

GRAU D'ENGINYERIA INFORMÀTICA

PROGRAMACIÓ II

Bloc 3:

Programació Orientada a Events (1)

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 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

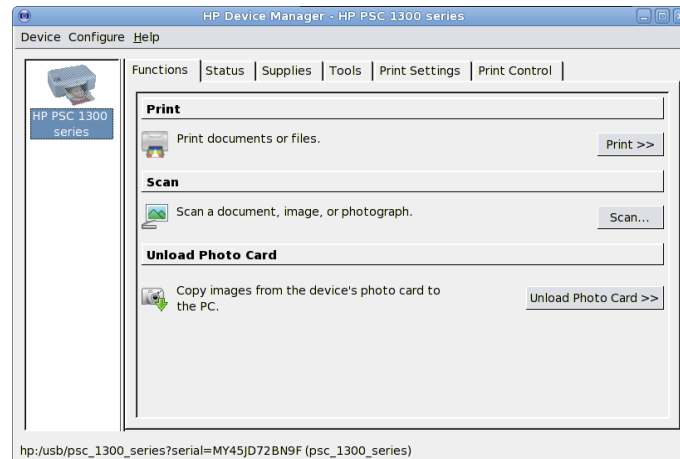
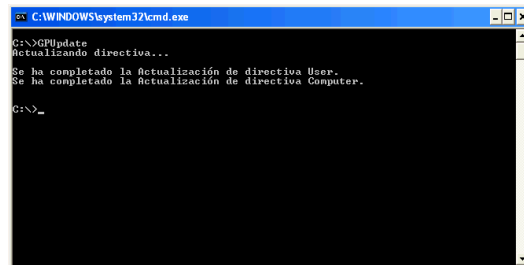
MECANISMES D'INTERACCIÓ

Introducció

- Programació orientada a events (POE)



- Forma de interacció:
 - Línia de comandos vs. Interfícies gràfiques d'usuari



Mecanismes d'interacció

- 1) Estil tradicional d'interacció amb els usuaris
- 2) Programació Orientada a Events

Mecanismes d'interacció

1) Estil tradicional d'interacció amb els usuaris:

- Un programa que necessita una entrada provinent de l'usuari l'obtindrà mitjançant l'execució repetida d'escenaris de la forma:

- 1 ...Efectuar algun càlcul...
- 2 Imprimir ("Si us plau, escriu el valor del paràmetre x")
- 3 Llegir entrada
- 4 x = valor llegit
- 5 ... Seguir endavant amb el càlcul fins que necessiti un altre valor de l'usuari ...

- Segueixen un **flux seqüencial** en el que es tenen cicles: entrada->processament->sortida

Mecanismes d'interacció

2) Programació Orientada a Events

- Els papers s'inverteixen, **les operacions ...**
 - **No es produeixen** perquè el software ha arribat a una determinada fase de la seva execució,
 - **Es produeixen** perquè un determinat **event** ha donat lloc a l'execució d'un determinat component de software.
- L'entrada determina l'execució del software i no al revés.

Concepte d'Event

- Missatge de software que indica que alguna cosa ha succeït:
 - Accions de l'usuari sobre una GUI,
 - Temporitzacions,
 - Canvi d'estat,
 - ...
- Exemples:
 - Polsar una tecla
 - Fer un click d'un ratolí,
 - Mantenir polsat el botó del ratolí
 - Soltar el botó del ratolí
 - Acaba de sonar una cançó
 - Passa un minut,...

Concepte d'Event


- POO ajuda a desenvolupar l'esquema de programació orientada a events.
- Hi ha un catàleg d'events.
- Un **objecte** event representarà una acció de l'usuari, per exemple.
- També es poden definir events personalitzats que un component software pot enviar explícitament mitjançant una crida a procediment.

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

POE per programació d'Interfícies Gràfiques d'Usuari

- POE s'utilitza en el context de programació **d'Interfícies Gràfiques d'Usuari** (GUI: Graphic User Interface)
- Quan programem una GUI hem de tenir en compte la **varietat de possibles interaccions** amb l'usuari.
 - En lloc d'un únic flux d'entrada de dades per consola, les GUIs **permeten moltes més accions de l'usuari**.
- Per exemple:
 - Pressionar botons gràfics,
 - Escriure text en un camp de text,
 - Moure elements gràfics.
- Elements:
 - Finestres,
 - Menús,
 - Botons,
 - Panells,

Gestionen entrada de l'usuari
Proporcionen un cas especial de context

Creació d'una Interfície Gràfica d'Usuari

Per construir una GUI fa falta:

1. **Un contenidor**, que és la finestra o part de la finestra on es situaran els components (botons, barres de desplaçament, menús, etc.) i a on es visualitzarà el que desitgem.
2. **Els components**: menús, botons de comandament, barres de desplaçament, caixes i àrees de text, botons de opció i selecció, etc.
3. **El model d'events**. L'usuari controla l'aplicació actuant sobre els components, amb el ratolí o amb el teclat – o altres entrades.
Cada vegada que l'usuari realitza una determinada acció, es produeix un event, que el sistema operatiu transmet al paquet de gestió.

GUI en Java

- Java inclou, com a part de la seva biblioteca de classes estàndard, un conjunt de **components** per a crear interfícies gràfiques d'usuari
- Aquests elements s'agrupen en dos paquets:
 - **AWT** (Abstract Window Toolkit) -> Web
 - **SWING** (AWT millorat)

GUI en Java

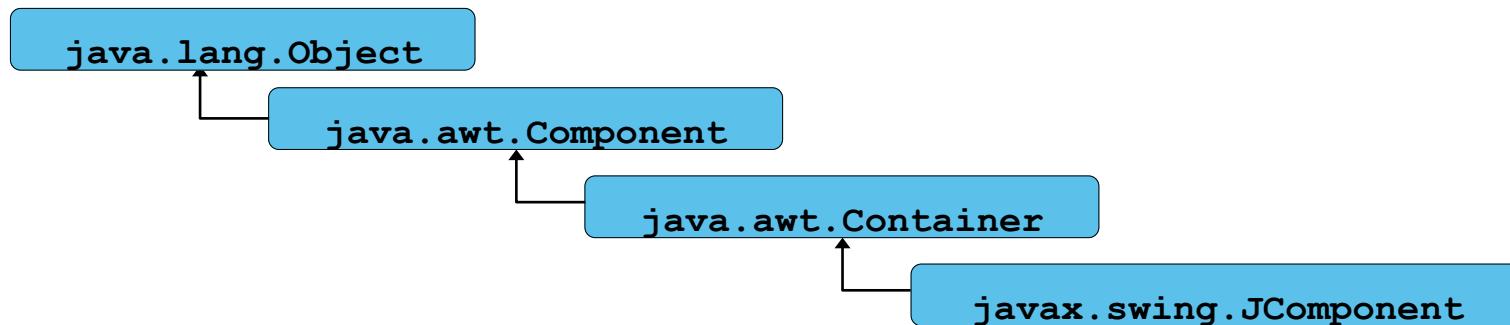
- `java.awt`
 - Els components AWT depenen de les facilitats gràfiques oferides per cada sistema operatiu: els programes escrits amb AWT tindran un “look and feel” diferent en Windows i en UNIX
- `java.swing`
 - SWING és 100% Java i, per tant, completament independent de la plataforma.
 - Les components gràfiques es pinten en temps d'execució (per aquest fet les aplicacions SWING solen ser un poc més lentes que les AWT).
- En la pràctica les aplicacions Java amb GUIs solen barrejar AWT i SWING.

GUI en Java

- El AWT crea un objecte d'una determinada classe d'event, derivada de AWTEvent.
- Aquest event es tramés a un determinat mètode per a que el gestioni.
- En Java el component o objecte que rep l'event ha de “registrar” o indicar prèviament quin objecte es farà càrrec de gestionar aquell event → **Model de Delegació d'Events.**

GUI en Java

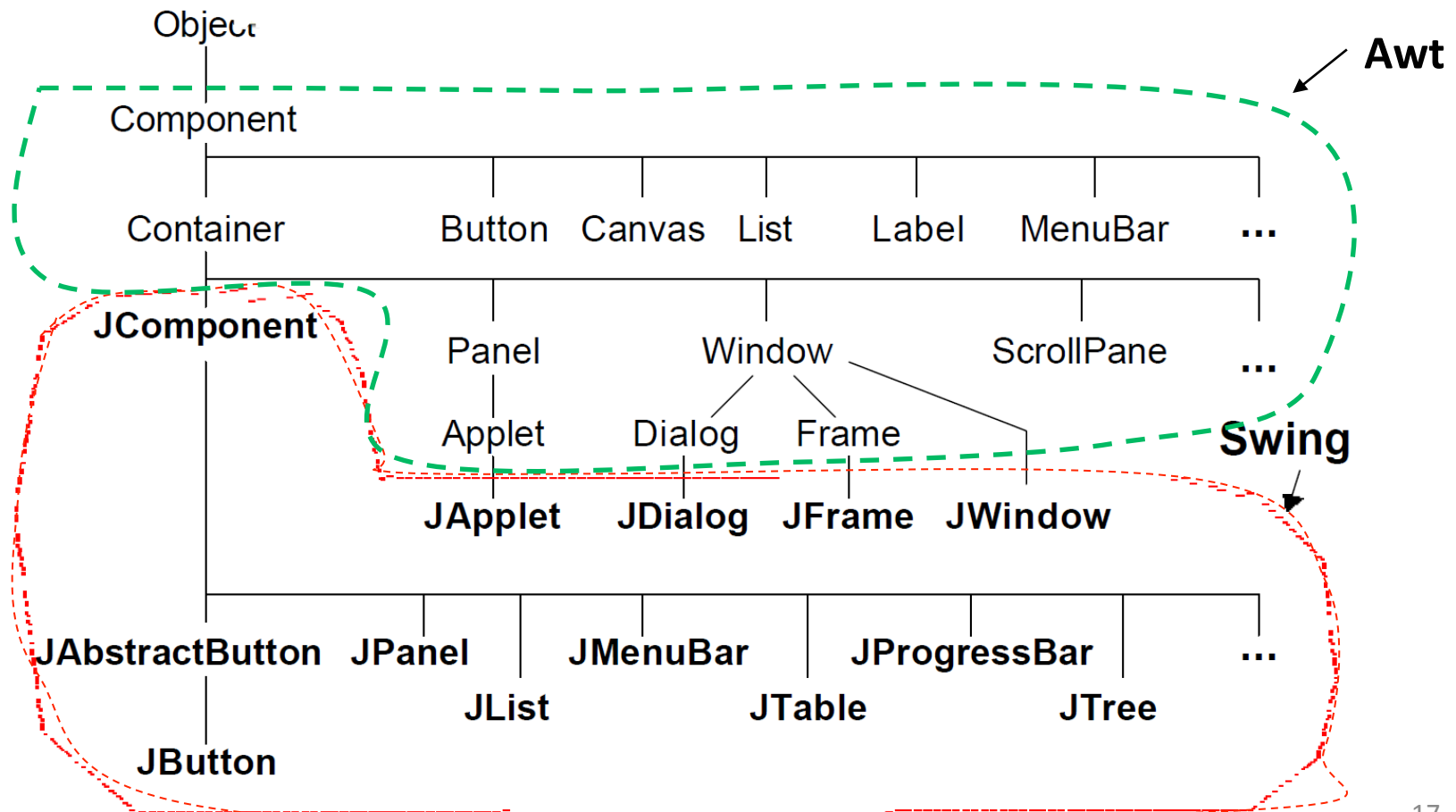
- Jerarquia d'herències dels components de Swing:



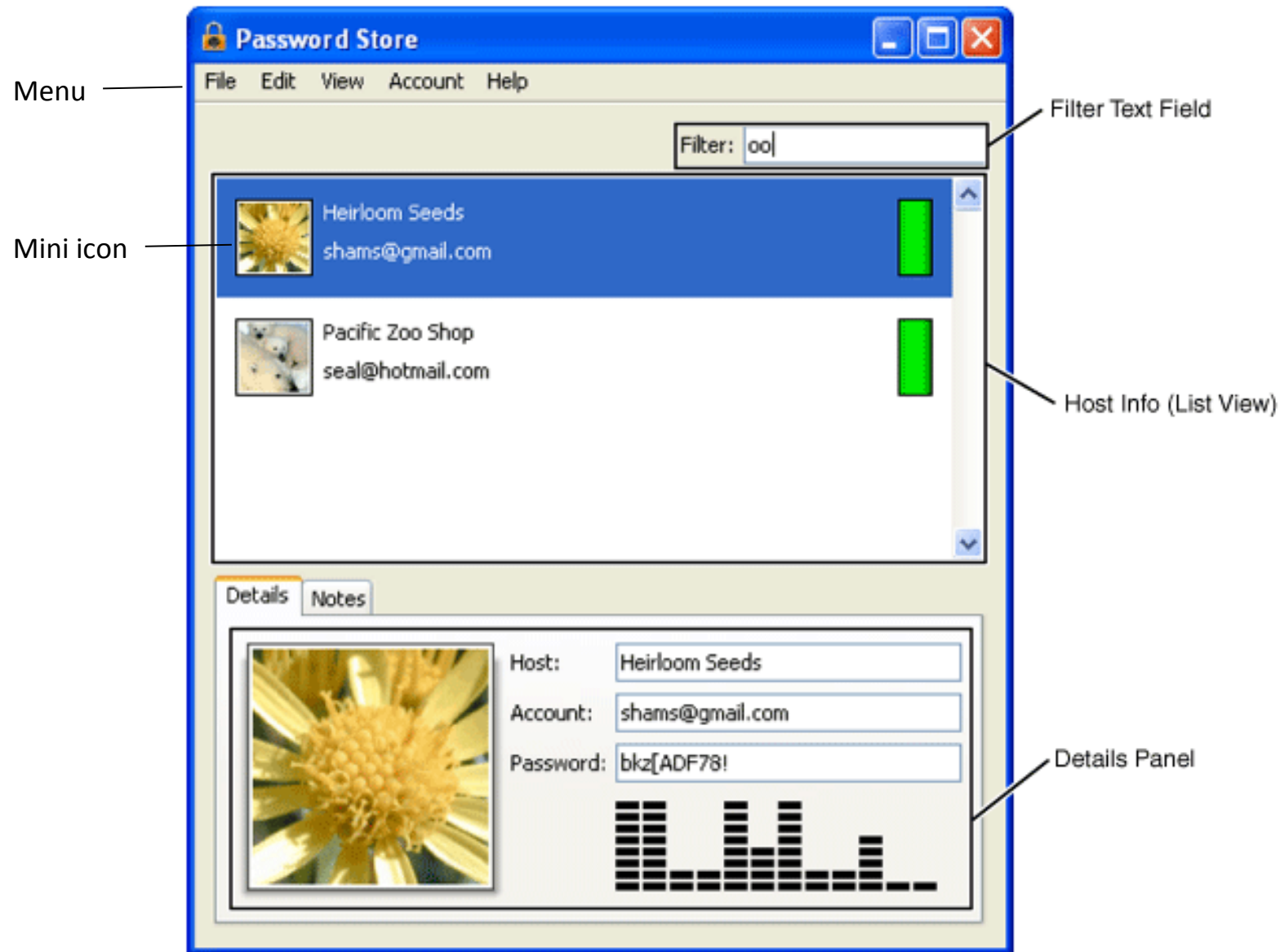
- **Component** defineix mètodes que poden ser usats en les seves subclasses
 - Per exemple: **paint** and **repaint**
- **Container** – col·lecció de components relacionades
 - Mètode **add** per afegir components a la finestra
- **JComponent** - superclasse de la major part dels components de Swing
 - Moltes de les funcionalitats dels components hereten d'aquestes classes

Catàleg de components

Relació jeràrquica entre components:



Example



Més sobre components

- Cada element gràfic de GUI és un component
- Cada component és **una instància d'una classe**
- Una component es crea com qualsevol altre objecte en Java

Classe Component

Métodos de Component	Función que realizan
boolean isVisible(), void setVisible(boolean)	Permiten chequear o establecer la visibilidad de un componente
boolean isShowing()	Permiten saber si un componente se está viendo. Para ello tanto el componente debe ser visible, y su container debe estar mostrándose
boolean isEnabled(), void setEnabled(boolean)	Permiten saber si un componente está activado y activarlo o desactivarlo
Point getLocation(), Point getLocationOnScreen()	Permiten obtener la posición de la esquina superior izquierda de un componente respecto al componente-padre o a la pantalla
void setLocation(Point), void setLocation(int x, int y)	Desplazan un componente a la posición especificada respecto al container o componente-padre
Dimension getSize(), void setSize(int w, int h), void setSize(Dimension d)	Permiten obtener o establecer el tamaño de un componente
Rectangle getBounds(), void setBounds(Rectangle), void setBounds(int x, int y, int width, int height)	Obtienen o establecen la posición y el tamaño de un componente
invalidate(), validate(), doLayout()	invalidate() marca un componente y sus contenedores para indicar que se necesita volver a aplicar el Layout Manager. validate() se asegura que el Layout Manager está bien aplicado. doLayout() hace que se aplique el Layout Manager
paint(Graphics), repaint() y update(Graphics)	Métodos gráficos para dibujar en la pantalla
setBackground(Color), setForeground(Color)	Métodos para establecer los colores por defecto

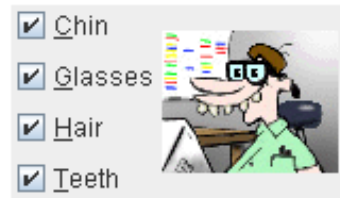
Tabla 5.4. Métodos de la clase Component.

Controls bàsics



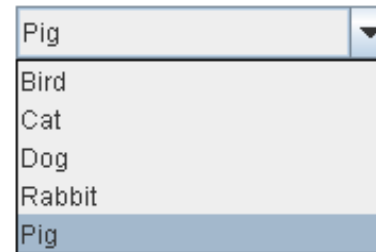
[JButton](#)

Botons



[JCheckBox](#)

Botons



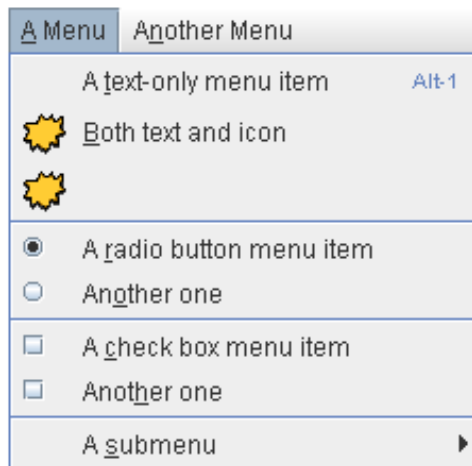
[JComboBox](#)

Caixes combo



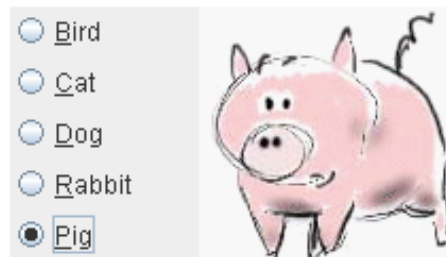
[JList](#)

Llistes



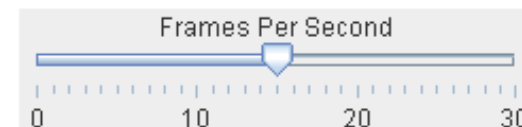
Menús

[JMenu](#)



[JRadioButton](#)

Botons



[JSlider](#)

Controls lliscants



[JSpinner](#)

Controls numèrics



[JTextField](#)

Camps de text (amb/
sense format)



[JPasswordField](#)

Informació sobre Components

- <https://docs.oracle.com/javase/tutorial/uiswing/components/index.html>

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 1: FINESTRA

- Seguim els passos bàsics:
 1. Importar paquets javax.swing.XXX
 2. Disposar un contenidor:
 - JFrame
 3. Agregar components al contenidor
 4. Mostrar el contenidor

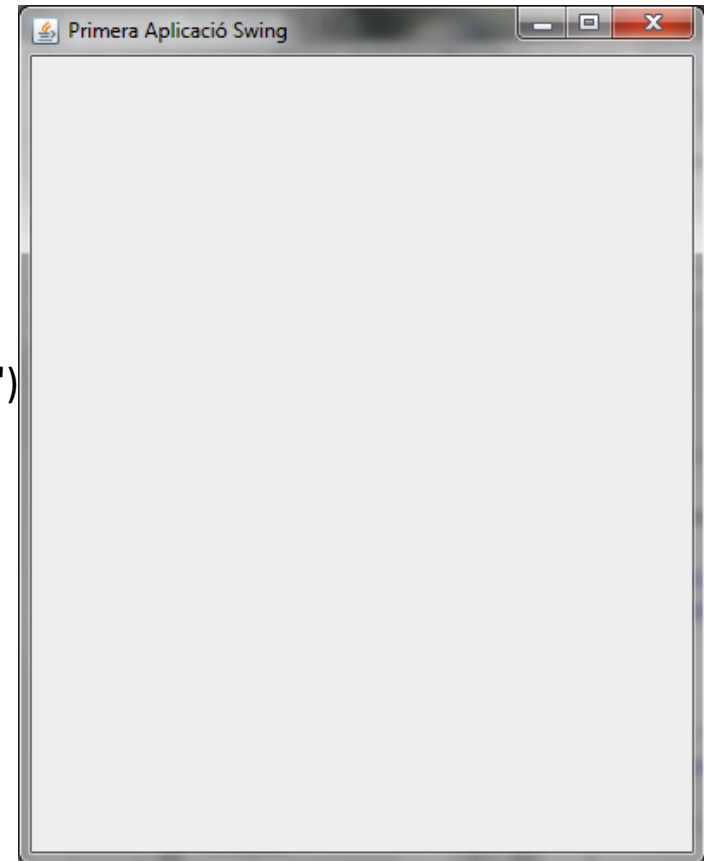
Exemple 1: Dos mètodes de creació de finestres

- La classe JFrame implementa un objecte finestra
- Per a crear una finestra, hi ha dues maneres principals de fer-ho:
 1. Crear un objecte de la classe JFrame
 2. Heretar de la classe JFrame

Exemple 1: Dos mètodes de creació de finestres

- Primera manera:
 - creant un objecte de tipus JFrame: JFrameWindow

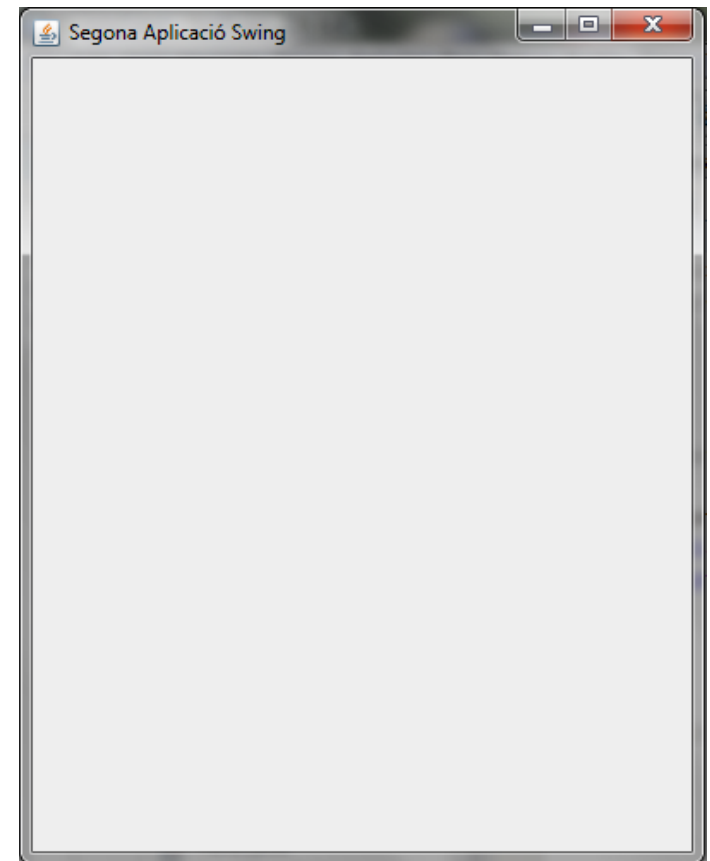
```
import javax.swing.*;  
public class Finestra{  
    public static void main(String []args) {  
        JFrame JFrameWindow = new JFrame();  
        JFrameWindow.setSize(400,500);  
        JFrameWindow.setTitle("Primera Aplicació Swing")  
        JFrameWindow.setVisible(true);  
    }  
}
```



Exemple 1: Dos mètodes de creació de finestres

- Segona manera:
 - estenent la classe JFrame

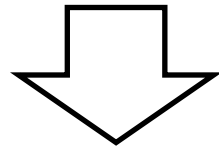
```
import javax.swing.*;  
public class Finestra extends JFrame {  
    public Finestra() {  
        this.setSize(400,500);  
        this.setTitle(" Segona Aplicació Swing");  
        this.setVisible(true);  
    }  
    public static void main(String []args) {  
        Finestra finestra = new Finestra();  
    }  
}
```



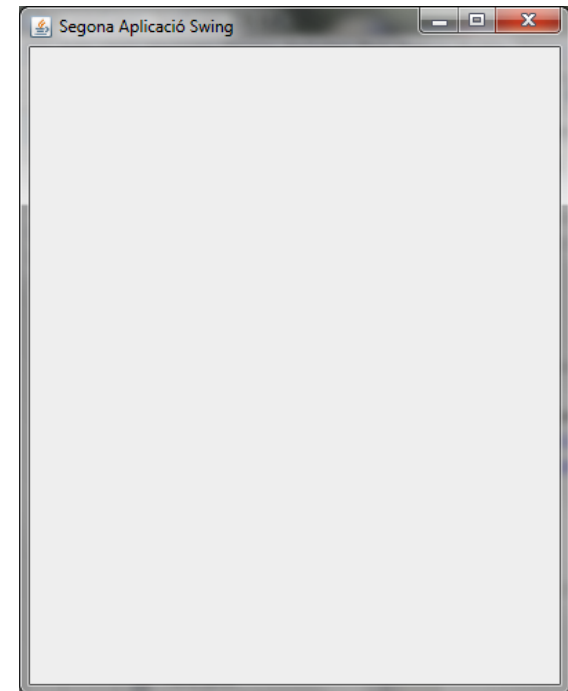
Exemple 1: Dos mètodes de creació de finestres

Fins aquí hem creat una aplicació senzilla de dues maneres.

Però, en cap dels dos casos l'aplicació no fa res



Afegim un botó que tingui una funcionalitat senzilla. Es a dir, que faci alguna cosa quan el premem. Caldrà capturar els events que es llancen



Exemple 2: FINESTRA amb botó

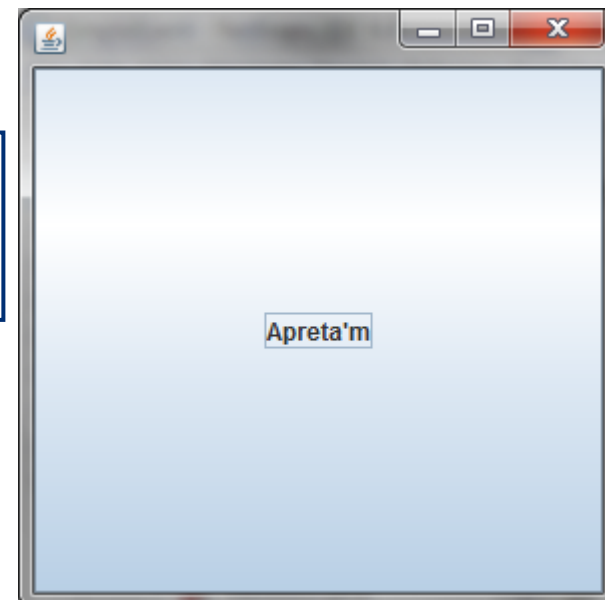
```
public static void main(String []args) {  
    JFrame frame = new JFrame();  
    JButton boto = new JButton ("Apreta'm");  
    frame.getContentPane().add(boto);  
    frame.setSize(300,300);  
    frame.setVisible(true);  
}
```

No afegim un botó al **frame** directament → Pensar en el frame com el marc de la finestra.
Afegim coses al **pane** (cristall) de la finestra

Definim el tamany

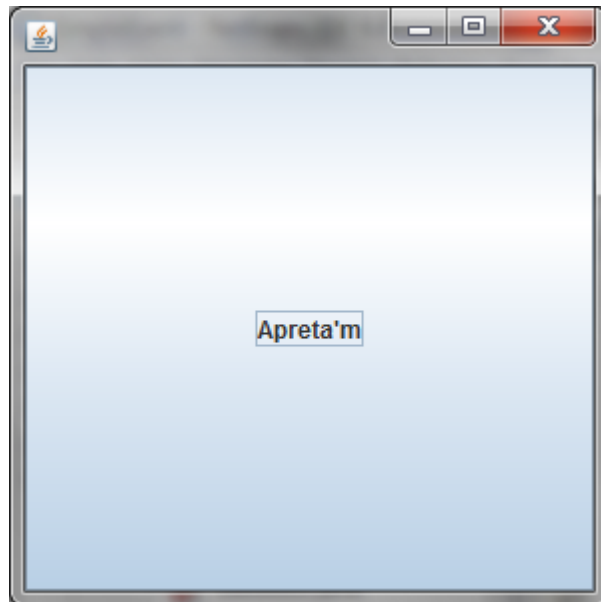
El fem visible

En realitat, el que hem de fer és el següent:
frame.getContentPane().add(BorderLayout.CENTER, boto);
Ho veurem més endavant amb detall.



Una vegada tenim el botó.

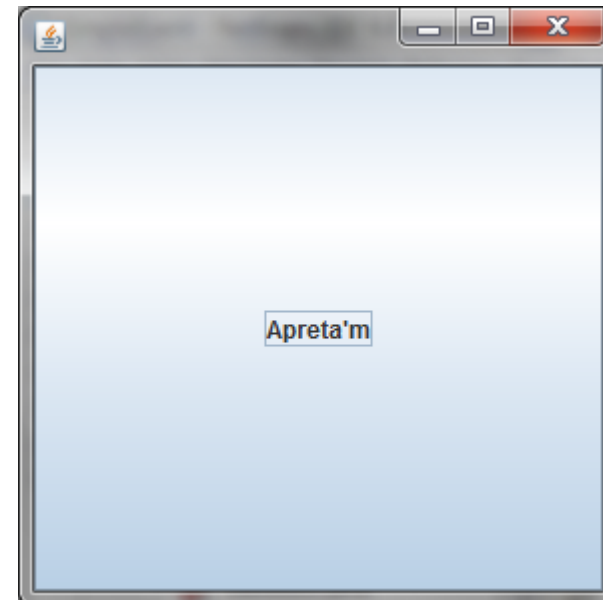
Veurem:



- Com controlarem la seva mida?
- Com controlarem el “look and feel”?
- Què passa quan el premem?
- Com podem fer que passin coses quan el premem?

Què passa quan premem el botó?

- Alguna cosa passa:
l'aspecte canvia



Com podem fer que passin més coses quan el premem?

Com podem fer que passin més coses quan el premem?

- Necessitem:
 1. Un mètode que es cridi quan el botó es prem.
 2. Una forma de saber quan s'ha d'invocar aquest mètode, es a dir, una forma de saber quan es prem el botó.

→ Estem interessats en:

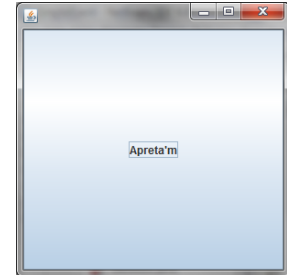
L'event: l'usuari prem el botó

Java UI – Manejar events

- En Java els events són representats per objectes
- Exemples:
 - Fer clic en un botó
 - Arrastrar el ratolí
 - Pulsar Enter
- Els components AWT y Swing generen, “disparen” (*fire*) events
- `java.awt.AWTEvent`

Exemple 2: afegim funcionalitat

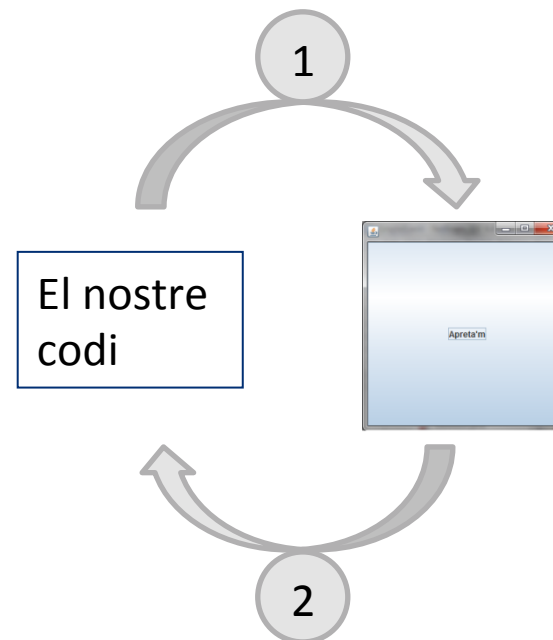
Volem que el text del botó canviï de “Apreta’m” a “He estat apretat”.



- Necessitem:
 - Un mètode per canviar el text del botó:

```
public void changelt() {  
    boto.setText("He estat apretat");  
}
```

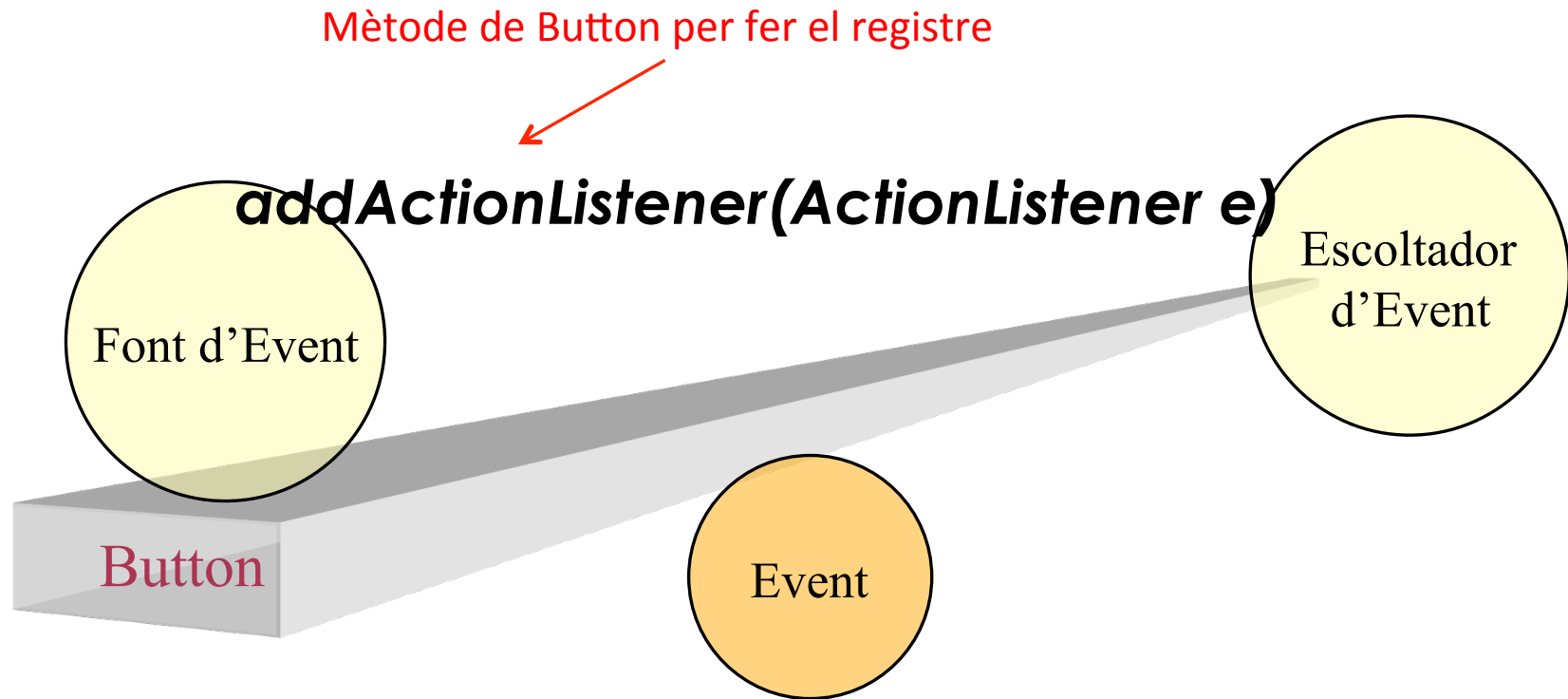
1. El botó ha de saber que ens interessa quan el premen
2. El botó ha de poder tornar a cridar-nos quan el premen



Model de Delegació d'Events

- Cada component pot generar events.
- A cada component es poden registrar **escoltadors (listeners) d'events** (dels tipus d'events que ells poden generar).
- Quan el component generi un event, invoca a tots els seus manejadors d'events.

Model de Delegació d'Events



Model de Delegació d'Events

1. Registrar el Listener

```
eventSourceObject.addActionListener(ActionListener e)
```



2. Definir els mètodes de la interfície: implementar el mètode per que faci el que volem.

Connectar un Listener amb una font d'events

- Definir una classe que implementi la interfície Listener (o que hereti d'una classe que la implementi)

```
public class GUISimple implements ActionListener {...
```

- Afegir la implementació de la interfície dins de la classe GUISimple.
- En el cas de ActionListener, només té un mètode **actionPerformed** a implementar.

```
...  
public void actionPerformed(ActionEvent e) {  
    // aquí és on implemento el que s'ha de fer quan l'acció (event) succeeix  
...}
```

- Registrar el Listener amb la font

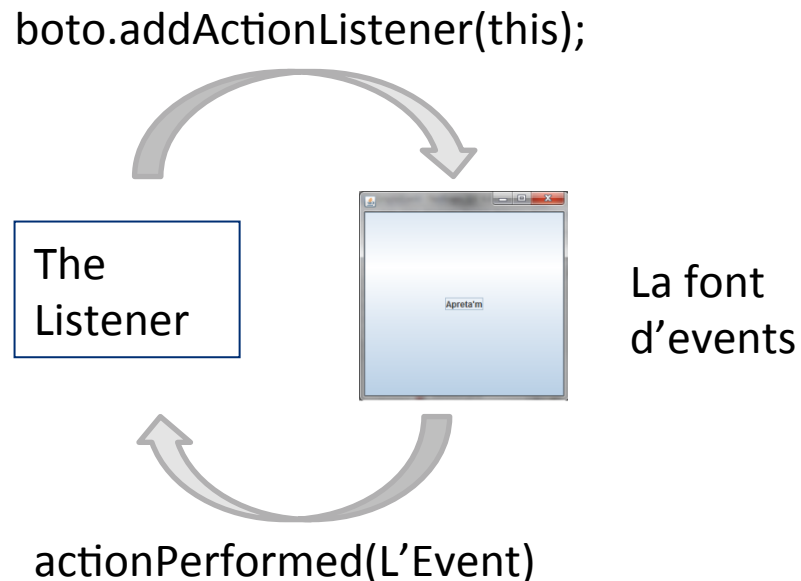
```
...  
JButton okButton = new JButton("OK");  
okButton.addActionListener(this);  
...
```

Exemple 2: Continuació

Volíem que el text del botó canviï de “Apreta’m” a “He estat apretat”.

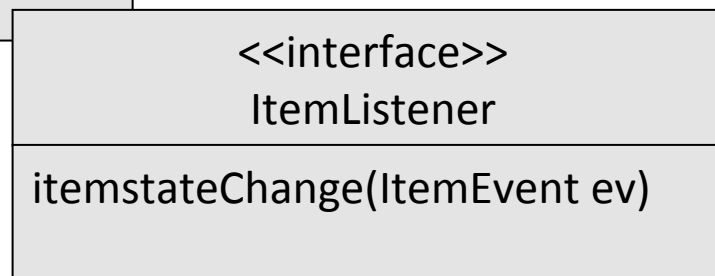
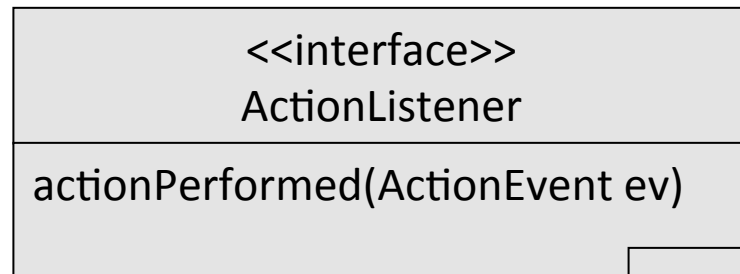
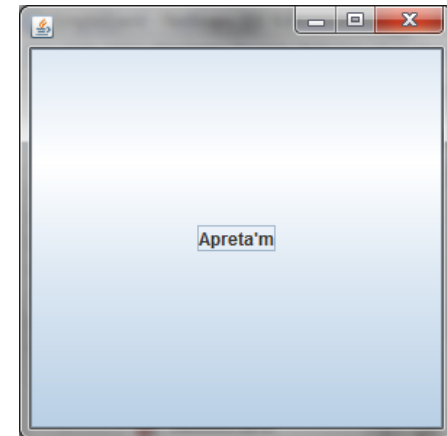
- Implementem un mètode per canviar el text del botó:

```
void changelt() {  
    boto.setText(“He estat apretat”);}
```
- Registrem el botó a l’objecte que tractarà els seus events.



Comunicació entre el Listener i la font

- Quan implementem una interfície Listener donem al botó una forma de tornar a cridar-nos
- La interfície és on el mètode de crida està **declarat**, però no implementat.




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

Dins estan les classes:
ActionListener i ActionEvent

```
public class GUISimple implements ActionListener {
    JButton boto;
```

La classe GUISimple
implementa la interfície
ActionListener

```
    public static void main (String[] args){
        GUISimple gui = new GUISimple ();
        gui.go();
    }
```

```
    public void go(){
        JFrame finestra = new JFrame();
        boto = new JButton("Apreta'm");
```

```
        boto.addActionListener(this);
```

Aquí, afegim aquest objecte
a la llista de listeners.

```
        finestra.getContentPane().add(boto);
```

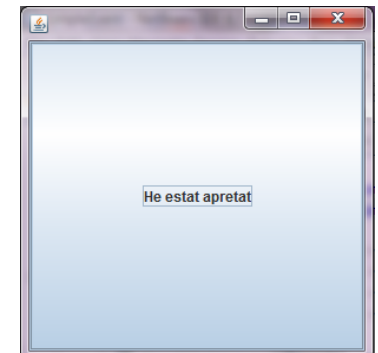
```
        finestra.setSize(300,300);
        finestra.setVisible(true);
    }
```

```
    public void actionPerformed(ActionEvent event) {
        boto.setText("He estat apretat");
    }
```

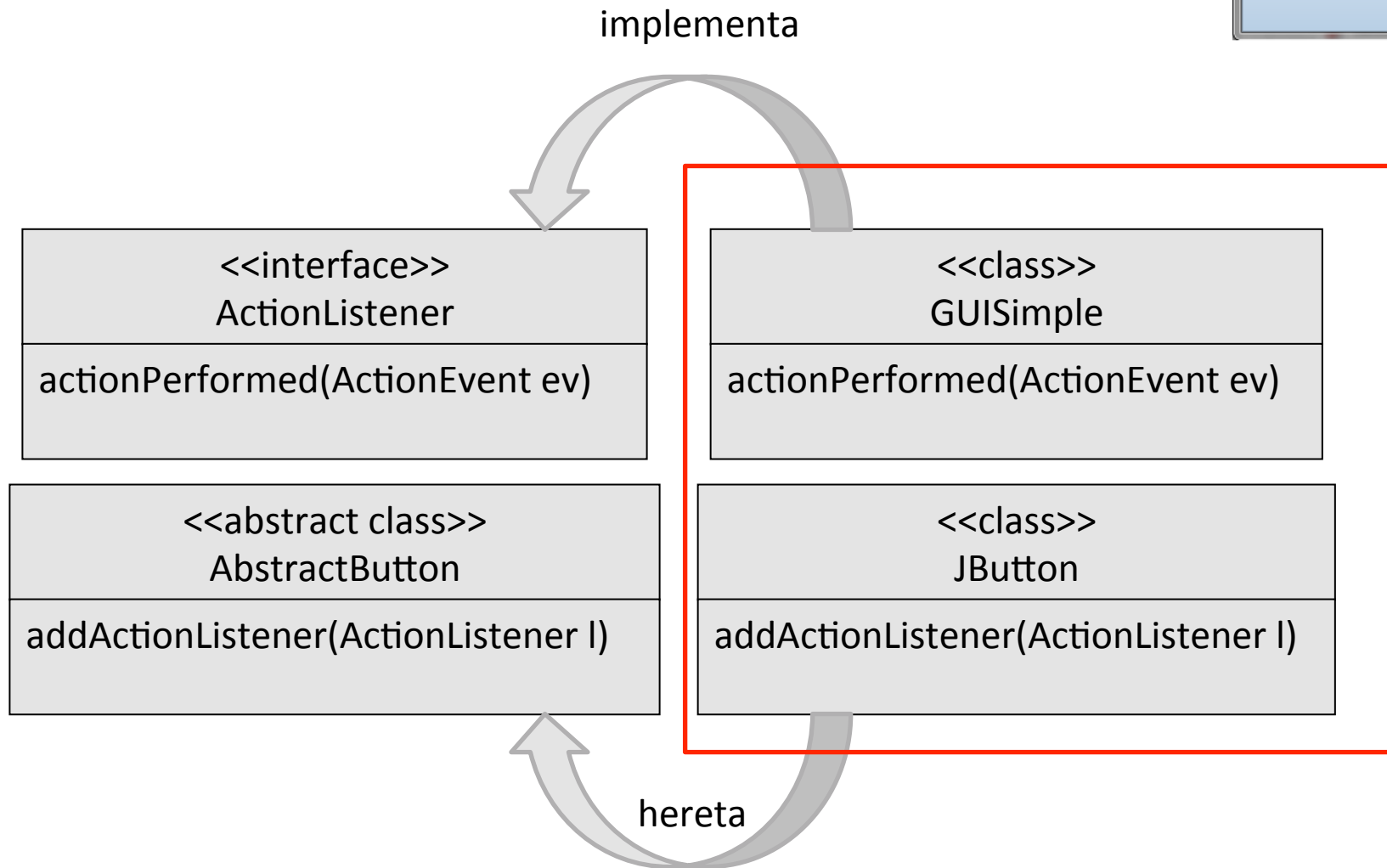
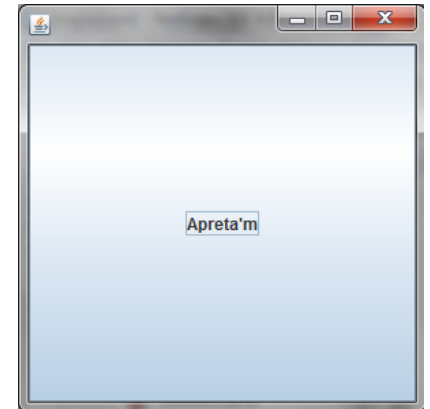
Aquest mètode
s'invoca quan
una acció
succeeix.

Aquí és on faig
el que s'ha de
fer quan l'acció
succeeix

Codi de l'exemple 2



Comentaris



Referències

- Bert Bates, Kathy Sierra. **Head First Java**. O'Reilly Media, 2005.
- **Oracle Tutorials:**
- <https://docs.oracle.com/javase/tutorial/uiswing/components/index.html>
- <http://docs.oracle.com/javase/tutorial/uiswing/>