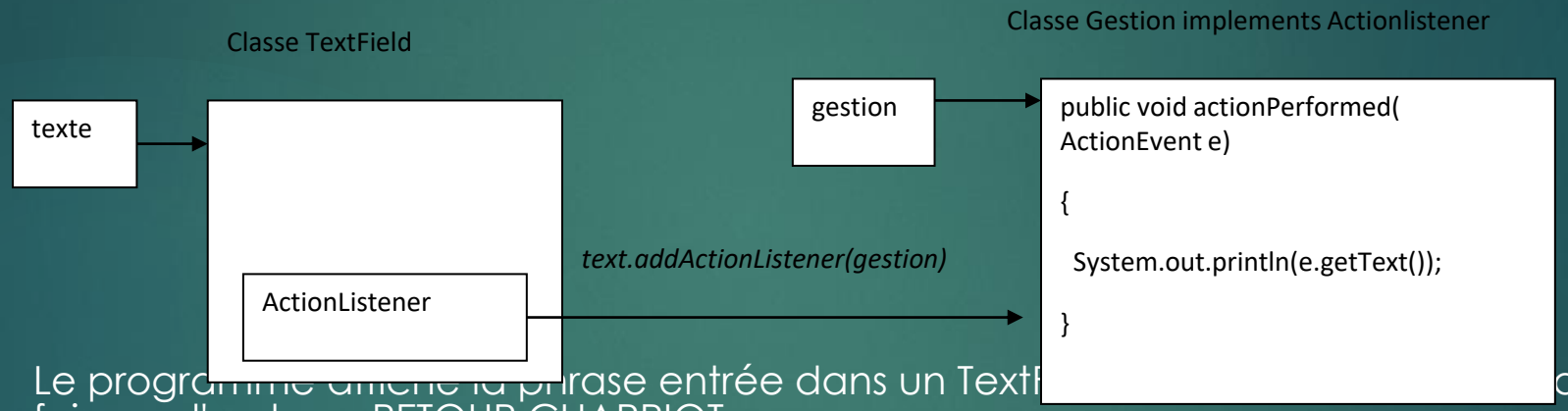
```
public interface ActionListener extends EventListener {

    public void actionPerformed(ActionEvent e);
}
```

- Les méthodes définies dans les interfaces seront appelées à l'émission d'un événement. A ce moment, un objet événement est passé en paramètre. Son type dépend de l'interface associée.

Gestion des événements

- Associer un contrôleur à un composant

1. Exemple1 : Utilisation de ActionEvent.



- Le programme affiche la phrase entrée dans un TextF que fois que l'on tape RETOUR CHARRIOT
- créer un objet texte de la classe TextField.
- grace à la méthode addActionListener lui associé la classe de gestion de l'évènement.
- La classe de gestion de l'évènement implémente les méthodes de l'interface ActionListener.
- Redéfinir la méthode actionPerformed de l'interface ActionListener.
- Le code de cette méthode sera appélé à chaque fois, que l'on tapera la touche Entrée

Gestion des événements

- Exemple.

```
import java.awt.*;
import java.awt.event.*;
public class EssaiEvenement extends Frame{
    public EssaiEvenement(String s) {
        super(s);
                TextField text = new        TextField(20);
        GestionEv gestion = new GestionEv();
        text.addActionListener(gestion);
        setSize(200,200);
        setLayout(new FlowLayout());
        add(text);
        show();
    }
   public static void main(java.lang.String[] args) {
     EssaiEvenement e= new
EssaiEvenement("EssaiEvenement");
    e.addWindowListener(new WindowAdapter(){
    public void windowClosing(WindowEvent e)
{System.exit(0);}
    });
  }
}
```

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

public class GestionEv implements
ActionListener {
   public void
actionPerformed(ActionEvent e){

System.out.println(e.getActionCommand()
);
   }
}
```

Gestion des événements

- ***Récapitulatif***

  Si on ajoute l'instruction addActionListener , cela signifie que le composant peut être source d'événements de type ActionEvent.

  Le paramètre indique qui gère ces événements. (càd qui est le contrôleur)

  - **addActionListener(ActionListener l)** : permet d'intercepter le clic de la souris sur le bouton.
  - **addComponentListener(ComponentListener l) :** permet d'intercepter tous les événements relatifs aux classes dérivées de Composant.
  - **addFocusListener(FocusListener l)** permet de gérer la prise et la perte du focus sur le composant
  - **addKeyListener(KeyListener l)** permet de détecter l'appuie sur une touche.
  - **addMouseListener(MouseListener l)** permet l'interception de tous les événements liés à la souris.
  - **addMouseMotionListener(MouseMotionListener l)** : gestion des mouvements de la souris.