

GRAU D'ENGINYERIA INFORMÀTICA

# PROGRAMACIÓ II

**Bloc 3:**

## Programació Orientada a Events (2)

**Sergio Sayago (basat en material de Laura Igual)**

Departament de Matemàtiques i Informàtica

Facultat de Matemàtiques i Informàtica

Universitat de Barcelona

# Índex Bloc 3:

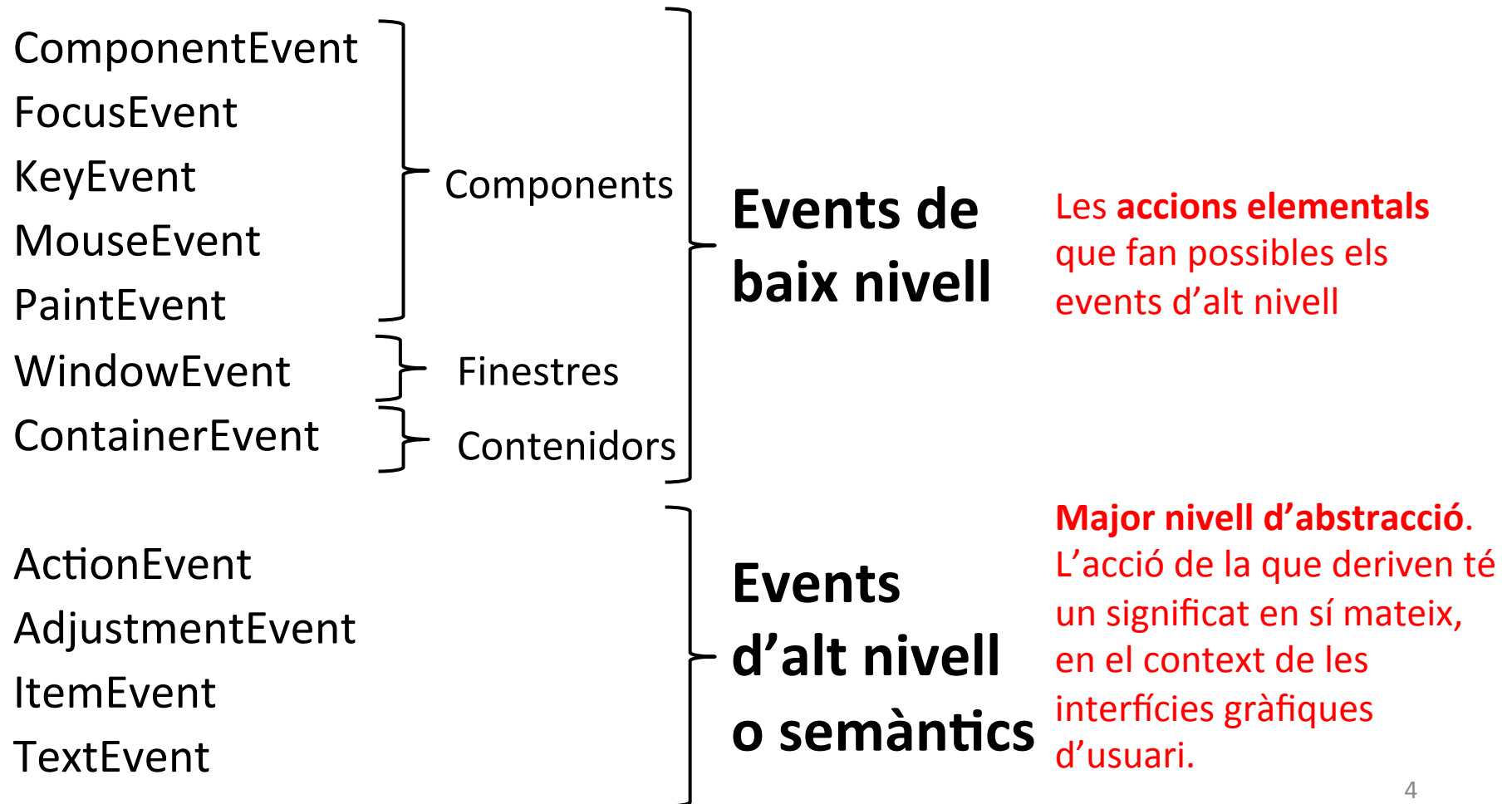
## Programació Orientada a Events

- Mecanismes d'interacció
  - Interacció mitjançant flux seqüencial
  - Interacció mitjançant programació orientada a events
- Programació d'Interfícies Gràfiques d'Usuari
- Model de gestió d'events: Exemple d'implementació d'una finestra.
- **Events i Listeners**
- **Components i Contenidors**
- Classes adapter i classes internes: Exemple d'implementació d'una finestra que es tanca.
- Layout manager
- Mes sobre swing components: Exemples
- Look and feel
- Panells i gràfics
- Animacions

# **COMPONENTS, EVENTS I LISTENERS**

# Events AWT

En Java els events poden ser d'alt o de baix nivell.



# Events AWT

En Java els event poden ser d'alt o de baix nivell.

ComponentEvent  
FocusEvent  
KeyEvent  
MouseEvent  
PaintEvent  
WindowEvent  
ContainerEvent

*Es produeixen amb les operacions  
elementals del ratoli, teclat,  
containers i windows.*

ActionEvent (*clicar sobre botons o escollir comandos en menús*)

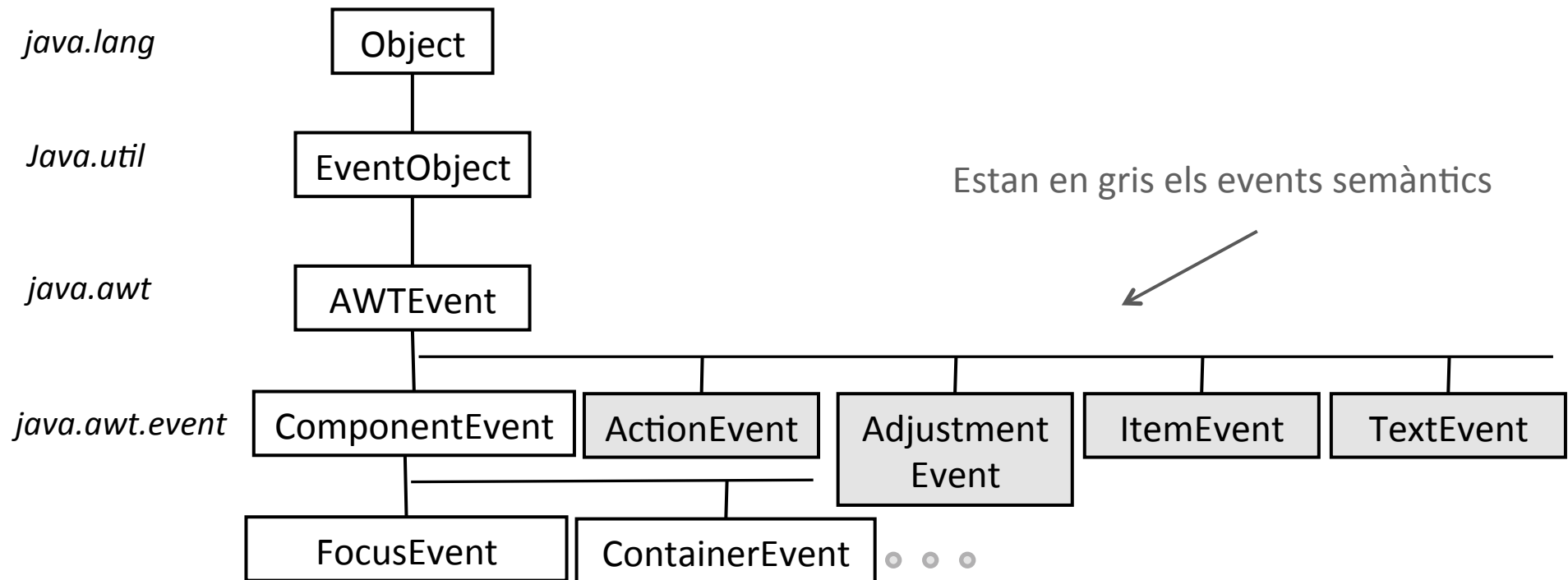
AdjustmentEvent (*canviar valors en barres de desplaçament*).

ItemEvent (*seleccionar/desseleccionar valors*)

TextEvent (*canviar el text*)

# Jerarquia d'Events

Tots els events de Java són objectes de classes que pertanyen a una determinada jerarquia de classes:



# Events semantics

- No són disparats per tots els components
- Exemple 1: **ItemEvent**
  - ◆ Disparat per JComboBox
  - ◆ No disparat per JButton

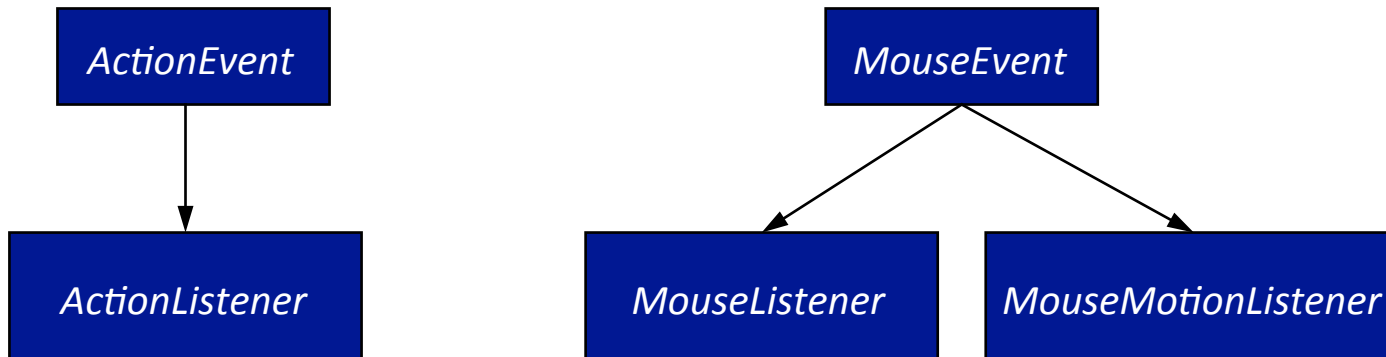
“A semantic event which indicates that an item was selected or deselected”

- Exemple 2: **ActionEvent**
  - ◆ Disparat per JComboBox
  - ◆ Disparat per JButton

“A semantic event which indicates that a component-defined action occurred”

# Escoltadors (Listeners) d'events

- Interfícies que manipulen els events (`java.util.EventListener`).
- Cada classe Event té la seva corresponent interfície Listener
- Pot haver-hi diversos Listeners per al mateix tipus d'events





# Escoltadors (Listeners) d'events

- Cada listener és un objecte que implementa la interfície corresponent al tipus d'event a escoltar:
  - ActionListener
  - WindowListener
  - MouseListener
  - KeyListener
  - FocusListener
  - Altres...

# Events i Interfícies Listener

- Cada tipus d'event té una interfícies Listener associada:

ComponentEvent	ComponentListener
FocusEvent	FocusListener
KeyEvent	KeyListener
MouseEvent	MouseListener, MouseMotionListener
WindowEvent	WindowListener
ContainerEvent	ContainerListener
ActionEvent	ActionListener
AdjustmentEvent	AdjustmentListener
ItemEvent	ItemListener
TextEvent	TextListener

# Listeners

## Exemples de Listeners:

```
public interface ActionListener extends EventListener {  
    public void actionPerformed(ActionEvent e);  
}
```

```
public interface ItemListener extends EventListener {  
    public void itemStateChanged(ItemEvent e);  
}
```

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

Algunes interfícies  
Listener tenen  
més d'un mètode,  
perquè l'event ve  
produït per més  
d'una acció.

# Listeners

```
public interface WindowListener extends EventListener {  
    void windowActivated(WindowEvent e);  
    void windowClosed(WindowEvent e);  
    void windowClosing(WindowEvent e);  
    void windowDeactivated(WindowEvent e);  
    void windowDeiconified(WindowEvent e);  
    void windowIconified(WindowEvent e);  
    void windowOpened(WindowEvent e);  
}
```

```
public interface ComponentListener extends EventListener {  
    public void componentHidden(ComponentEvent e);  
    public void componentMoved(ComponentEvent e);  
    public void componentResized(ComponentEvent e);  
    public void componentShown(ComponentEvent e);  
}
```

# Registre de Listeners

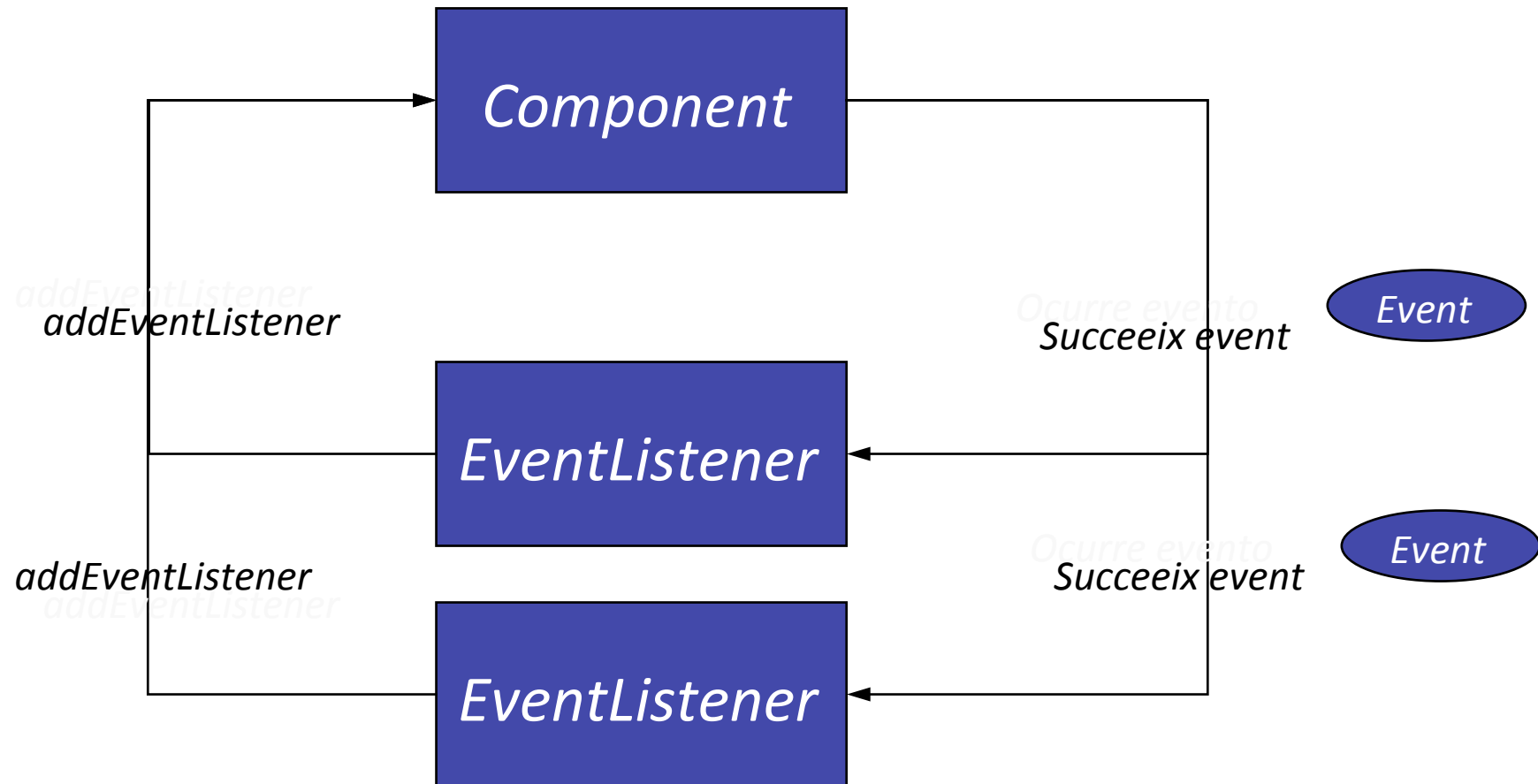
- Un Listener, en primer lloc, ha de registrar-se amb la/les font/s que pugui/n generar els events del seu interès. Ho faran mitjançant un mètode de la forma:

public void **addXXXListener(XXXListener e)**

Per exemple: addActionListener, addItemListener, ...

- Per al mateix event en un component, pot haver diversos Listeners registrats
  - Un event pot provocar nombroses respostes
  - Els events són passats a tots els seus Listeners

# Múltiples Listeners



# Múltiples Listeners

public class ExempleMouseListener implements **MouseListener**, **ActionListener**

```
{
    JButton myButton;
    JFrame myWindow;
    public static void main(String[] args) {
        ExempleMouseListener ex = new ExempleMouseListener();
        ex.go();
    }
    public void go(){
        myWindow = new JFrame();
        myWindow.setTitle("Prova mouse listener");
        myWindow.setSize(200, 200);
        myButton = new JButton();
        myButton.setText("Click me");
        myButton.addMouseListener(this);
        myButton.addActionListener(this);
        myWindow.getContentPane().add(myButton);
        myWindow.setVisible(true);
    }
    (...)
    (...)
    @Override
    public void mouseClicked(MouseEvent e) { }
    @Override
    public void mousePressed(MouseEvent e) {
        myButton.setText("uno");
    }
    @Override
    public void mouseReleased(MouseEvent e) {
        myButton.setText("dos");
    }
    @Override
    public void mouseEntered(MouseEvent e) { }
    @Override
    public void mouseExited(MouseEvent e) {
        myButton.setText("fuera");
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        myButton.setBackground(Color.red);
    }
}
```

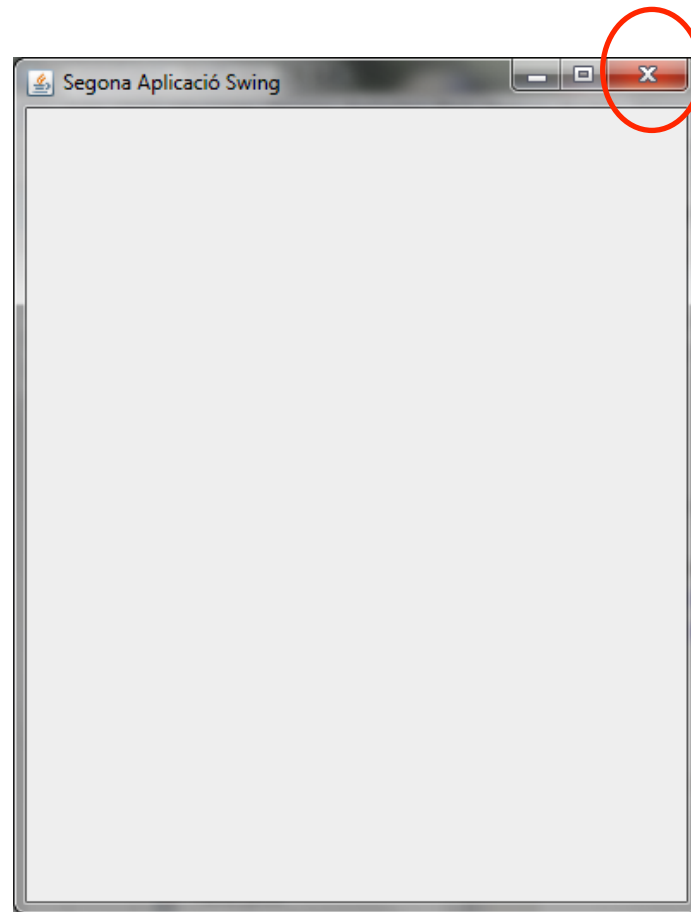
# Índex Bloc 3:

## Programació Orientada a Events

- Mecanismes d'interacció
  - Interacció mitjançant flux seqüencial
  - Interacció mitjançant programació orientada a events
- Programació d'Interfícies Gràfiques d'Usuari
- Model de gestió d'events: Exemple d'implementació d'una finestra.
- Events i Listeners
- Components i Contenedors
- **Classes adapter i classes internes: Exemple d'implementació d'una finestra que es tanca.**
- Layout manager
- Mes sobre swing components: Exemples
- Look and feel
- Panells i gràfics
- Animacions



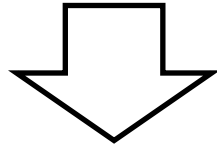
## Exemple 3: FINESTRA que es tanca



## Exemple 3: dos mètodes de creació de finestres

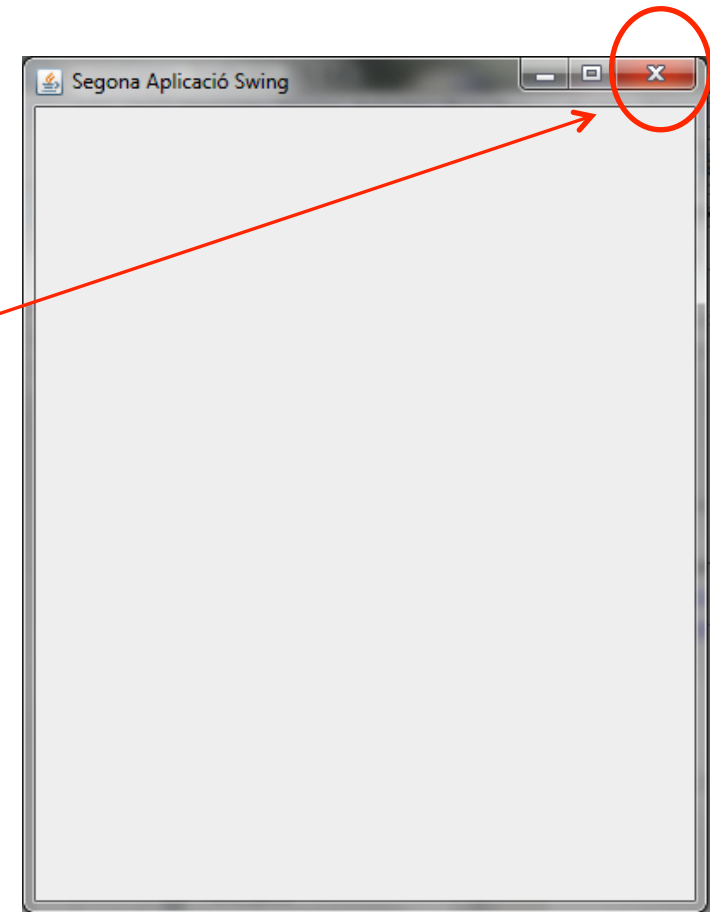
A l'exemple 1 vam crear una aplicació senzilla de dues maneres.

Però, en cap dels dos casos l'aplicació termina quan fem click en el botó de tancar de la finestra



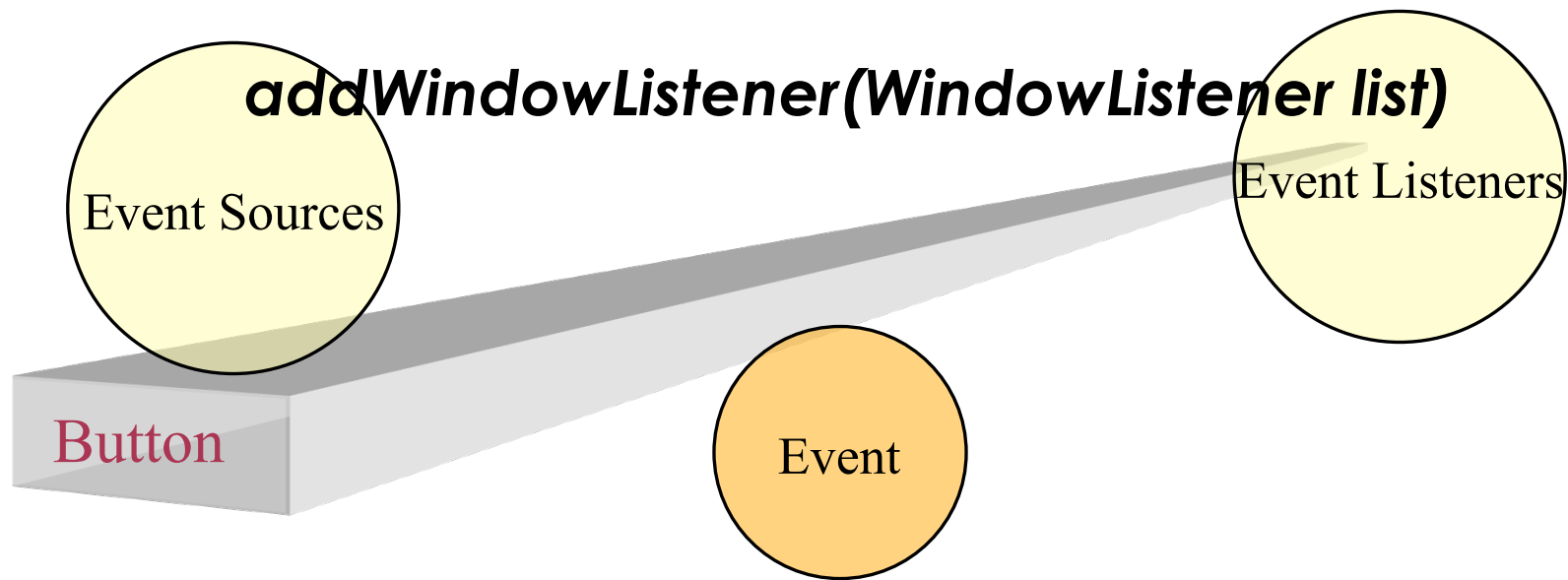
Caldrà relacionar el botó per tancar la finestra amb l'acció de terminar l'aplicació

Caldrà capturar els events que es llancen



## Exemple 3: Afegim Listener

- Volem fer que l'aplicació termini quan es tanca la finestra.
- Per això, definim un event i el gestionem.



- A continuació mostrem **quatre** implementacions possibles.

# Recordem: WindowListener

```
public interface WindowListener extends EventListener {  
    void windowActivated(WindowEvent e);  
    void windowClosed(WindowEvent e);  
    void windowClosing(WindowEvent e);  
    void windowDeactivated(WindowEvent e);  
    void windowDeiconified(WindowEvent e);  
    void windowIconified(WindowEvent e);  
    void windowOpened(WindowEvent e);  
}
```

<https://docs.oracle.com/javase/10/docs/api/java/awt/event/WindowListener.html>

## Exemple 3 (a)

```
// Fitxer FinestraTancable.java
import java.awt.event.*;
import javax.swing.*;
class FinestraTancable extends JFrame implements WindowListener {
    // Constructor
    public FinestraTancable () {
        this.setTitle("La meva finestra tancable");
        this.setSize( 300, 200);
        // causa events de window per a ser enviat al objecte window listener
        this.addWindowListener(this);
    }
    // mètodes de la interfície WindowListener:
    public void windowActivated( WindowEvent e) {}
    public void windowDeactivated( WindowEvent e) {}
    public void windowIconified( WindowEvent e) {}
    public void windowDeiconified( WindowEvent e) {}
    public void windowOpened( WindowEvent e) {}
    public void windowClosed( WindowEvent e) {}
    // reescriure el mètode windowClosing per sortir del programa
    public void windowClosing( WindowEvent e) {
        System.exit( 0); // sortida normal
    }
} // Final de la classe FinestraTancable
```

La classe implementa  
la interfície  
WindowListener

La classe registra l'objecte  
com a listener

Mètodes de la interfície  
WindowListener

### Comentaris:

1. Aquest escoltador només es registrarà amb aquesta finestra de tipus FinestraTancable i no altres  
2. Hi ha 7 mètodes en la Interfície WindowListener, però aquí només ens interessem per l'event tancament de finestra.

```
// Fitxer Main.java
class Main {
    public static void main( String[] args) {
        FinestraTancable finestra= new FinestraTancable();
        finestra.setVisible(true); // fa el JFrame visible
    }
} // Final de la classe Main
```

## Exemple 3 (b): Separant les classes

La interfície WindowListener és implementada per la classe MeuEscoltadorFinestra, així la seva instància pot respondre als events de la finestra en la que s'hagi registrat.

Continua havent 7 mètodes en la Interfície WindowListener, encara que només un fa alguna cosa.

```
// Fitxer FinestraTancable2.java
import java.awt.event.*;
import javax.swing.*;
class FinestraTancable2 extends JFrame {
    // Constructor:
    public FinestraTancable2 () {
        setTitle("La meva finestra tancable");
        setSize( 300, 200);
        // causa events de window per a ser enviat a l'objecte window listener
        this.addWindowListener(new MeuEscoltadorFinestra ());
    }
} // Final de la classe FinestraTancable2
```

Un objecte MeuEscoltadorFinestra és registrat amb addWindowListener

```
// Fitxer MeuEscoltadorFinestra.java
class MeuEscoltadorFinestra implements WindowListener {
    // Do nothing methods required by interface
    public void windowActivated( WindowEvent e) {}
    public void windowDeactivated( WindowEvent e) {}
    public void windowIconified( WindowEvent e) {}
    public void windowDeiconified( WindowEvent e) {}
    public void windowOpened( WindowEvent e) {}
    public void windowClosed( WindowEvent e) {}
    // reescriure el mètode windowClosing per sortir del programa
    public void windowClosing( WindowEvent e) {
        System.exit( 0); // normal exit
    }
} // Final de la classe
```

```
// Fitxer Main.java
class Main {
    public static void main( String[] args) {
        FinestraTancable2 finestra = new FinestraTancable2();
        finestra.setVisible(true); // fa el JFrame visible
    }
} //Final de la classe Main
```

# Classes Adapter

- Són classes abstractes que implementen una interfície Listener amb mètodes buits (“dummy”), utilització herència.
- Útils per a interfícies Listeners amb més d’un mètode
- Principalment, per raons de conveniència
  - Al tenir una implementació “null” dels mètodes de la interfície, quan heretem, únicament hem d’implementar els mètodes que necessitem. La nostra classe no té mètodes buits – com passa si implementem una interfície i no necessitem alguns dels seus mètodes
- Exemples:
  - MouseAdapter, WindowAdapter

## Exemple 3 (c): Classes Adapter

```
// Fitxer FinestraTancable3.java
import java.awt.event.*;
import javax.swing.*;
class FinestraTancable3 extends JFrame {
    // Constructor
    public FinestraTancable3() {
        setTitle("La meva finestra tancable");
        setSize( 300, 200);
        TancarFinestra tf = new TancarFinestra();
        this.addWindowListener(tf);
    }
} // Finla de la classe FinestraTancable3
```

Un objecte TancarFinestra és registrat amb  
addWindowListener

Definim una classe auxiliar que  
hereta de WindowAdapter

```
// Fitxer TancarFinestra.java
class TancarFinestra extends WindowAdapter {
    public void windowClosing(WindowEvent we){
        System.exit(0); // normal exit
    }
} // Final de la classe TancarFinestra
```

```
// Fitxer Main.java
class Main {
    public static void main( String[] args) {
        FinestraTancable3 finestra = new FinestraTancable3();
        finestra.setVisible(true); // fa el JFrame visible
    }
} // Final de la classe Main
```



# Classes Adapter

- Desavantatge: Java no permet herència múltiple
  - `public class Biblioteca extends Carpeta, WindowsAdapter ...`
- Solució: utilitzar classes internes anònimes o amb nom.

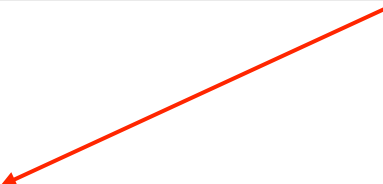
# Classes internes

- *Inner Class*
- En Java una classe pot ser definida en qualsevol lloc
  - ◆ Niuada dins d'altres classes
  - ◆ En la invocació d'un mètode
- Tenen accés als membres i mètodes de totes les classes externes a elles
- Poden tenir noms o ser anònimes
- Poden estendre d'una altra classe o implementar interfícies
- Molt útils per a la manipulació d'events

# Classes internes amb nom

- Es defineixen com les classes normals
- No poden ser **public**

Ha d'estar dins de la classe si  
volem que tingui accés als  
mètodes de la classe externa.



```
public class Aplicacio{
    ....
    class ManejadorBoto implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            faLaAccio();
        }
    } // Final de la classe interna
    private void faLaAccio() { // aquí es fa l'acció del botó
        ...
    }
    public void go() {
        ...
        JButton okButton = new JButton("OK");
        okButton.addActionListener(new ManejadorBoto()); // crea el escoltador
        ....
    }
} // Final de la classe Aplicacio
```

# Classes internes amb nom

Un altre exemple, utilitzant classe adapter en lloc d'una interfície:

```
public class LaMevaClasse extends JPanel {  
    ....  
    class ElMeuAdapter extends MouseAdapter {  
        public void mouseClicked(MouseEvent e) {  
            ...  
        } // Final del mètode mouseClicked  
    } // Final de la classe interna ElMeuAdapter  
    public void go() {  
        ...  
        unObjecte.addMouseListener(new ElMeuAdapter());  
        ...  
    }  
} // Final de LaMevaClasse
```

# Classes internes anònimes

- Definida dins de addXXXListener:  
*(new className( ) { classBody });*  
*(new interfaceName() { classBody });*
- Dins del cos no pot definir constructors.

```
public class Applicacio {  
    públic void go(){  
        JButton okButton = new JButton("OK");  
        okButton.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent event) {  
                faLaAccio();  
            }  
        });  
        ....  
    }  
    private void faLaAccio() {  
        // Aquí és on s'implementa l'acció  
    }  
    ....  
}
```

## Exemple 3 (d): Classes internes anònimes de Java

// Fitxer FinestraTancable4.java

```
import java.awt.event.*;
import javax.swing.*;
class FinestraTancable4 extends JFrame {
    // Constructor
    public FinestraTancable4() {
        setTitle("La meva finestra tancable");
        setSize( 300, 200);
        this.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                finestraTancableaWindowClosing();
            }
        });
    }
    private void finestraTancableaWindowClosing(){
        System.exit(0);
    }
} // Final de la classe FinestraTancable
```

No s'està creant un nou objecte de **WindowAdapter** (entre altres coses perquè la classe és **abstracta**), sinó que s'està estenent la classe **WindowAdapter**, encara que la paraula **extends** no aparegui

```
class Main {
    public static void main( String[] args) {
        FinestraTancable4 finestra = new FinestraTancable4();
        finestra.setVisible(true); // fa el JFrame visible
    }
} //Final de la classe Main
```

## Exemple 3 (e): Classes internes amb nom de Java

```
public class FinestraTancable extends JFrame {  
    public FinestraTancable(){  
        this.setTitle("Exemple");  
        this.setSize(200, 200);  
        this.addWindowListener(new ElMeuAdapter());  
    }  
    class ElMeuAdapter extends WindowAdapter{  
        public void windowClosing(WindowEvent we){  
            finestraTancableWindowClosing();  
        }  
    }  
    private void finestraTancableWindowClosing(){  
        System.exit(0);  
    }  
}  
  
    public class AdapterInnerClass {  
        public static void main(String[] args) {  
            FinestraTancable ft = new FinestraTancable();  
            ft.setVisible(true);  
        }  
    }
```

## Exemple 4

- Com gestionem events `ActionEvent` per a dos botons quan cada botó necessita fer una cosa diferent?

Per pensar



# Exemple 4: Com gestionem events ActionEvent per a dos botons quan cada botó necessita fer una cosa diferent?



# Referències

- **Package java.awt.event**
- <http://docs.oracle.com/javase/7/docs/api/java/awt/event/package-summary.html>
- **Event Listeners:**
- <https://docs.oracle.com/javase/tutorial/uiswing/events/index.html>