



# MENÚS, BARRAS DE HERRAMIENTAS Y DIÁLOGOS EN JAVAFX

---

Interfaces Persona Computador

Depto. Sistemas Informáticos y Computación

UPV

# Índice

- Menús
- Barras de herramientas
- Diálogos en JavaFX 8
- Modalidad
- Internacionalización
- Bibliografía

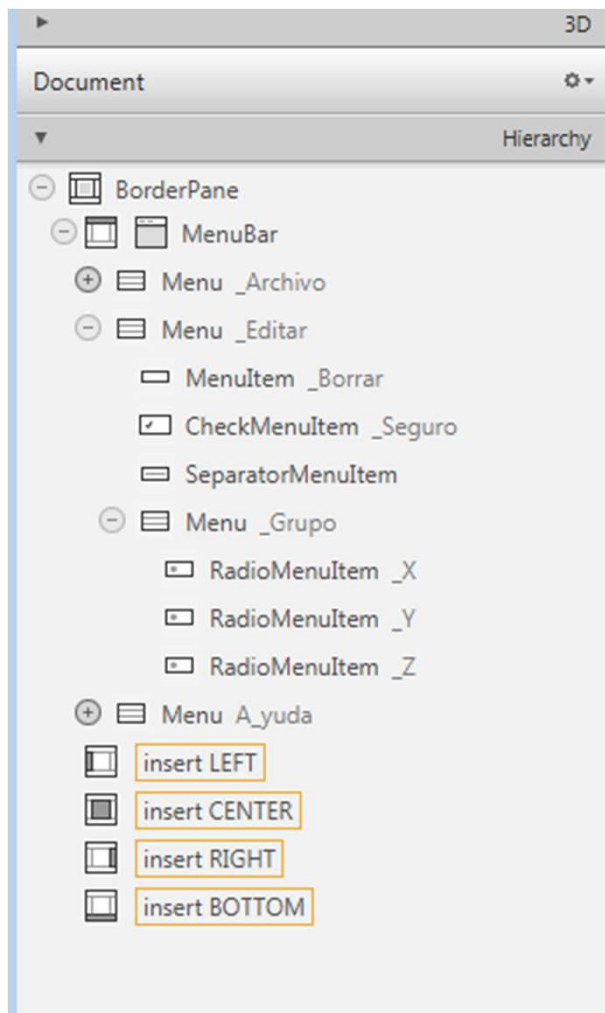
# Menús

- El menú de una aplicación muestra al usuario las acciones disponibles
- Están organizados jerárquicamente en varios menús de alto nivel (Archivo, Edición, Ayuda...)
  - Un menú puede tener submenús, que puede tener submenús...
- Se pueden definir elementos que se comporten como *checkboxes* y como *radiobuttons*.
- Se pueden asociar atajos de teclado para trabajar con los menús sin ratón

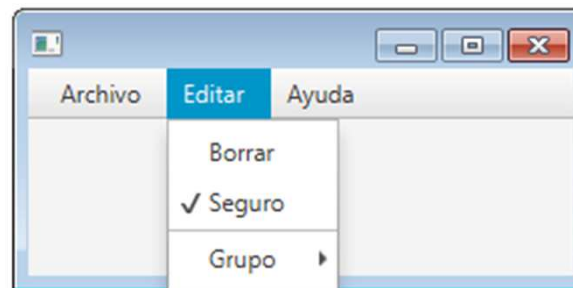
# Menús

- Clases relacionadas con menús:
  - MenuBar: barra de menú, que contiene una entrada por cada menú de alto nivel
  - Menu: menú de alto nivel, que puede contener elementos de menú y otros menús (que contienen más elementos y son submenús del anterior). Organizado en forma de árbol.
  - MenuItem: elemento hoja de un menú, encargado de lanzar una acción concreta
  - SeparatorMenuItem: un separador
  - CheckMenuItem: similar a un CheckBox, con un estado de seleccionado/no seleccionado
  - RadioMenuItem: igual que el CheckMenuItem, pero en el que sólo uno de los RadioMenuItem que comparten un ToggleGroup puede estar seleccionado

# Construyendo el menú con SceneBuilder

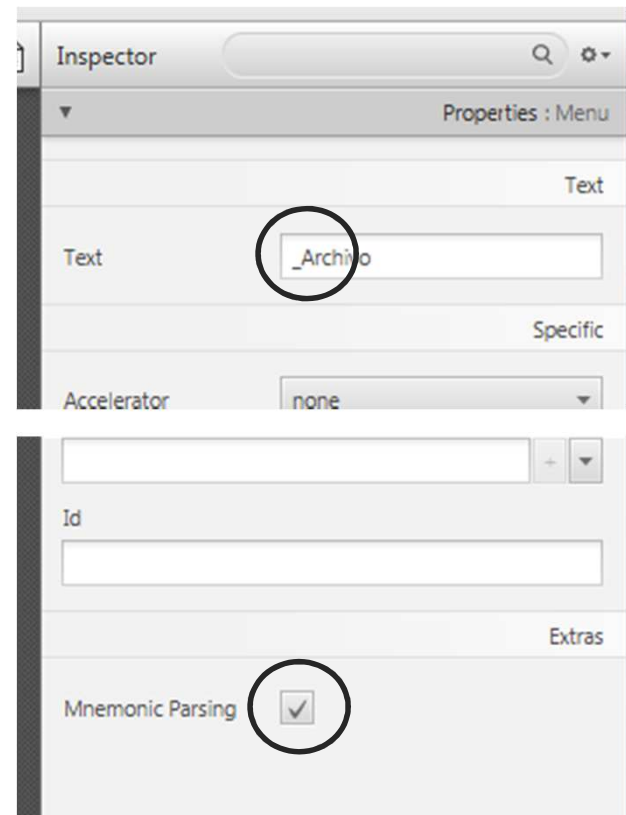
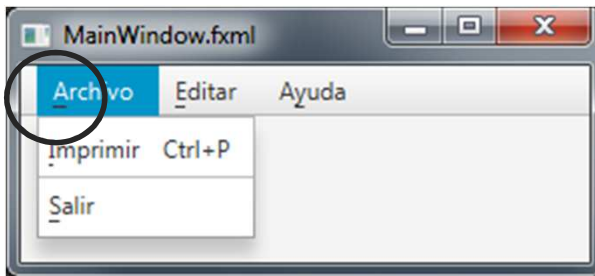


- Normalmente pondremos la barra de menú en el borde superior de un BorderPane
- Arrastramos sobre el panel Hierarchy los elementos que queremos utilizar
- Podemos darle el nombre desde este panel haciendo doble clic
- Se pueden añadir menús dentro menús



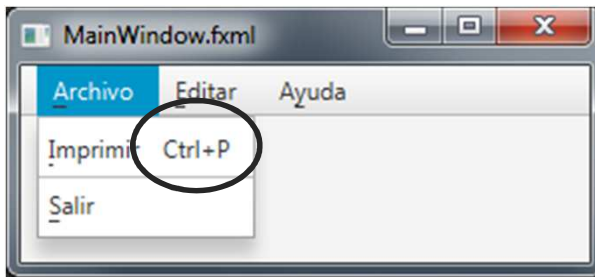
# Atajos de teclado

- Un elemento de menú se puede ejecutar seleccionándolo con el ratón, pero también mediante:
  - Tecla de acceso

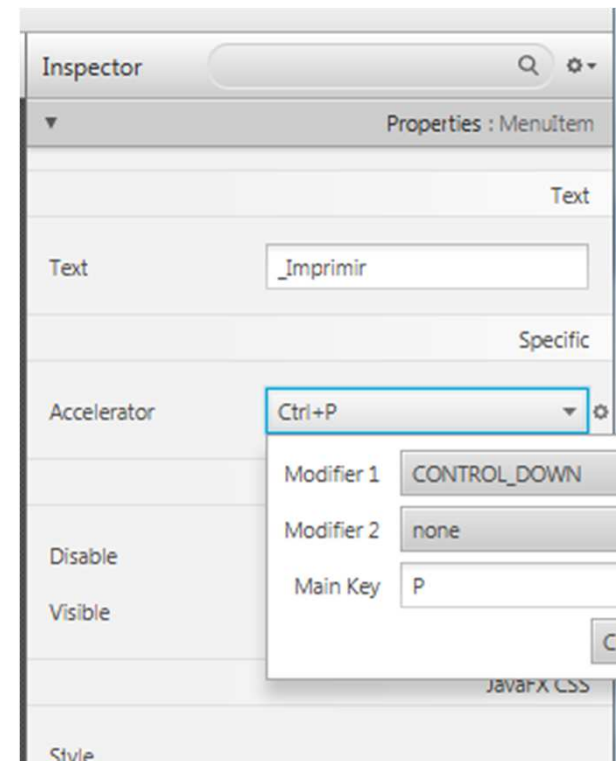


# Atajos de teclado

- Un elemento de menú se puede ejecutar seleccionándolo con el ratón, pero también mediante:
  - Atajos de teclado



El modificador SHORTCUT representa la tecla Ctrl en Windows y Meta en Mac.



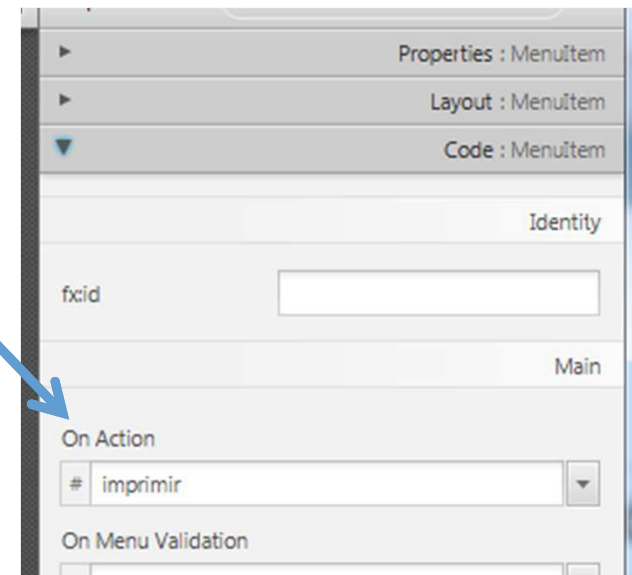
# Atendiendo eventos de menú

- Los elementos de un menú se comportan como un botón normal y corriente, por lo que sólo hay que asociarlos con un método que reciba un `ActionEvent`:

```
@FXML
private void imprimir(ActionEvent e) {
    System.out.println("Imprimir");
}
```

ó

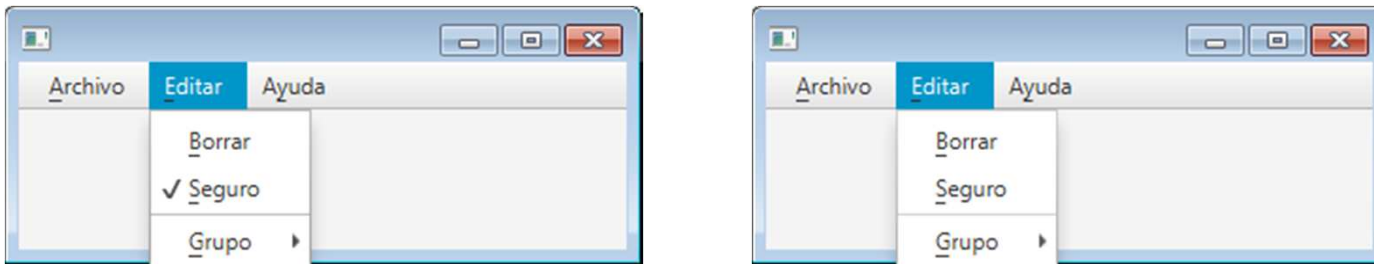
```
@FXML private MenuItem menuBorrar;
@FXML void initialize() {
    menuBorrar.setOnAction(this::borrar);
}
private void borrar(ActionEvent e) {
    System.out.println("Borrar");
}
```





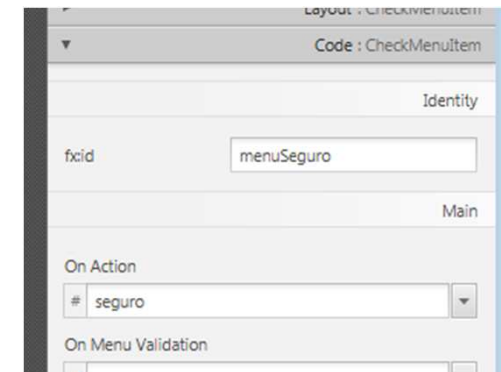
# CheckMenuItem

- Es una mezcla entre un MenuItem, y un CheckBox



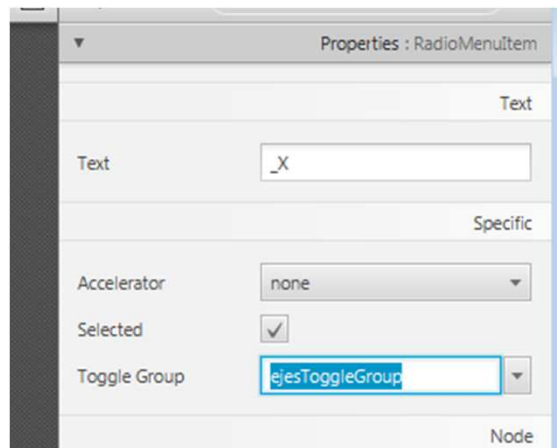
- Cuando se selecciona el elemento del menú, se ejecuta el manejador asociado, y también se cambia su estado (seleccionado/no seleccionado)

```
@FXML private CheckMenuItem menuSeguro;  
@FXML private void seguro(ActionEvent e) {  
    System.out.println("Estas seguro: " +  
        (menuSeguro.isSelected() ? "SI" : "NO"));  
}
```

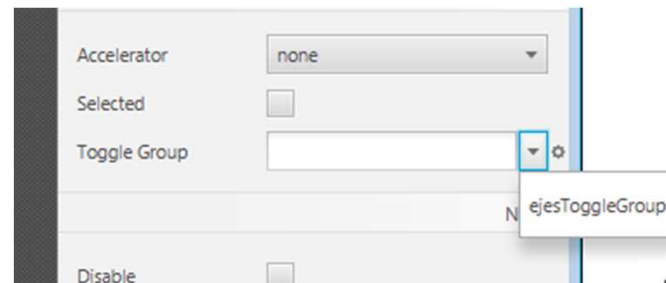


# RadioMenuItem

- Se comporta como un CheckMenuItem, con la salvedad que sólo un elemento de un mismo ToggleGroup puede estar seleccionado a la vez



1. Dale un nombre cualquiera al *toggle group* del primer *radio button*

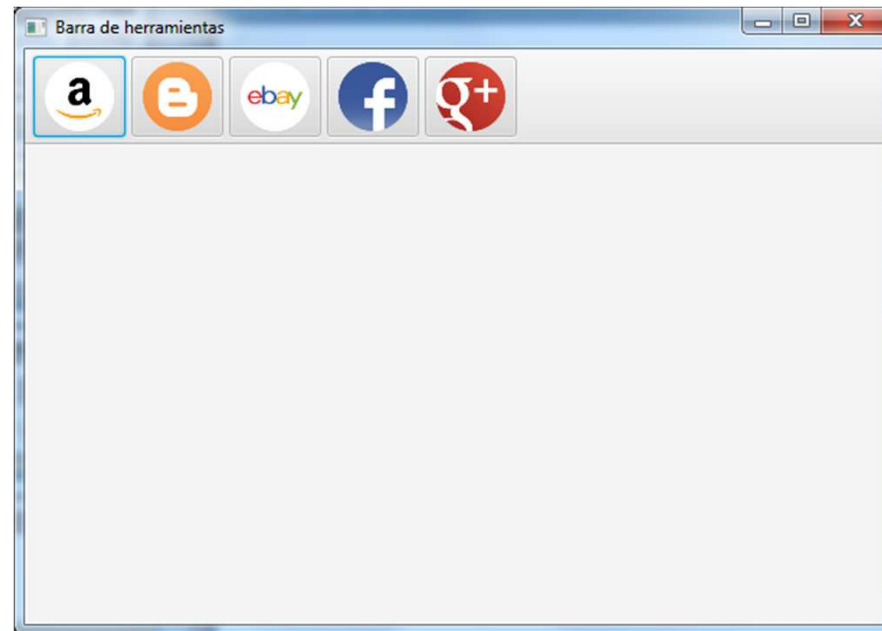


2. Selecciona el mismo nombre de la lista en el resto de *radio buttons* del grupo



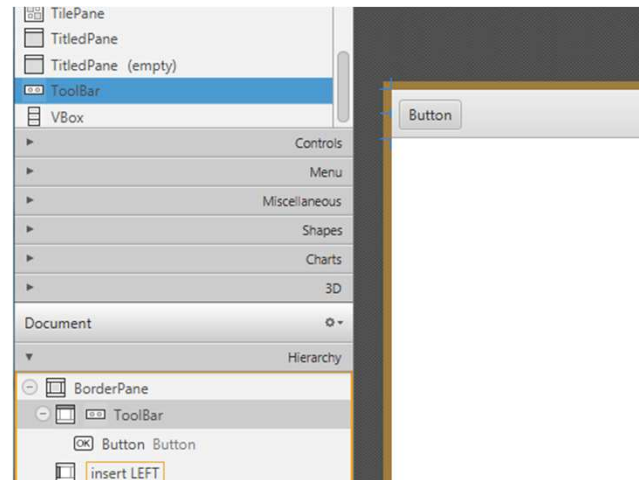
# Barras de herramientas

- La clase ToolBar de JavaFX implementa un contenedor de botones, que se puede utilizar para implementar una barra de herramientas



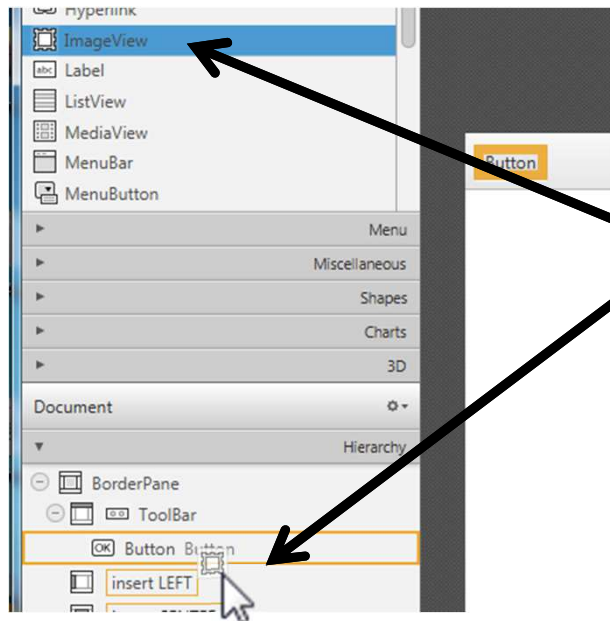
# Barras de herramientas

- La barra de herramientas puede contener cualquier tipo de nodo, pero lo normal es usar botones

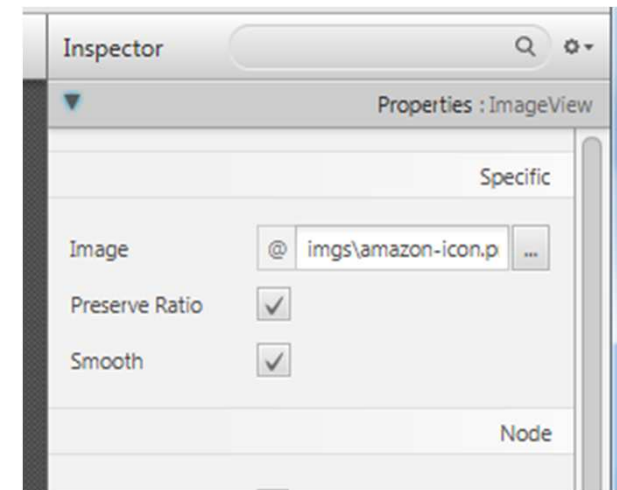


# Barras de herramientas

- Los botones normalmente muestran una etiqueta de texto, pero pueden mostrar también una imagen



1. Arrastra un ImageView al botón



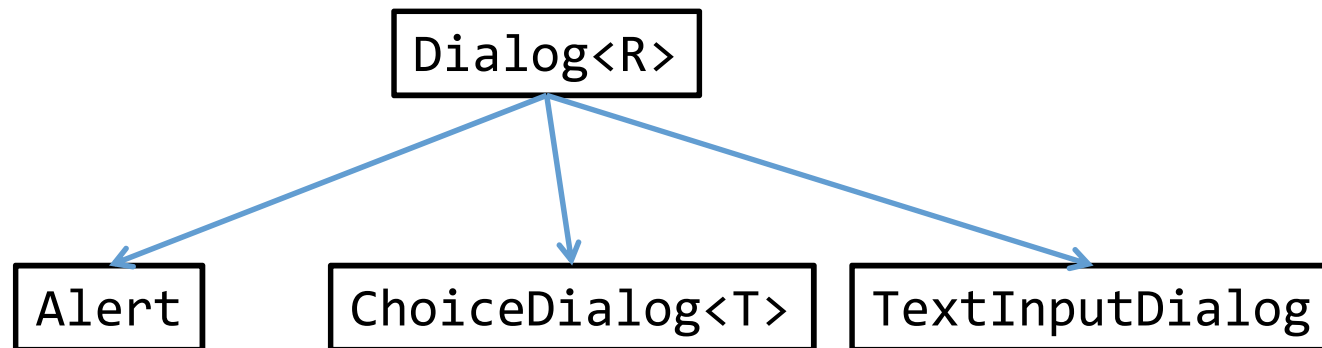
2. Asigna la imagen al ImageView.  
¡Cuidado! Debe estar dentro del directorio src

# Diálogos

- Un diálogo es una ventana que se abre durante la ejecución para pedir información al usuario
  - Diálogos modales: mientras que esté abierto el diálogo, el usuario no puede interactuar con el resto de la aplicación (p.e., el diálogo Imprimir)
  - Diálogos no modales: el usuario puede interactuar con el diálogo o con el resto de la aplicación indistintamente (p.e., el diálogo Buscar)

# Clases involucradas


- Paquete: `javafx.scene.control`



Cuadros de diálogo  
de notificación

Cuadros de diálogo de entrada de  
información (un elemento de una lista o una  
cadena de texto, respectivamente)

# Estructura general de un diálogo

	Título
	Cabecera ( <i>masthead</i> ) en negrita y con iconos
	Contenido
	Botones

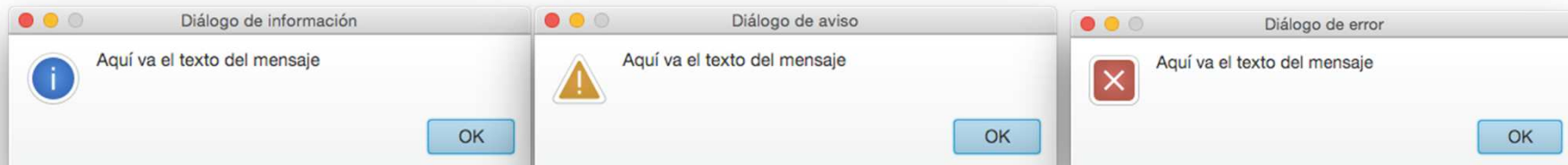


# Diálogos estándar

- Con cabecera



- Sin cabecera



# Diálogos estándar

- Código para crear y mostrar el diálogo:

```
Alert alert = new Alert(AlertType.INFORMATION);  
    // ó AlertType.WARNING ó AlertType.ERROR ó AlertType.CONFIRMATION  
alert.setTitle("Diálogo de información");  
alert.setHeaderText("Cabecera");  
    // ó null si no queremos cabecera  
alert.setContentText("Aquí va el texto del mensaje");  
alert.showAndWait();
```

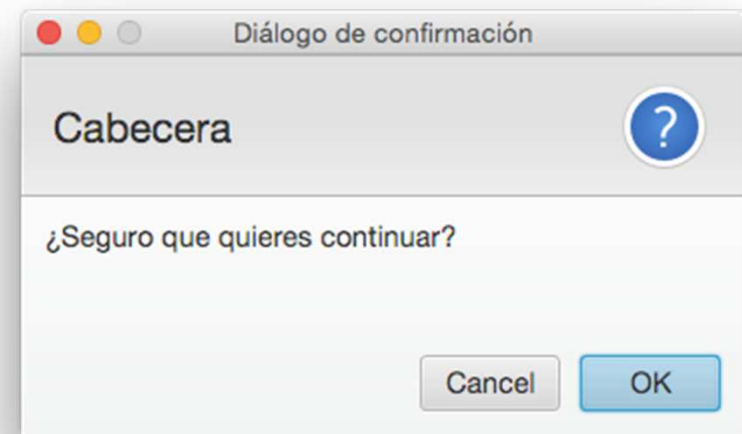
// También

```
Alert alert = new Alert(AlertType.INFORMATION, "Contenido");
```

# Diálogo de confirmación

```
Alert alert = new Alert(AlertType.CONFIRMATION);  
alert.setTitle("Diálogo de confirmación");  
alert.setHeaderText("Cabecera");  
alert.setContentText("¿Seguro que quieres continuar?");
```

```
Optional<ButtonType> result = alert.showAndWait();  
if (result.isPresent() && result.get() == ButtonType.OK){  
    System.out.println("OK");  
} else {  
    System.out.println("CANCEL");  
}
```



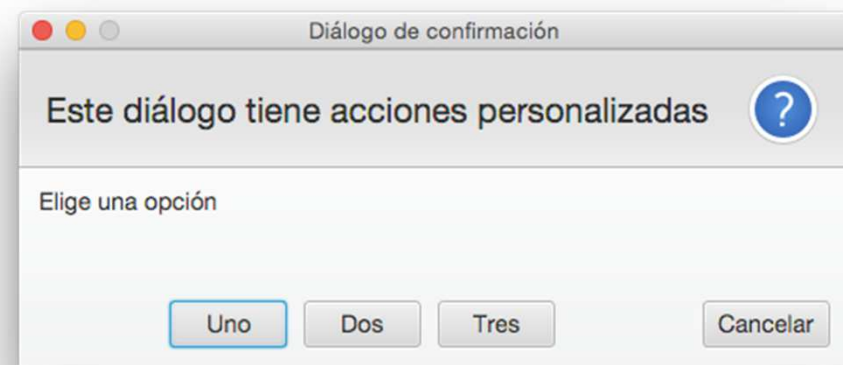
# Diálogo de confirmación con acciones personalizadas

```
Alert alert = new Alert(AlertType.CONFIRMATION);
alert.setTitle("Diálogo de confirmación");
alert.setHeaderText("Este diálogo tiene acciones personalizadas");
alert.setContentText("Elige una opción");

ButtonType buttonTypeOne = new ButtonType("Uno");
ButtonType buttonTypeTwo = new ButtonType("Dos");
ButtonType buttonTypeThree = new ButtonType("Tres");
ButtonType buttonTypeCancel = new ButtonType("Cancelar", ButtonData.CANCEL_CLOSE);

alert.getButtonTypes().setAll(buttonTypeOne, buttonTypeTwo, buttonTypeThree, buttonTypeCancel);

Optional<ButtonType> result = alert.showAndWait();
if (result.isPresent()) {
    if (result.get() == buttonTypeOne)
        System.out.println("Uno");
    else if (result.get() == buttonTypeTwo)
        System.out.println("Dos");
    else if (result.get() == buttonTypeThree)
        System.out.println("Tres");
    else
        System.out.println("Cancelar");
}
```



# Diálogo de entrada de texto

```
TextInputDialog dialog = new TextInputDialog("Pepe"); // Por defecto  
dialog.setTitle("Diálogo de entrada de texto");  
dialog.setHeaderText("Cabecera");  
dialog.setContentText("Introduce tu nombre:");
```

```
Optional<String> result = dialog.showAndWait();  
// Obteniendo el resultado (pre Java 8)  
if (result.isPresent()){  
    System.out.println("Hola " + result.get());  
}
```

```
// Obteniendo el resultado con una lambda  
result.ifPresent(name -> System.out.println("Hola " + name));
```



# Diálogo de selección

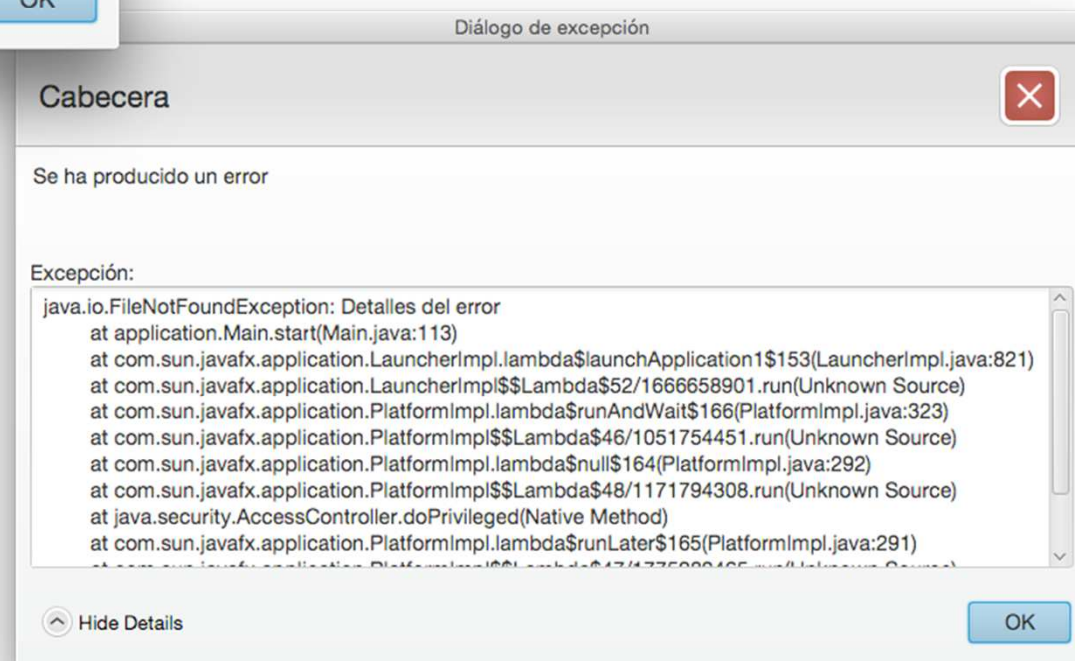
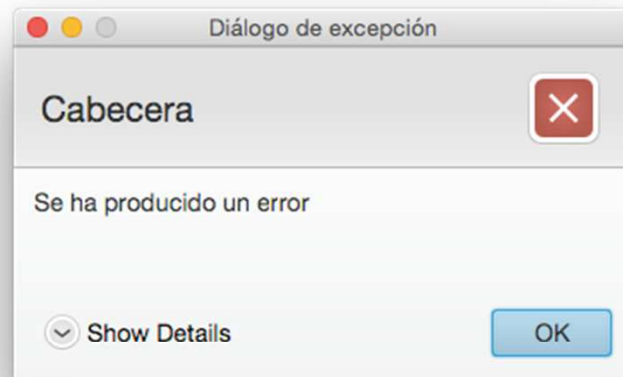
```
List<String> choices = new ArrayList<>();  
choices.add("uno");  
choices.add("dos");  
choices.add("tres");
```

```
ChoiceDialog<String> dialog = new ChoiceDialog<>("dos", choices);  
dialog.setTitle("Diálogo de selección");  
dialog.setHeaderText("Cabecera");  
dialog.setContentText("Elige un número:");
```

```
Optional<String> result = dialog.showAndWait();  
// Pre Java 8  
if (result.isPresent()) {  
    System.out.println("Has elegido: " + result.get());  
}  
// Obteniendo el resultado con una lambda  
result.ifPresent(number-> System.out.println("Has elegido: " + number));
```



# Diálogo de error



# Diálogo de error

```
Alert alert = new Alert(AlertType.ERROR);
alert.setTitle("Diálogo de excepción");
alert.setHeaderText("Cabecera");
alert.setContentText("Se ha producido un error");
```

```
Exception ex = new FileNotFoundException("Detalles del error");
```

```
StringWriter sw = new StringWriter();
PrintWriter pw = new PrintWriter(sw);
ex.printStackTrace(pw);
String exceptionText = sw.toString();
```

```
Label label =
    new Label("Excepción:");
```

```
TextArea textArea =
    new TextArea(exceptionText);
textArea.setEditable(false);
textArea.setWrapText(true);
```

```
textArea.setMaxWidth(Double.MAX_VALUE);
textArea.setMaxHeight(Double.MAX_VALUE);
GridPane.setVgrow(textArea,
    Priority.ALWAYS);
GridPane.setHgrow(textArea,
    Priority.ALWAYS);
```

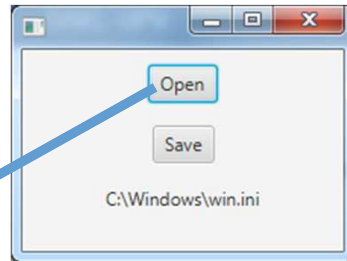
```
GridPane expContent = new GridPane();
expContent.setMaxWidth(Double.MAX_VALUE);
expContent.add(label, 0, 0);
expContent.add(textArea, 0, 1);
```

```
alert.getDialogPane().
    setExpandableContent(expContent);
```

```
alert.showAndWait();
```

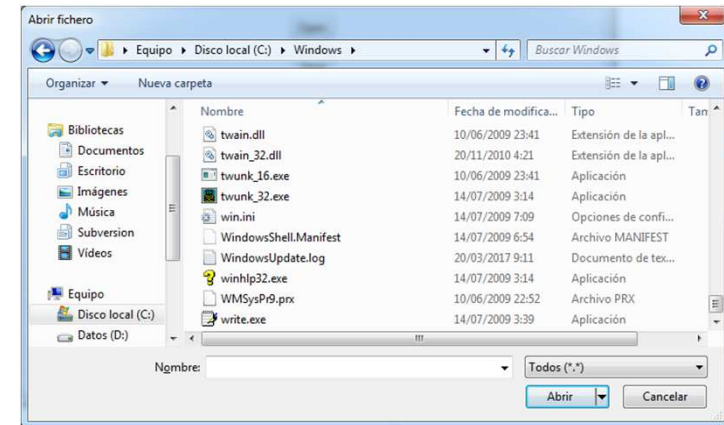


# Diálogos para abrir/guardar ficheros



@FXML

```
private void onOpen(ActionEvent event) {  
    FileChooser fileChooser = new FileChooser();  
    fileChooser.setTitle("Abrir fichero");  
    fileChooser.getExtensionFilters().addAll(  
        new ExtensionFilter("Ficheros de texto", "*.txt"),  
        new ExtensionFilter("Imágenes", "*.png", "*.jpg", "*.gif"),  
        new ExtensionFilter("Sonidos", "*.wav", "*.mp3", "*.aac"),  
        new ExtensionFilter("Todos", "*.*"));  
    File selectedFile = fileChooser.showOpenDialog(  
        ((Node)event.getSource()).getScene().getWindow());  
    if (selectedFile != null) {  
        label.setText(selectedFile.getAbsolutePath());  
    }  
}
```



*showOpenDialog* tiene un parámetro: la ventana (*stage*) padre del diálogo. Si no es null, el diálogo será modal con respecto a la ventana. Este código muestra cómo conseguir el *stage* en el que se encuentra un nodo.

# Diálogos para abrir/guardar ficheros

- Otras funciones de `FileChooser`:
  - `File showSaveDialog(Window ownerWindow)`
    - Abrir un diálogo para guardar fichero
  - `List<File> showOpenMultipleDialog(Window ownerWindow)`
    - Abrir un diálogo para abrir múltiples ficheros
  - `final void setInitialFileName(String value)`
    - En un fichero para guardar, establece el nombre por defecto del fichero
  - `final void setInitialDirectory(File value)`
    - Directorio que muestra el diálogo al abrirse

# Modalidad

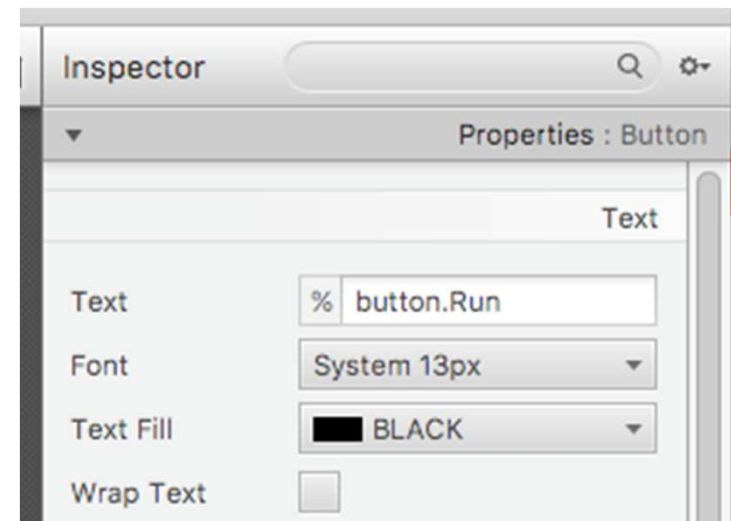
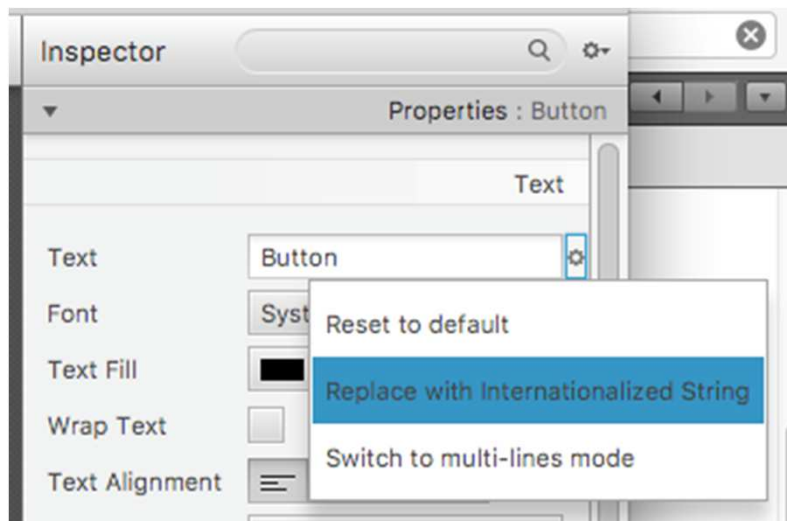
- Los diálogos en JavaFX son modales por defecto (es decir, no se puede interactuar con el resto de la aplicación)
  - Se puede establecer la modalidad con el método `dialog.initModality(modality)`  
donde:
    - `modality`: `Modality.NONE`, `Modality.WINDOW_MODAL`, ó `Modality.APPLICATION_MODAL`
- Aparte de la modalidad, se puede decidir si la llamada de apertura del diálogo es bloqueante o no
  - Bloqueante: `showAndWait()`
  - No bloqueante: `show()`

# Otras opciones

- Establecer el padre de un diálogo
  - `dialog.initOwner(parentWindow);`
  - Si no se establece o es `null`, el diálogo no depende de otra ventana

# Internacionalización

- SceneBuilder facilita la tarea de traducir una aplicación permitiendo definir, en vez de la etiqueta de un control, una variable cuyo valor dependerá del idioma



# Internacionalización

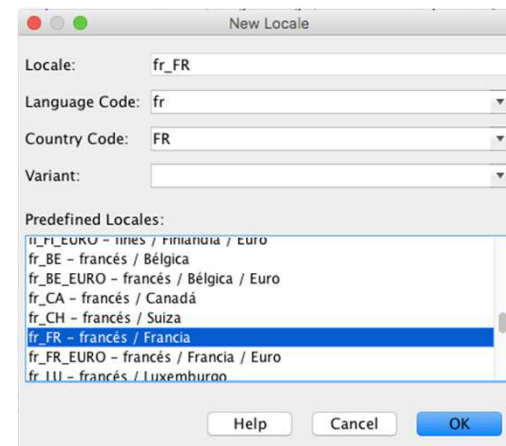
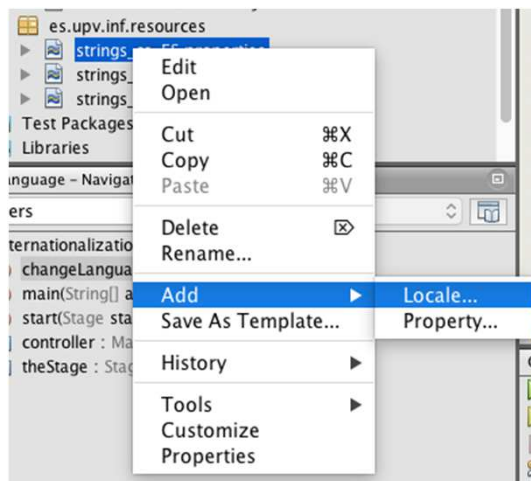
- En el fichero FXML, las etiquetas a traducir se distinguen por empezar por un signo %

```
<Button mnemonicParsing="false" text="%button.Run" />
```

- ¡Cuidado!: en SceneBuilder, no se puede escribir directamente el nombre de una variable con un % delante: hay que seleccionar la opción “Replace with Internationalized String” e introducir el nombre de la variable, sin %

# Internacionalización

- Cada traducción de las variables a un idioma dado se almacena en un fichero de propiedades (también llamado *resource bundle* en Java)
- En NetBeans:
  - File\New File...\Other\Properties File
  - Su nombre debe tener la forma: <base>\_<idioma>\_<pais>.properties
  - Haciendo clic derecho sobre el fichero se pueden añadir más:



# Internacionalización

- La clase Locale en Java representa una región geográfica, política o cultural
- Hay operaciones que dependen de un objeto Locale
  - Por ejemplo, representar un número: 3,1416 o 3.1416
- Contiene varios campos con información sobre el idioma (“en”, “ja”...), alfabeto (latino, cirílico...), país o región (“ES”, “GB”...), etc
- En tiempo de ejecución, Java selecciona un Locale teniendo en cuenta la configuración de la máquina



```
NumberFormat defNum = NumberFormat.getInstance();
NumberFormat usNum = NumberFormat.getInstance(US);
NumberFormat jpNum = NumberFormat.getInstance(JAPAN);
System.out.println("DEF Num: " + defNum.format(3.1416));
System.out.println("US Num: " + usNum.format(3.1416));
System.out.println("JAPAN Num: " + jpNum.format(3.1416));

NumberFormat defCurr = NumberFormat.getCurrencyInstance();
NumberFormat usCurr = NumberFormat.getCurrencyInstance(US);
NumberFormat jpCurr = NumberFormat.getCurrencyInstance(JAPAN);
System.out.println("DEF Currency: " + defCurr.format(3.1416));
System.out.println("US Currency: " + usCurr.format(3.1416));
System.out.println("JAPAN Currency: " + jpCurr.format(3.1416));

NumberFormat defPercent = NumberFormat.getPercentInstance();
NumberFormat usPercent = NumberFormat.getPercentInstance(US);
NumberFormat jpPercent = NumberFormat.getPercentInstance(FRENCH);
System.out.println("DEF Percent: " + defPercent.format(0.1416));
System.out.println("US Percent: " + usPercent.format(0.1416));
System.out.println("JAPAN Percent: " + jpPercent.format(0.1416));

Locale defaultLoc = Locale.getDefault();
System.out.println(defaultLoc);
System.out.println(defaultLoc.getDisplayCountry());
System.out.println(defaultLoc.getDisplayLanguage());
System.out.println(defaultLoc.getDisplayName());
```

```
DEF Num: 3,142
US Num: 3.142
JAPAN Num: 3.142

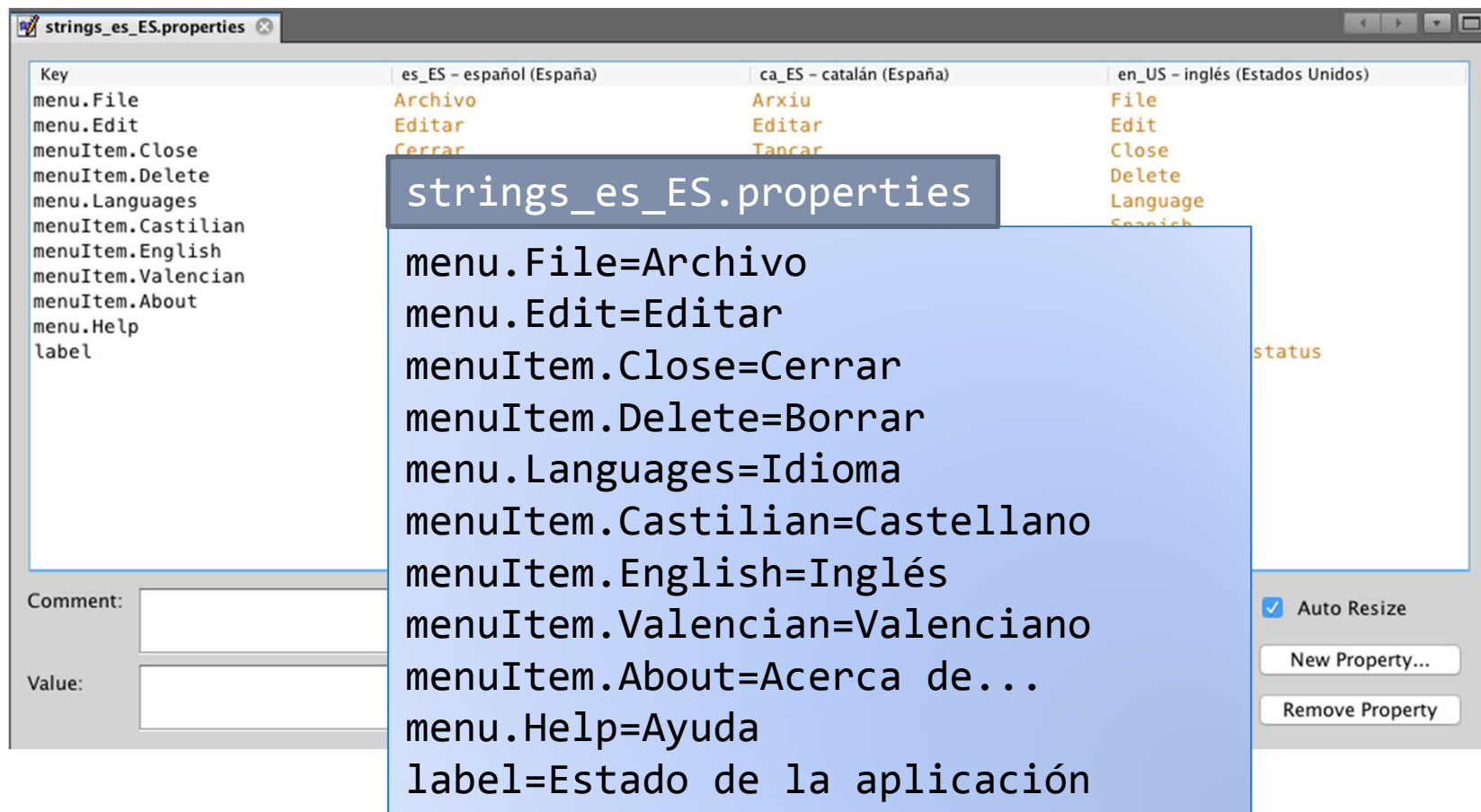
DEF Currency: 3,14 €
US Currency: $3.14
JAPAN Currency: ¥3

DEF Percent: 14%
US Percent: 14%
JAPAN Percent: 14 %

es_ES
España
español
español (España)
```

# Internacionalización


- NetBeans tiene un editor que facilita la traducción de las cadenas (botón derecho sobre un *bundle*, Open):



# Internacionalización

- En la clase principal, al cargar el FXML:

```
public void start(Stage stage) throws Exception {  
    // Seleccionar el locale por defecto  
    Locale locale = Locale.getDefault();  
    // Cargar el bundle (p.e., strings_es_ES.properties)  
    ResourceBundle bundle =  
        ResourceBundle.getBundle("es.upv.inf.resources.strings", locale);  
    // Pasar el bundle al FXMLLoader  
    Parent root =  
        FXMLLoader.load(getClass().getResource("MainWindow.fxml"), bundle);  
    Scene scene = new Scene(root);  
    stage.setScene(scene);  
    stage.show();  
}  
  
// En el controlador...  
@Override  
public void initialize(URL url, ResourceBundle rb) {
```



# Internacionalización

- Con los pasos anteriores, todas las cadenas de la interfaz definidas mediante variables tomarán el valor almacenado en el bundle
- Como en el controlador se recibe el bundle, le podemos pedir cadenas traducidas en tiempo de ejecución (por ejemplo, para ir cambiando el valor de una etiqueta en tiempo de ejecución):

```
private ResourceBundle bundle;
@FXML private Label status;
@Override
public void initialize(URL url, ResourceBundle rb) {
    bundle = rb;
}
private void updateLabel(String key) {
    status.setText(bundle.getString(key));
}
```

# Ejercicio

- Implementa la siguiente aplicación



Barra de estado (una Label)

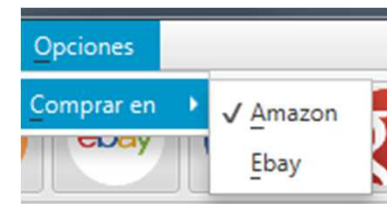
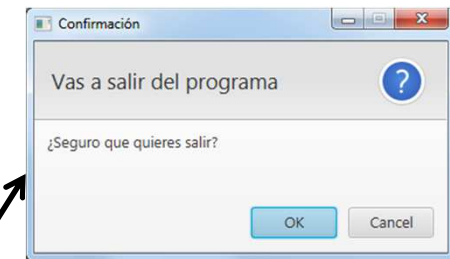
Archivo  
Salir

Lanzar

Amazon  
Blogger  
Ebay  
Facebook  
Google+

Opciones

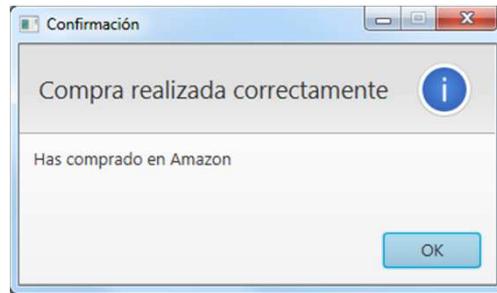
Comprar en  
Amazon  
Ebay



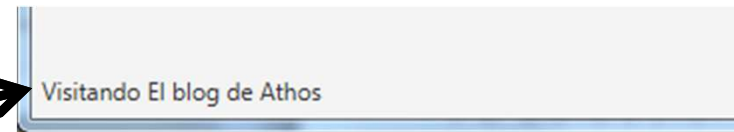
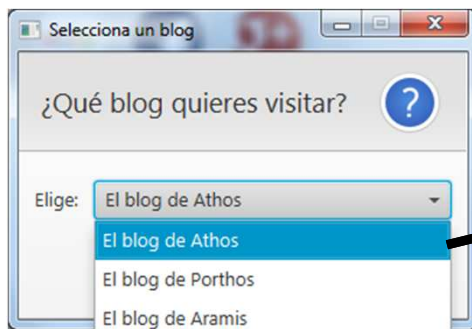
RadioMenuItem

# Ejercicio

- Al pulsar Amazon o Ebay, se comprobará si está marcada en el menú Opciones la misma opción, y se mostrará un mensaje de confirmación o un mensaje de error

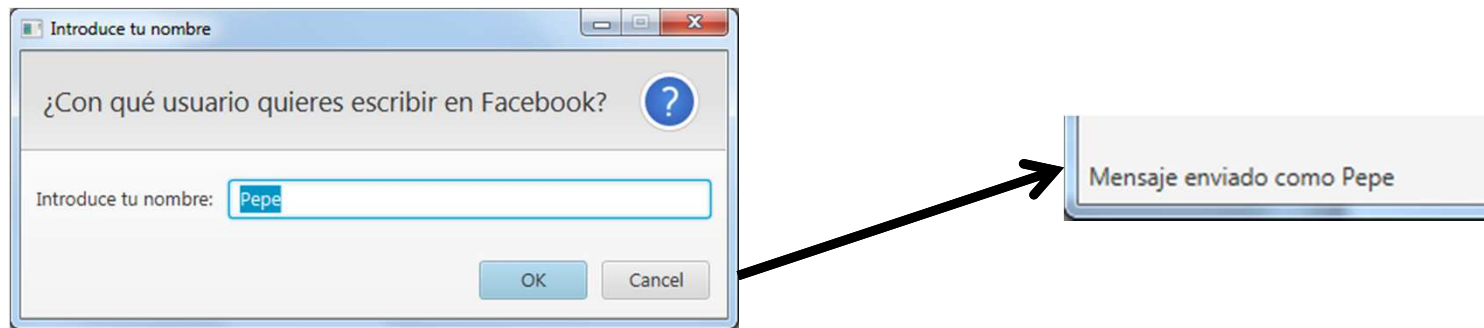


- Al seleccionar Blogger, se pedirá qué blog se desea visitar, y se mostrará en la barra de estado:



# Ejercicio

- Al seleccionar Facebook, se pedirá un nombre de usuario con el que escribir un mensaje, y luego se mostrará dicho nombre en la barra de estado



- Ampliación: traduce la aplicación a un idioma distinto al de tu sistema, y carga dicho Locale a mano para que la aplicación se muestre en dicho idioma

# Bibliografía

- Diálogos: <http://code.makery.ch/blog/javafx-dialogs-official>
  - <https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Dialog.html>
  - <https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Alert.html>
  - <https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/TextInputDialog.html>
  - <https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/ChoiceDialog.html>
  - <https://docs.oracle.com/javase/8/javafx/api/javafx/stage/FileChooser.html>
- Internacionalización
  - [http://docs.oracle.com/javafx/scenebuilder/1/user\\_guide/i18n-support.htm](http://docs.oracle.com/javafx/scenebuilder/1/user_guide/i18n-support.htm)
  - <https://docs.oracle.com/javase/8/docs/api/java/util/Locale.html>
- C. Dea y otros. JavaFX 8. Introduction by Example. Apress, 2014