GRAU D'ENGINYERIA INFORMÀTICA

PROGRAMACIÓ II

Bloc 3:

Programació Orientada a Events (2)

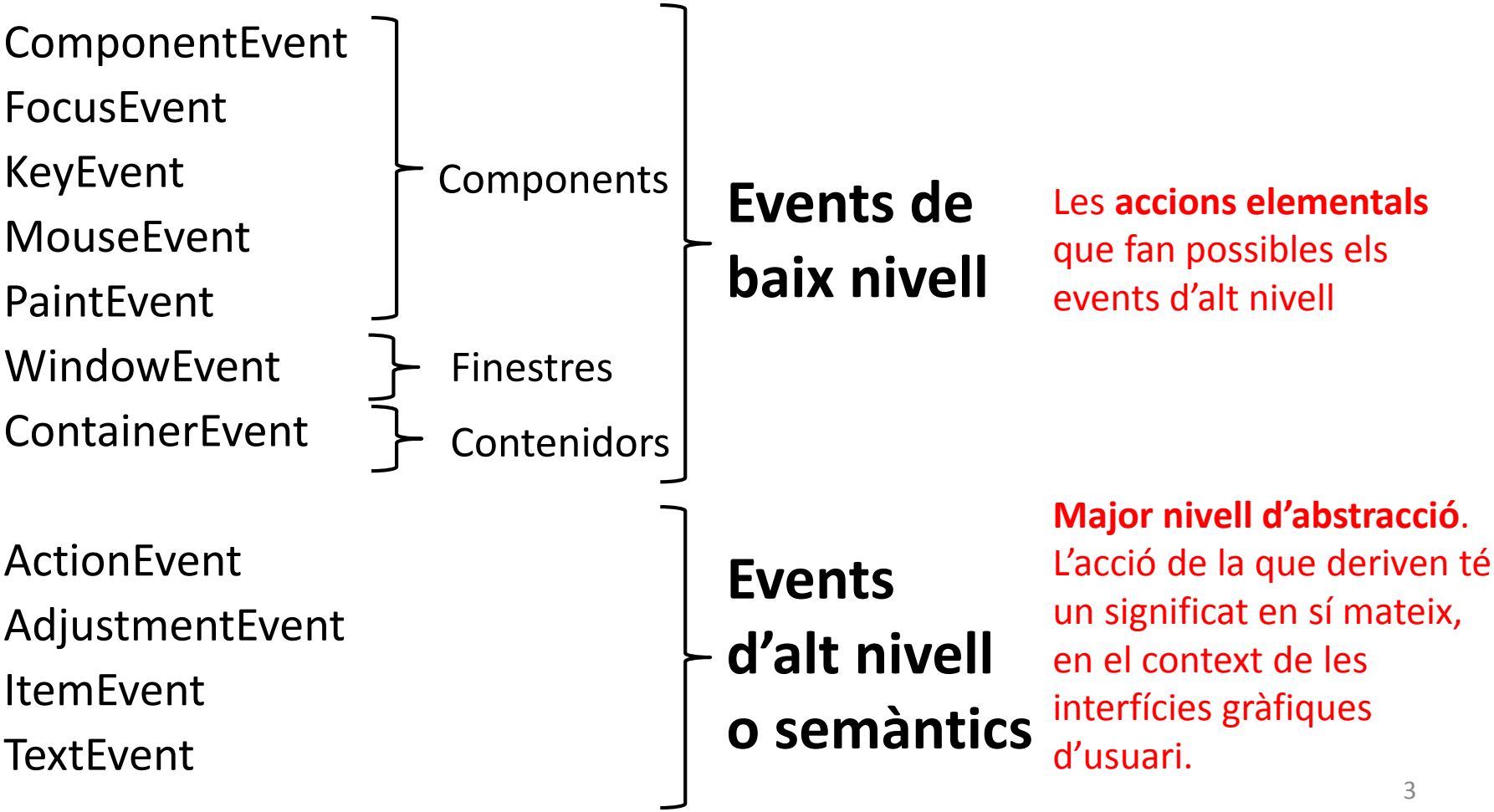
Laura Igual
Departament de Matemàtica Aplicada i Anàlisi
Facultat de Matemàtiques
Universitat de Barcelona

UNIVERSITAT DE BARCELONA

COMPONENTS, EVENTS I LISTENERS

Events AWT

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



Events AWT

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

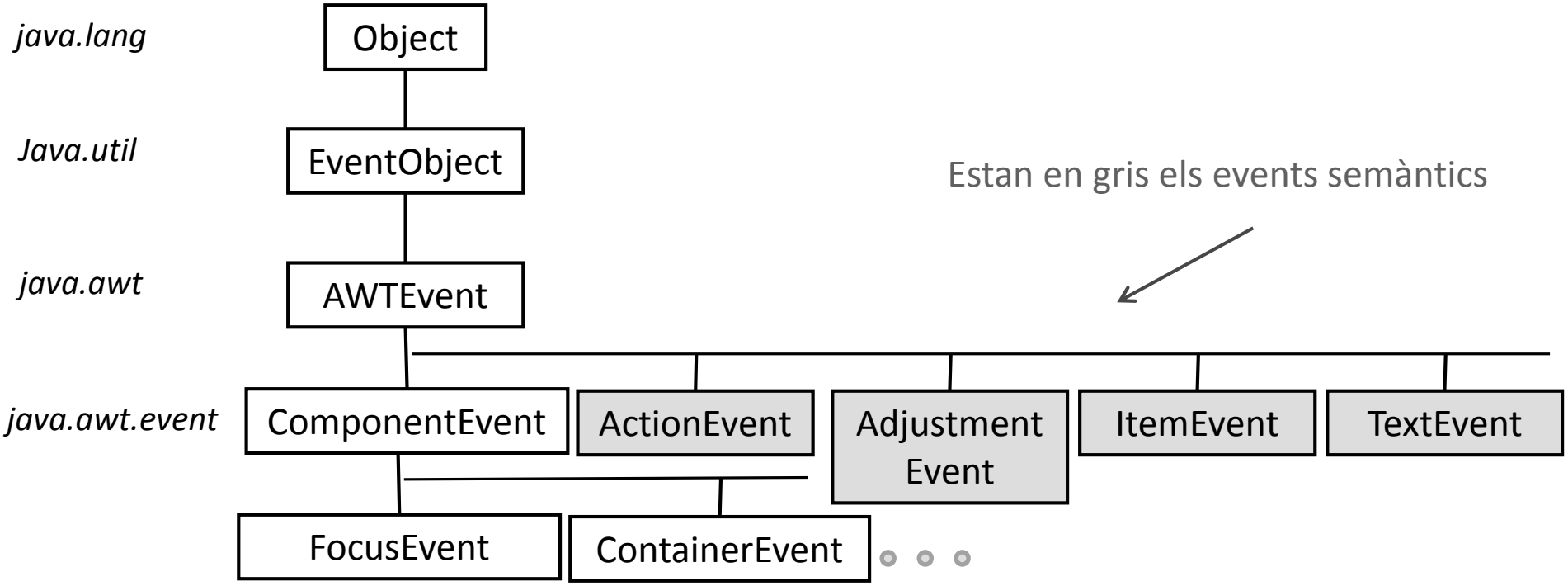
ComponentEvent
FocusEvent
KeyEvent
MouseEvent
PaintEvent
WindowEvent
ContainerEvent

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:



Componentes i events soportats per el AWT de Java

Tots aquests events també es poden produir en les subclasses de Component

Tenen relació amb el context que defineix la component

Component	Eventos generados	Significado
Button	ActionEvent	Clicar en el botón
Checkbox	ItemEvent	Seleccionar o deseleccionar un item
CheckboxMenuItem	ItemEvent	Seleccionar o deseleccionar un item
Choice	ItemEvent	Seleccionar o deseleccionar un item
Component	ComponentEvent	Mover, cambiar tamaño, mostrar u ocultar un componente
	FocusEvent	Obtener o perder el focus
	KeyEvent	Pulsar o soltar una tecla
	MouseEvent	Pulsar o soltar un botón del ratón; entrar o salir de un componente; mover o arrastrar el ratón (tener en cuenta que este evento tiene dos Listener)
Container	ContainerEvent	Añadir o eliminar un componente de un container
List	ActionEvent	Hacer doble click sobre un item de la lista
	ItemEvent	Seleccionar o deseleccionar un item de la lista
MunuItem	ActionEvent	Seleccionar un item de un menú
Scrollbar	AdjustementEvent	Cambiar el valor de la scrollbar
TextComponent	TextEvent	Cambiar el texto
TextField	ActionEvent	Terminar de editar un texto pulsando Intro
Window	WindowEvent	Acciones sobre una ventana: abrir, cerrar, iconizar, restablecer e iniciar el cierre

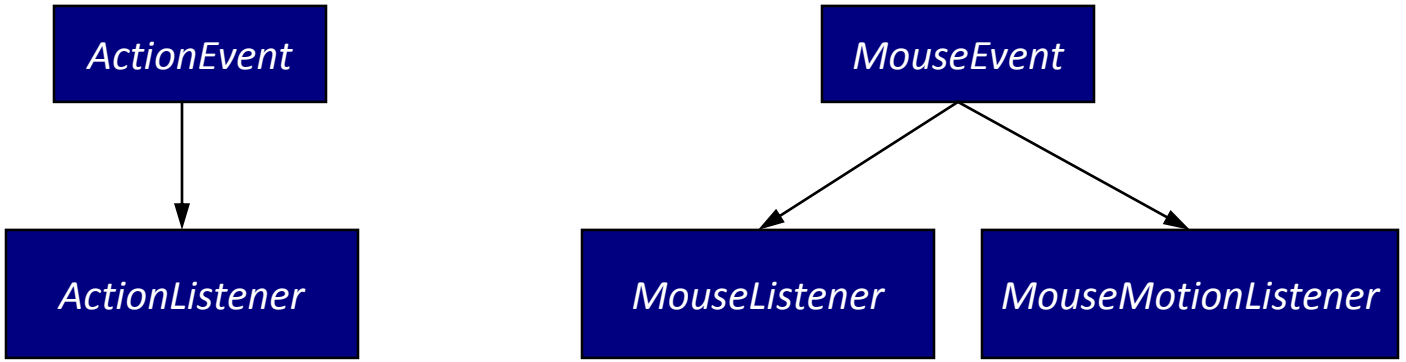
Tabla 5.1. Componentes del AWT y eventos específicos que generan.

Events semàntics

- No són disparats per tots els components
- Exemple 1: **ItemEvent**
  - Disparat per JComboBox
  - No disparat per JButton
- Exemple 2: **ActionEvent**
  - Disparat per JComboBox
  - Disparat per JButton

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 EventListerner {
        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

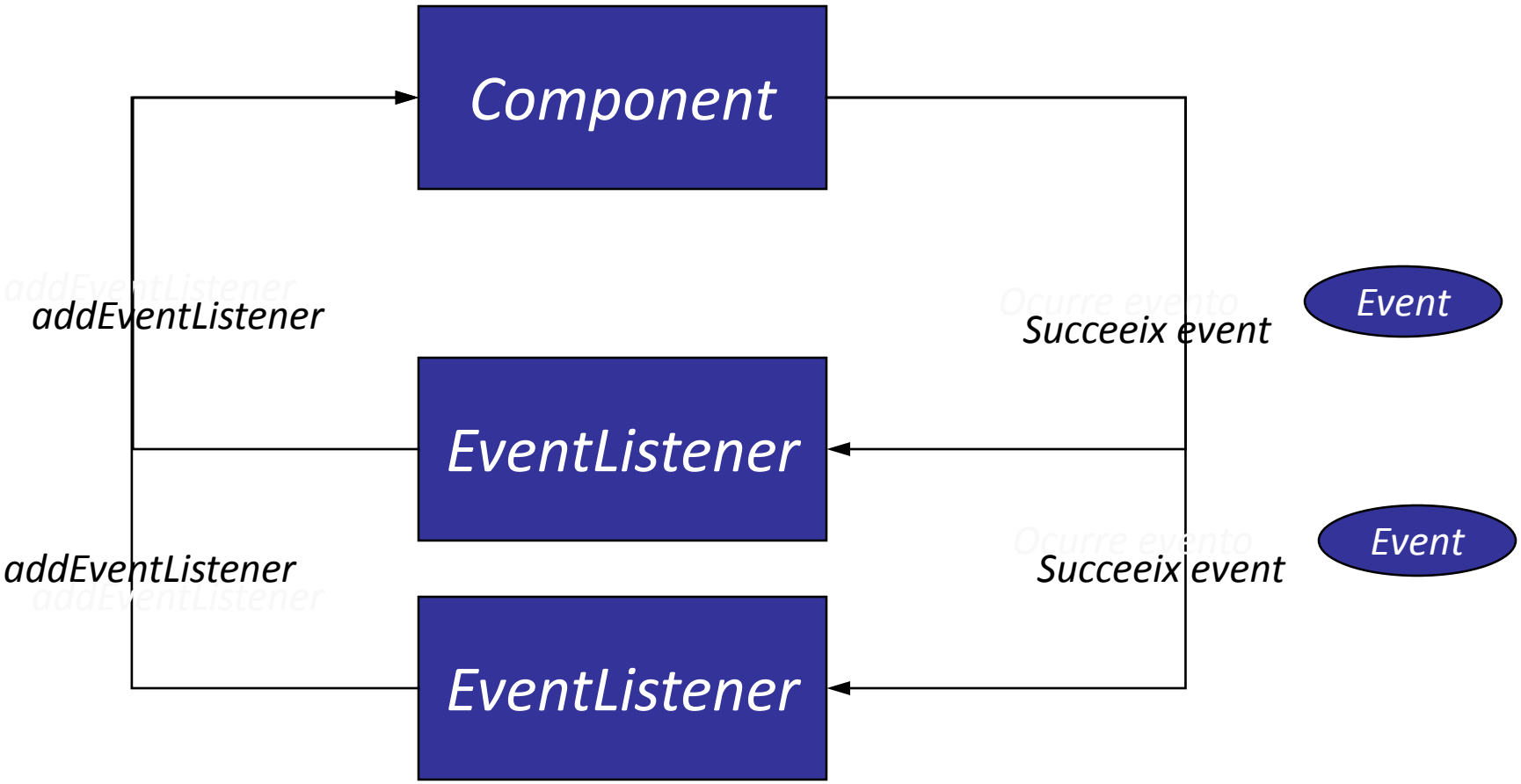
- Un Listener, en primer lloc, ha de registrar-se amb la/les font/s que pugui/n generar els events del seu interès. Ho faran mitjançant un mètode de la forma:

  public void addXXXListener(XXXListener e)

  Per exemple: addActionListener, addItemListener, …

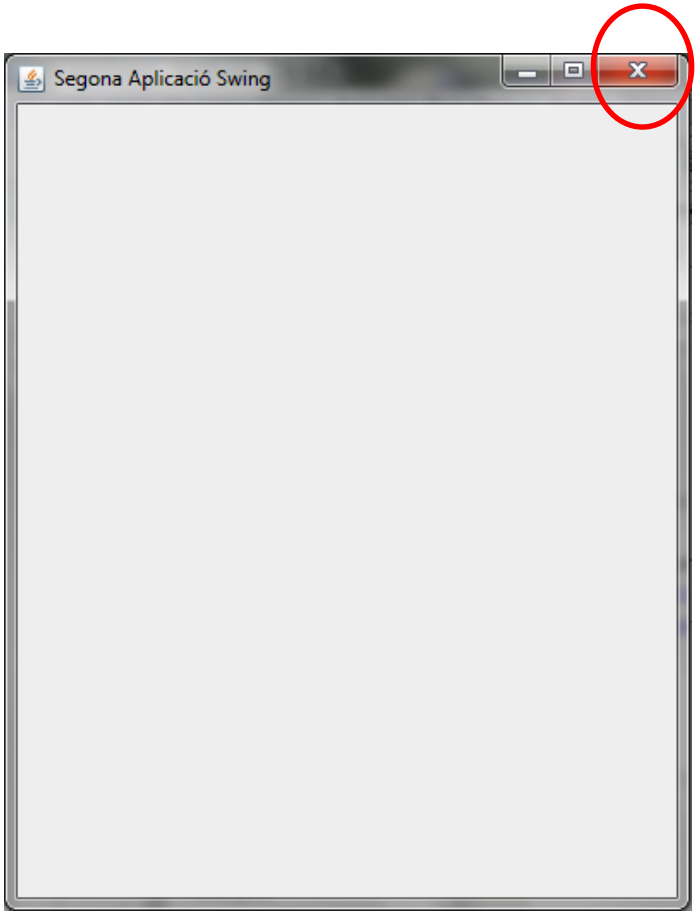
- Per al mateix event en un component, pot haver diversos Listeners registrats
  - Un event pot provocar nombroses respostes
  - Els events són passats a tots els seus Listeners

Múltiples Listeners



CLASSES ADAPTER I CLASSES INTERNES: EXEMPLE D'IMPLEMENTACIÓ D'UNA FINESTRA QUE ES TANCA.

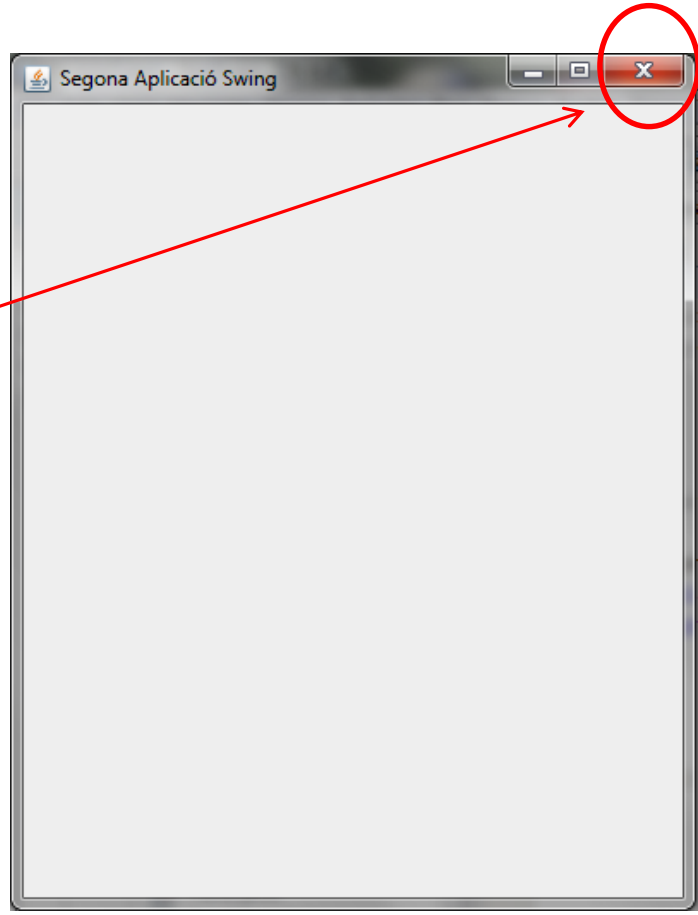
Exemple 3: FINESTRA que es tanca



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

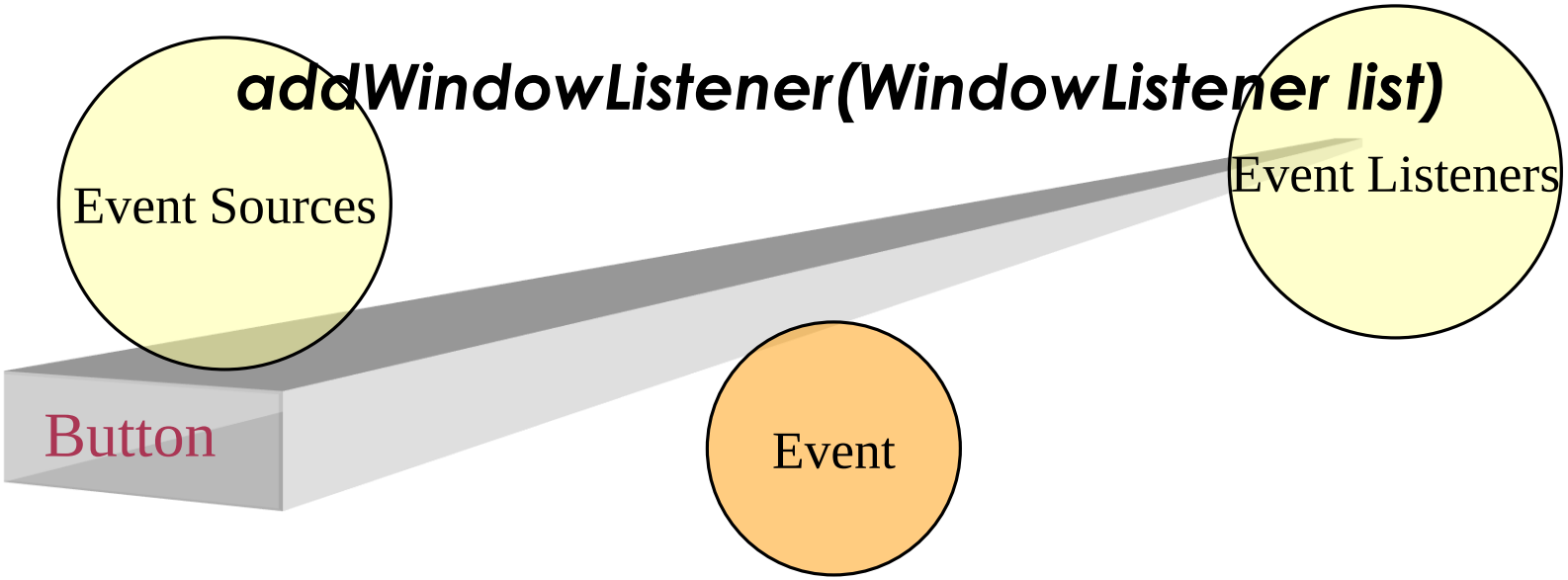


Caldrà relacionar el botó per tancar la finestra amb l'acció de terminar l'aplicació

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 EventListerner {
        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);
}
```

Consultar: http://docs.oracle.com/javase/7/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 () {
                setTitle("La meva finestra tancable");
                setSize( 300, 200);
                // causa  events de window  per a ser enviat al objecte window listener
                addWindowListener(this);
         }
        // mètodes de la interfície WindowsListener
         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) {}
         // reesecriure  el mètode windowClosing  per sortir del programa
         public void windowClosing( WindowEvent e) {
           System.exit( 0); // sortida  normal
         }
} // Final de la classe FinestraTancable
class Main {
  public static void main( String[] args) {
    FinestraTancable finestra= new FinestraTancable();
    finestra.setVisible(true); // fa el JFrame visible
  }
} // Final de la classe Main
```

La classe implementa la interfície WindowsListener

La classe registra l'objecte com a listener

mètodes de la interfície WindowsListener

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.

Exemple 3 (b): Separant les classes

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

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.

// 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 al //objecte
      window listener
      addWindowListener( new MeuEscoltadorFinestra ());
   }
} // Final de la classe FinestraTancable2
```

```
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) {}
   // reesecriure  el mètode windowClosing  per sortir del
programa
  public void windowClosing( WindowEvent e) {
    System.exit( 0); // normal exit
  }
} // Final de la classe
```

Un objecte MeuEscoltadorFinestra és registrat amb addWindowLister

```
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 només per a interfícies Listeners amb més d'un mètode
- Principalment, per raons de conveniència
- 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);
  }
} // fi de la classe FinestraTancable3
```

Un objecte TancarFinestra és registrat amb addWindowLister

Definim una classe auxiliar que hereta de WindowAdapter

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

```
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
- 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 ApplicationFrame {
        ….
        class ButtonHandler implements ActionListener {
                public void actionPerformed(ActionEvent e) {
                        doTheOKThing();                    }
         }
        private void doTheOKThing() { // here's where I handle OK
        }
        ….
        JButton okButton = new JButton("OK");
        okButton.addActionListener(new ButtonHandler());  // create inner class listener
        ….
}
```

nomBotoActionPerformed();

Classes internes amb nom

```
public class MyClass extends JPanel {
        …
         anObject.addMouseListener(new MyAdapter());
      …
      class MyAdapter extends MouseAdapter {
            public void mouseClicked(MouseEvent e) {
                        // ...
            } // end mouseClicked
      } // end inner class
} // end MyClass
```

Classes internes anònimes

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

```
public class ApplicationFrame {
….
        JButton okButton = new JButton(“OK”);
        okButton.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent event) {
                        doTheOKThing();
                }
        } );
        ….
        private void doTheOKThing() {
                // here’s where I handle the OK
        }
….
```

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
```

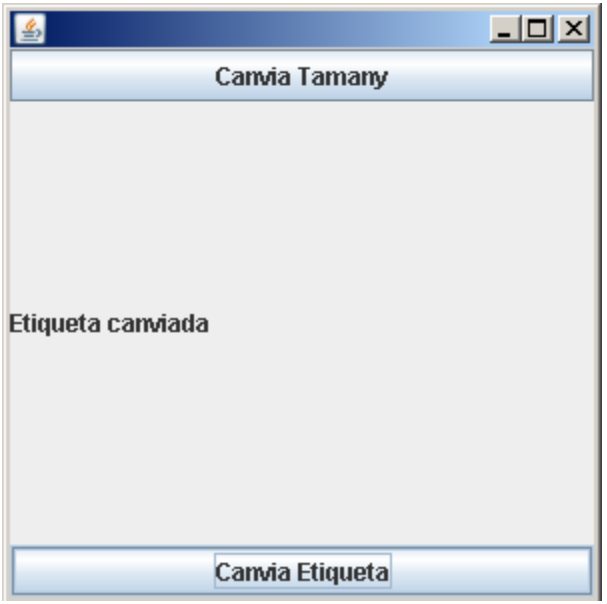
Exemple 3 (e): Classes internes amb nom de Java

Per fer

Exemple 4

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

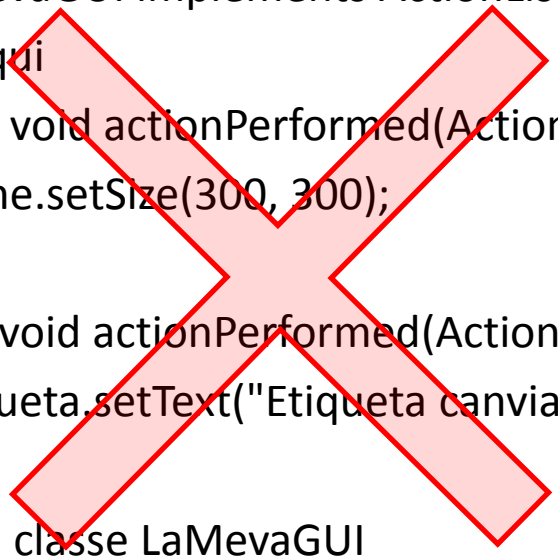
Exemple 4: Com gestionem events ActionEvent per a dos botons quan cada botó necessita fer una cosa diferent?



Exemple 4: Opició A

- Implementar dos mètodes actionPerformed()

```
class LaMevaGUI implements ActionListener {
  // codi aqui
       public void actionPerformed(ActionEvent ev) {
          frame.setSize(300, 300);
       }
      public void actionPerformed(ActionEvent ev) {
          etiqueta.setText("Etiqueta canviada");
       }
} // fi de la classe LaMevaGUI
```



Però això és impossible

Exemple 4: Opició B

- Registrar el mateix listener amb dos botons

```
class LaMevaGUI implements ActionListener {
  // declarem variables aqui
      public void go() {
            //construir gui
            boto1 = new JButton();
            boto2 = new JButton();
            boto1.addActionListener(this);
            boto2.addActionListener(this);
            // més codi aquí
      }
      public void actionPerformed(ActionEvent ev) {
            if(ev.getSource() == boto1 ){
                  frame.setSize(300, 300);
            }else{
                  etiqueta.setText("Etiqueta canviada");
            }
      }
} // fi de la classe LaMevaGUI
```

Registre'm el mateix listener

Mirem quin event és

Opció correcta, però pot arribar a ser complicada de gestionar

Exemple 4: Mes Opicions?

- Exercici.

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