



CENTRE ESPECÍFIC  
D'EDUCACIÓ A DISTÀNCIA DE  
LA COMUNITAT VALENCIANA

## UF12.- PROGRAMACIÓ GRÀFICA

- *Tutorial* -

PROGRAMACIÓ  
CFGS DAW

Guillermo Garrido Portes  
Joan Vicent Cassany Coscollà

[g.garridoportes@edu.gva.es](mailto:g.garridoportes@edu.gva.es)  
[jv.cassanyicoscolla@edu.gva.es](mailto:jv.cassanyicoscolla@edu.gva.es)

2022/2023 1

# 1. INTRODUCCIÓ

Una interfície gràfica d'usuari (*GUI*, per les seues sigles en anglès) és un conjunt d'elements visuals i controls que permeten als usuaris interactuar amb un programari o sistema informàtic de manera intuïtiva i visual.

La GUI proporciona una capa d'abstracció sobre els comandaments i operacions complexes del sistema subjacent, permetent als usuaris interactuar amb el programari de manera més accessible i senzilla.

Els elements de la GUI poden incloure finestres, botons, menús, barres d'eines, quadres de diàleg i altres elements interactius que permeten als usuaris interactuar amb el programari de manera visual i directa.

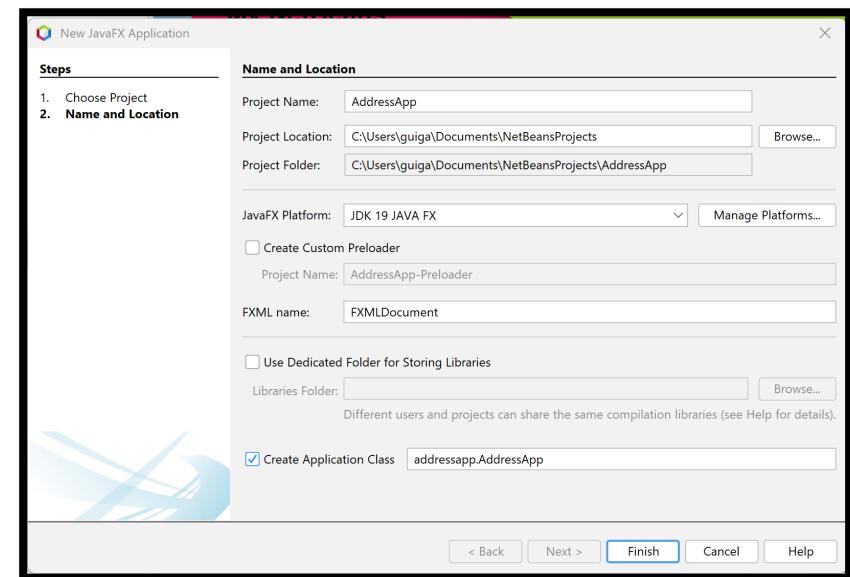
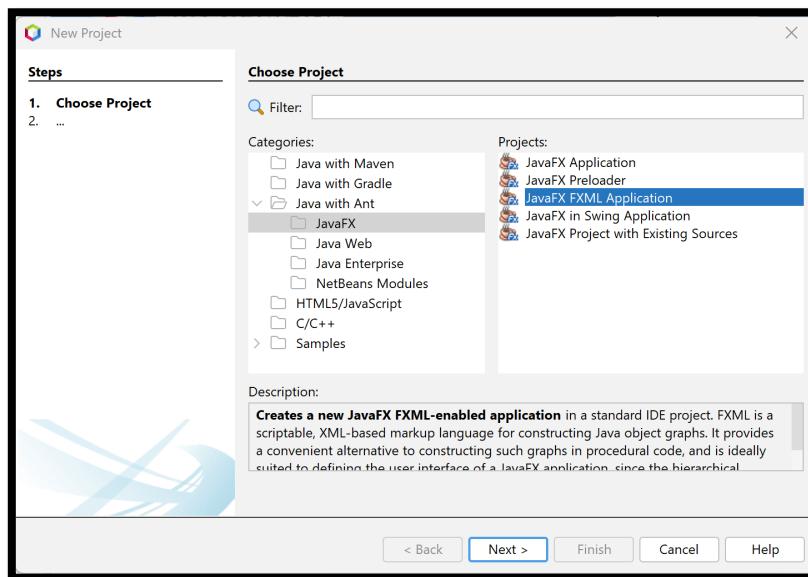
En general, la GUI pot tindre un impacte significatiu en l'eficàcia i l'eficiència de la interacció humà-computadora.

## 2. CREACIÓ DEL PROJECTE

Creem un nou projecte JavaFX -> *JavaFX FXML Application*.

Especifica el nom del projecte “**UF12AddressApp**”

Plataforma JDK 19 JAVA FX.

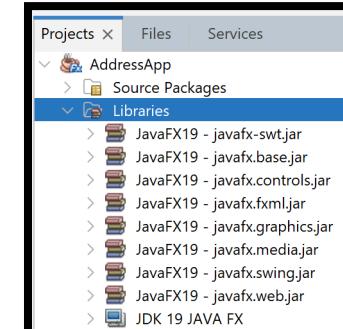
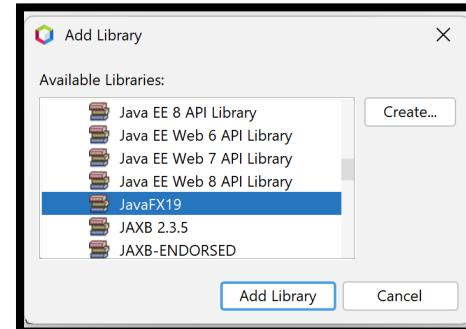
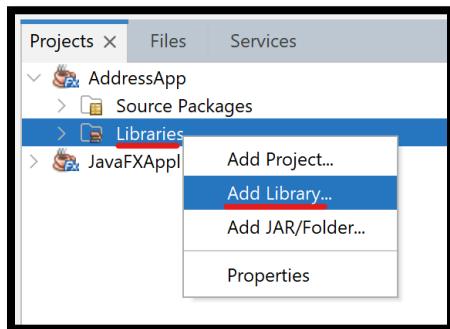


## 2. CREACIÓ DEL PROJECTE

Importem la llibreria de JavaFX.

En el ratolí fem clic dret en la carpeta “**Libraries**” i en el submenú seleccionarem l’opció “**Add Library**”.

Per últim, seleccionarem la llibreria de JavaFX que tenim configurada en el sistema.



## 2. CREACIÓ DEL PROJECTE

En l'actualitat és molt habitual trobar projectes amb l'arquitectura denominada Model-Vista-Controlador (MVC).

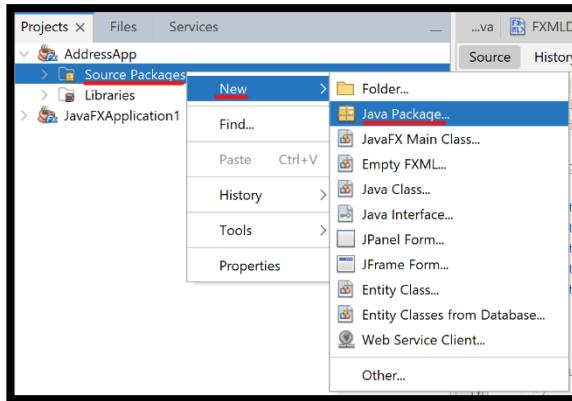
Aquesta arquitectura promou la divisió del nostre codi en tres apartats per a que siguen mes reutilitzables i mantenibles.

Els projectes Maven preconfiguren aquesta arquitectura de manera forçosa.

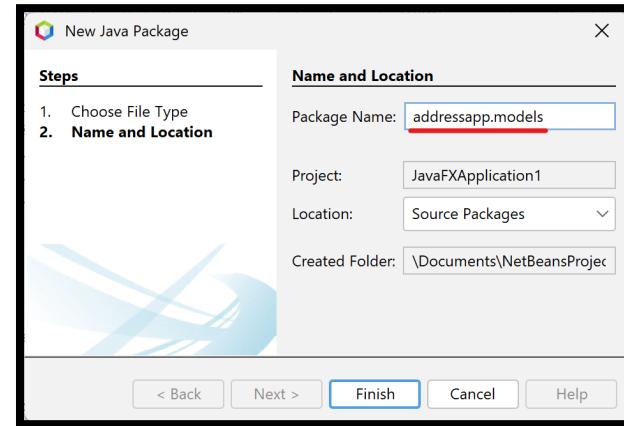
En el nostre cas anirem a poc a poc implementant aquesta separació mitjançant la creació de tres paquets separats.

## 2. CREACIÓ DEL PROJECTE

Amb el ratolí fem clic dret en la carpeta “**Source Packages**” i en el submenú seleccionarem l’opció **New -> Java Package**.

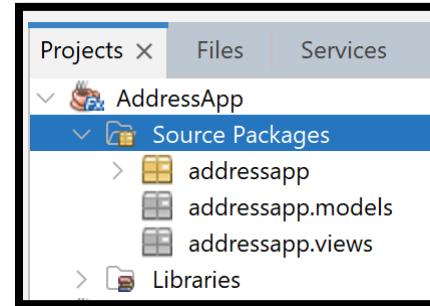
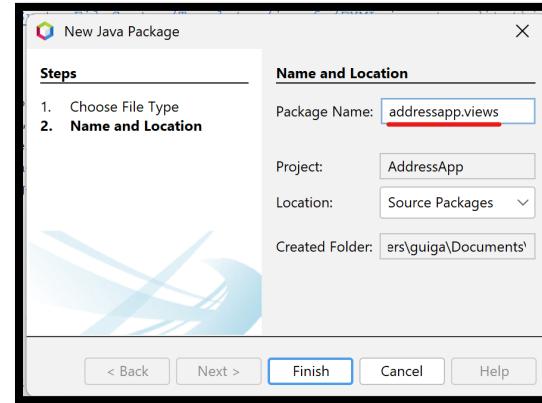


En aquest cas crearem el paquet “**uf12addressapp.models**” que contindrà les classes o models (M) de l’aplicació.



## 2. CREACIÓ DEL PROJECTE

A continuació, repetirem els passos anteriors per a crear el paquet “**uf12addressapp.views**” que contindrà les vistes (V) de l'aplicació.



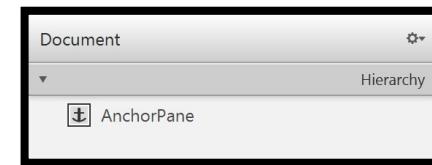
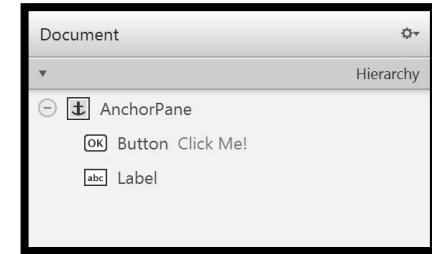
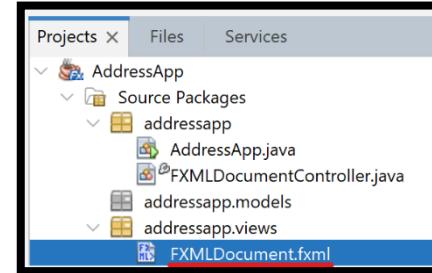
Per últim, deixarem el paquet “**uf12addressapp**” per a agrupar els controladors de l'aplicació (C).

## 2. CREACIÓ DEL PROJECTE

Movem l'arxiu **FXMLDocument.fxml** al paquet **uf12addressapp.views** i fem doble clic en ell per a què obriga el Scene Builder.

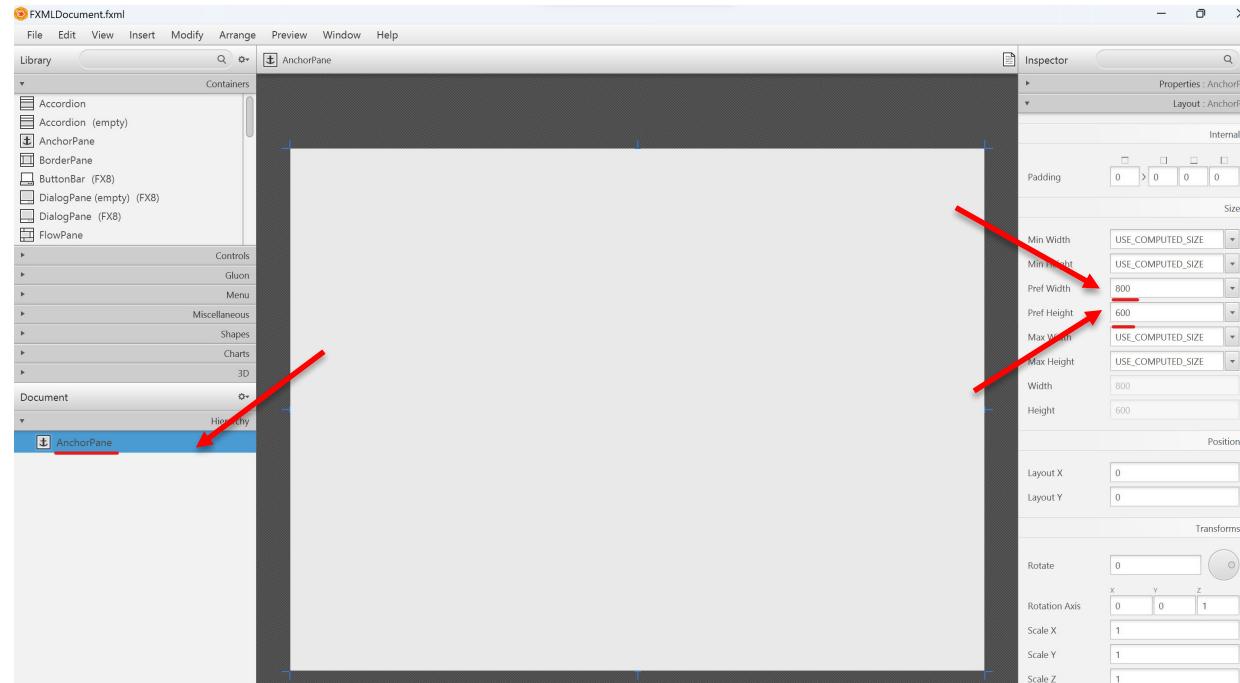
Ara veurem el Scene Builder amb un AnchorPane (visible en la jerarquia de components situada a l'esquerra).

De manera predefinida conté un botó i una etiqueta. Els esborrarem.



### 3. CREACIÓ DE LA VISTA INICIAL

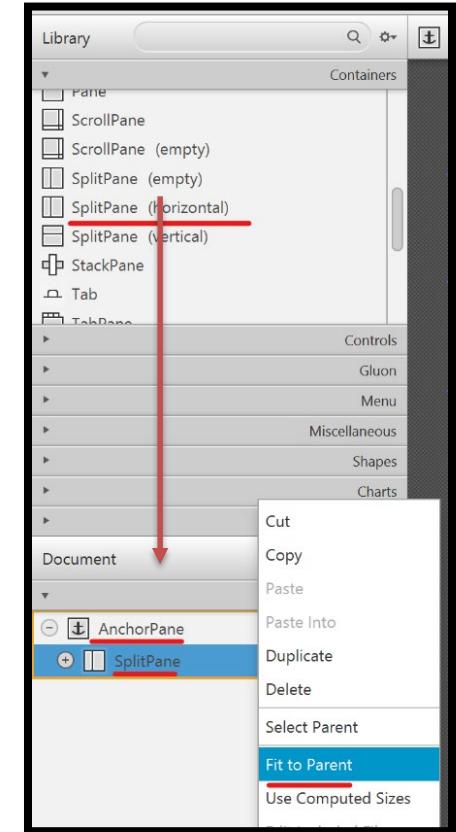
Seleccionem l'AnchorPane en la nostra jerarquia i ajustem la grandària en l'apartat Layout (a la dreta).



### 3. CREACIÓ DE LA VISTA INICIAL

Afegim un *SplitPane* (*Horizontal Flow*) arrossegant-lo des de la llibreria (Library) a l'àrea principal d'edició.

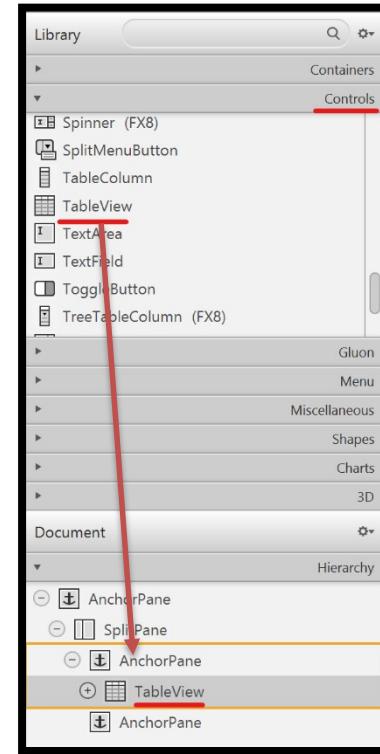
Fem clic dret sobre el *\*SplitPane* en la jerarquia i triem *\*Fit \*to \*Parent*.



### 3. CREACIÓ DE LA VISTA INICIAL

Si no apareixen ja inclosos, afegim dos AnchorPane que ens serviran per a cada apartat dels SplitPane.

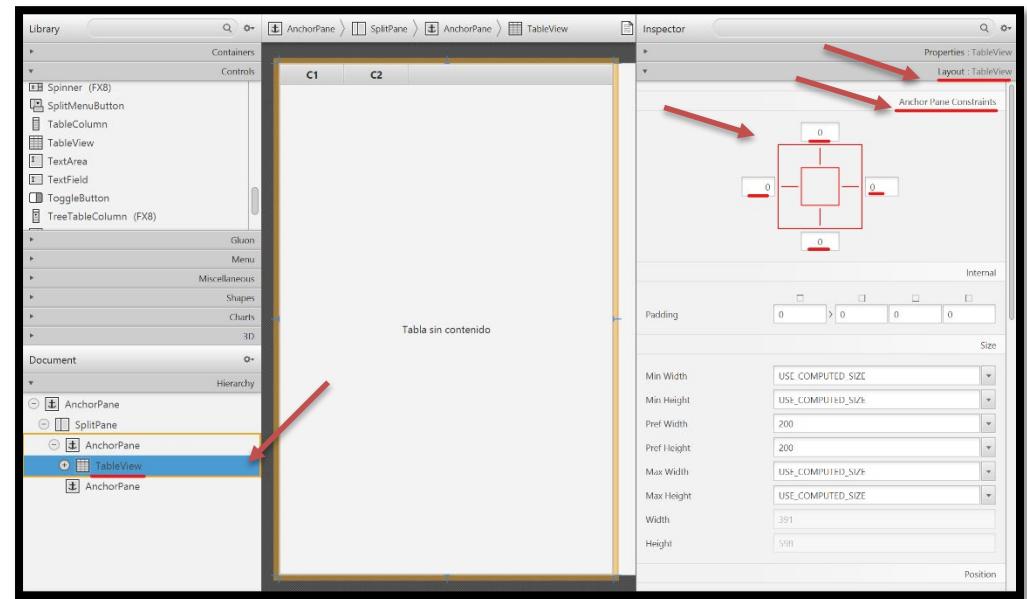
A continuació, arrosseguem un *TableView* (*Controls*) dins del primer AnchorPane del *SplitPane*.



### 3. CREACIÓ DE LA VISTA INICIAL

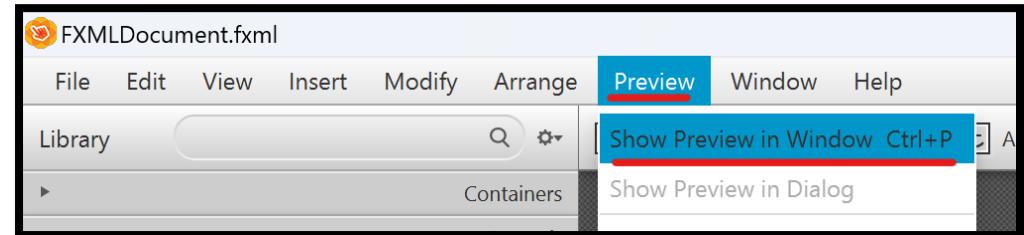
Seleccionem la TableView i establim les següents restriccions d'aparença (Layout Anchor Pane Constraints, 0-0-0-0) per a la TableView.

Dins d'un *AnchorPane* sempre es poden establir ancoratges (anchors) per a les quatre vores. Aquestes indicaran el marge que ha de deixar el TableView respecte a son pare (*AnchorPane*)



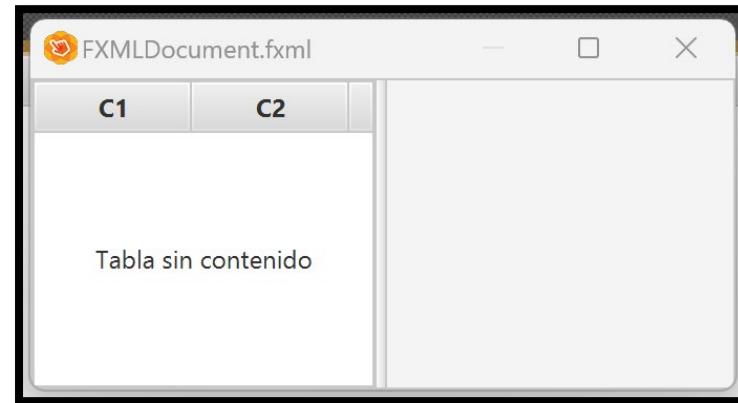
### 3. CREACIÓ DE LA VISTA INICIAL

Anirem al menú **Preview** -> “**Show Preview in Window**” per a comprovar si es visualitza correctament.



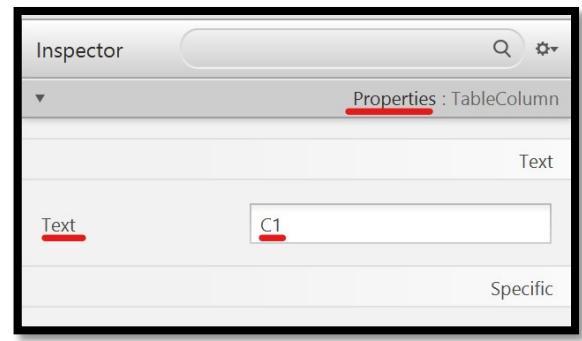
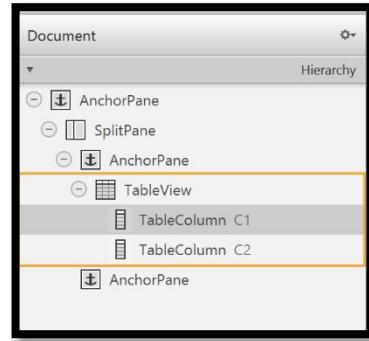
Intentem canviar la grandària de la finestra.

La TableView hauria d'ajustar la seu grandària a la grandària de la finestra, perquè està “ancorada” a les seues vores.

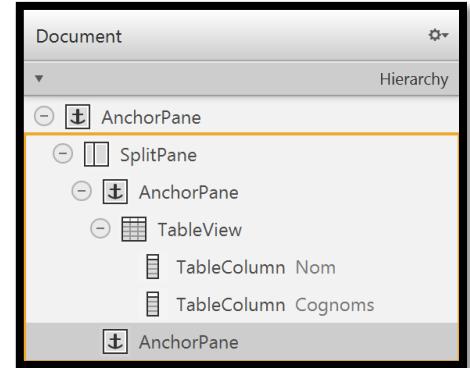


### 3. CREACIÓ DE LA VISTA INICIAL

Seleccionem una de les columnes del TableView i busquem en **Properties** l'atribut “**Text**”.

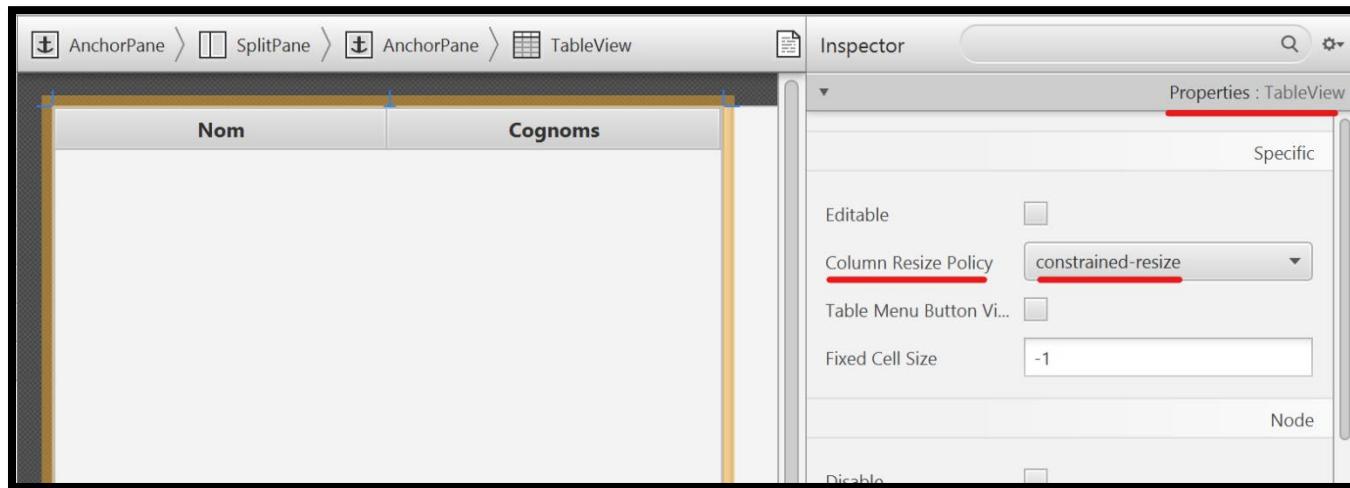


Canvia el text de les dues columnes a “Nom” i “Cognoms”.



### 3. CREACIÓ DE LA VISTA INICIAL

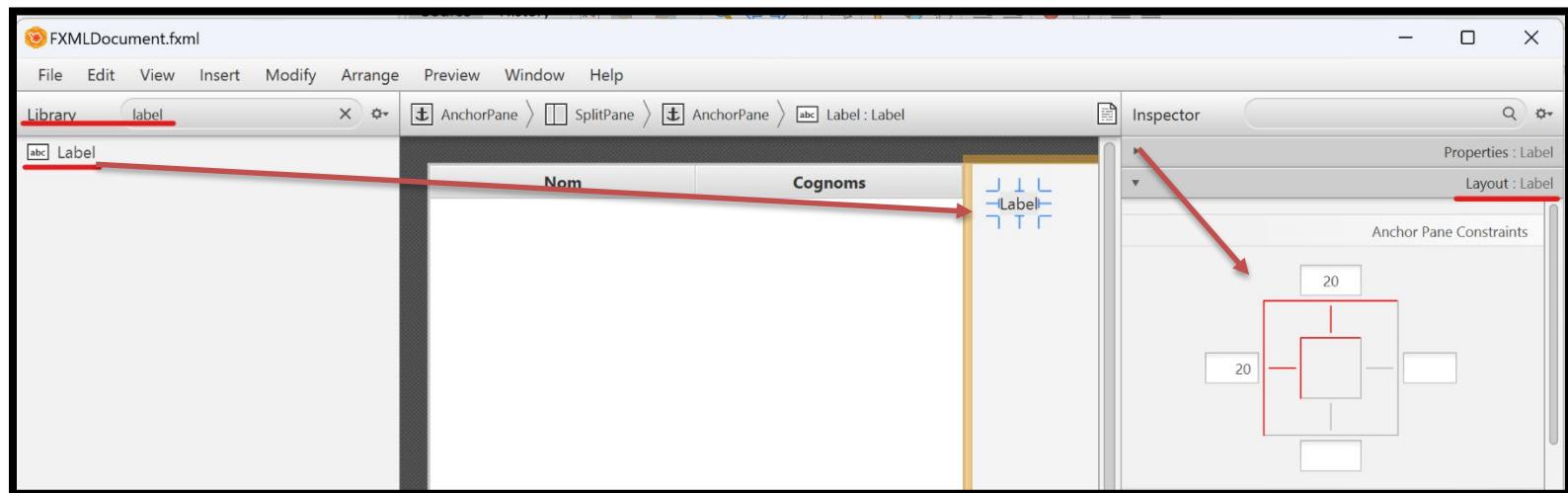
Seleccionem la **TableView** i en **Properties** busquem la propietat **Column Resize Policy** i triem l'opció “**constrained-resize**” per a assegurar que les columnes es repartiran i utilitzaran sempre tot l'espai que tinguen disponible.



### 3. CREACIÓ DE LA VISTA INICIAL

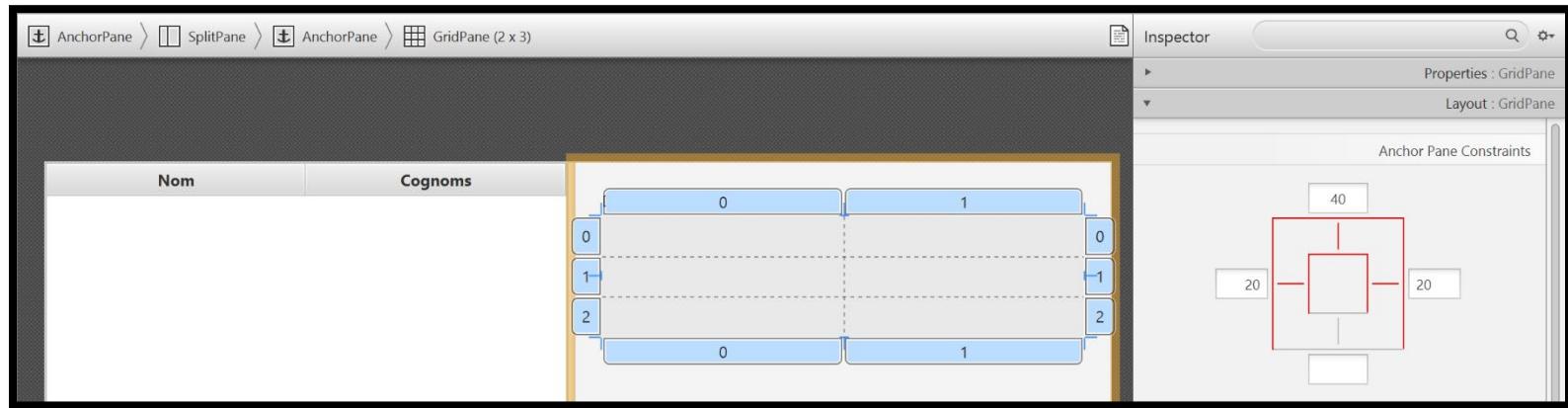
Afegim una **Label** al costat dret del SplitPane (truc: usa la cerca en la llibreria per a trobar el component Label).

Ajustem la seu aparença usant ancoratges i canviem el text a “Detalls de perfil”.



### 3. CREACIÓ DE LA VISTA INICIAL

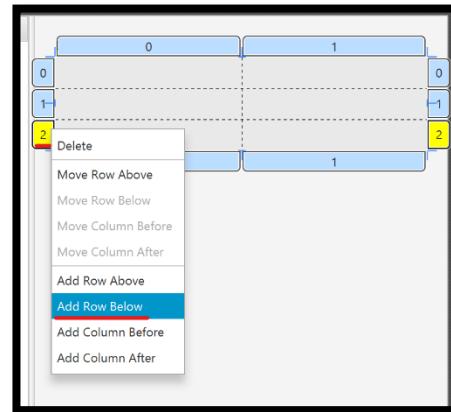
Afegim un GridPane al costat dret, el seleccionem i ajustem la seu aparença usant ancoratges (superior, dret i esquerre).



### 3. CREACIÓ DE LA VISTA INICIAL

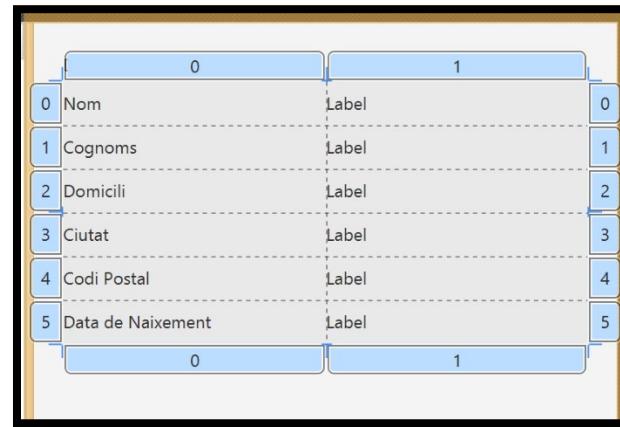
Afegim 3 files mes al *GridPane*. Deuen quedar 6 files en total (del 0 al 5).

Nota: Per a afegir una fila al *GridPane* seleccionem un número de fila existent (es tornarà de color groc), fem clic en el botó dret del ratolí sobre el número de fila i triem “**Add Row Below**”.



### 3. CREACIÓ DE LA VISTA INICIAL

Afegim les següents etiquetes (Label) a les cel·les del *GridPane* i posem els textos corresponents en les etiquetes de la part esquerra.



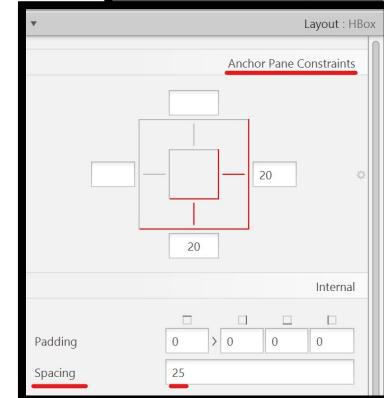
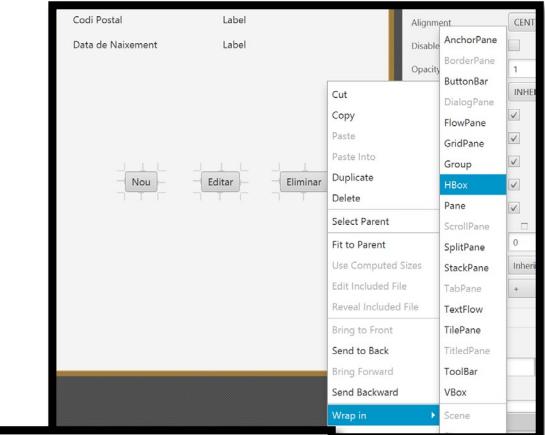
### 3. CREACIÓ DE LA VISTA INICIAL

Afegim 3 botons a la part inferior.

Mantenint el botó “Control” del teclat pressionat, els seleccionem tots.

A continuació, fem clic dret del ratolí i en el submenú seleccionem **Wrap In** i l'opció **HBox**. Això els agruparà als 3 junts en una caixa.

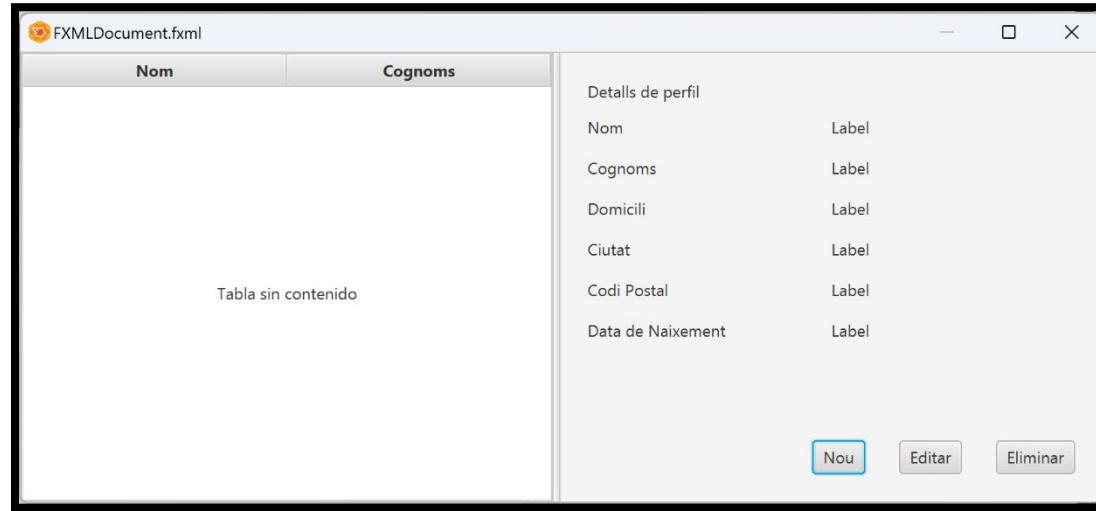
Per últim, establim un espaiat de 25 en **Layout -> Spacing** dins de l'HBox. Després, establim també ancoratges (dret i inferior) a 20 perquè es mantinguin en el lloc correcte.



### 3. CREACIÓ DE LA VISTA INICIAL

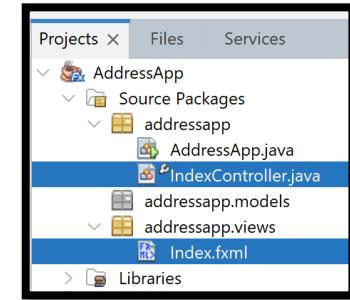
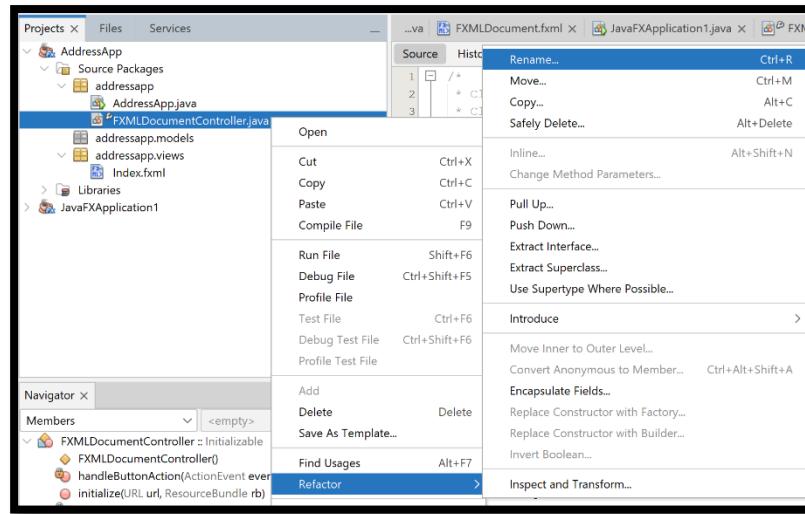
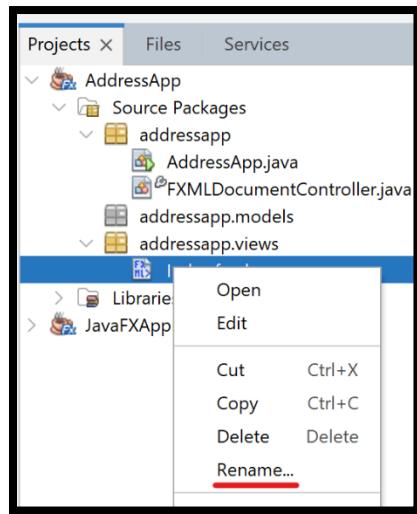
Ara es convenient guardar els canvis i usar el menú **Preview** per a comprovar el seu comportament en canviar la grandària de la finestra.

Veurem un disseny semblant al següent.



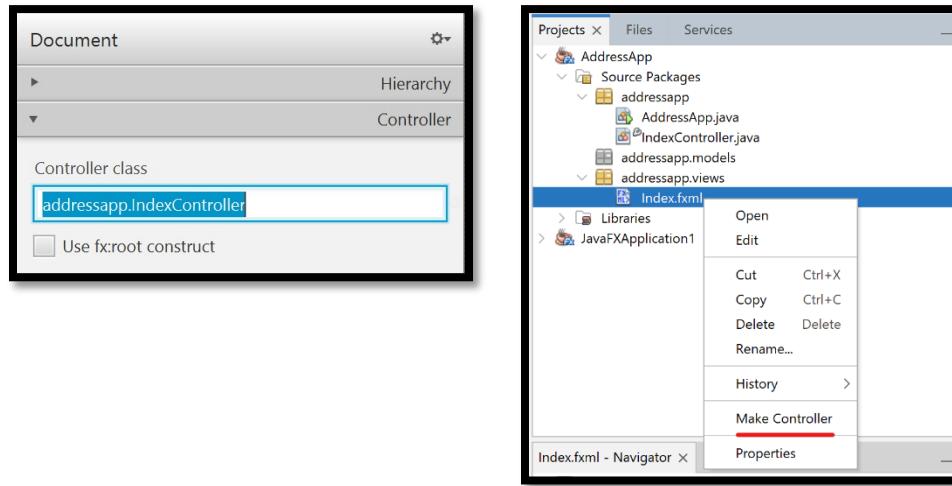
### 3. CREACIÓ DE LA VISTA INICIAL

Ara per a donar-li mes sentit als noms dels arxius de la aplicació anem a canviar el nom de la nostra vista i del nostre controlador



### 3. CREACIÓ DE LA VISTA INICIAL

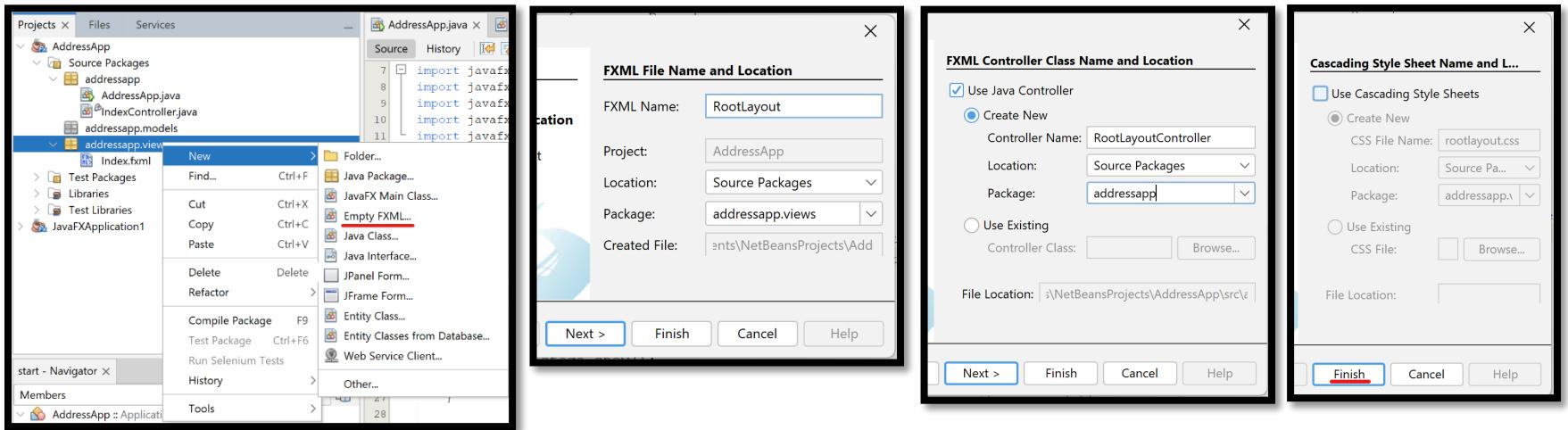
Hem de recordar-nos de modificar el nom en el controlador de la vista i recordem guardar la vista. Per últim, actualitzar els canvis amb el **Make Controller**.



## 4. CREACIÓ DE LA VISTA CONTROLADORA

Necessitem un altra vista (arxiu FXML) que s'utilitzarà com a vista principal. Esta contindrà una barra de menús i encapsularà la vista Index.fxml.

Crea un altre arxiu FXML dins del paquet **views** anomenat RootLayout.fxml.



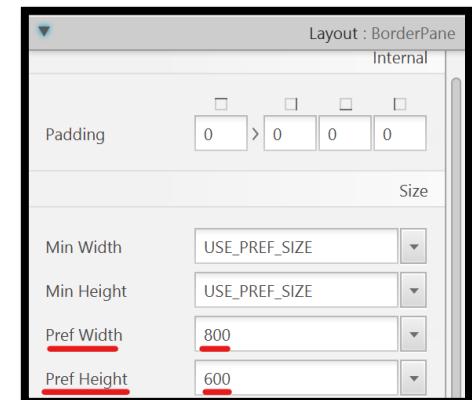
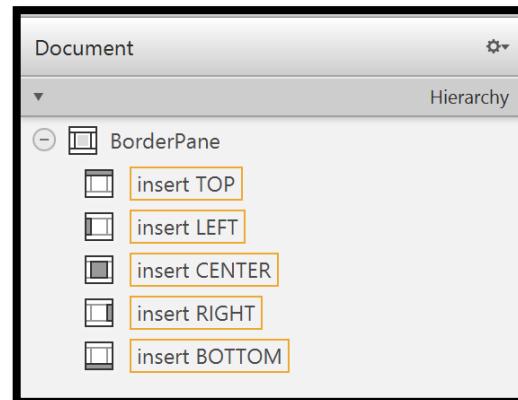
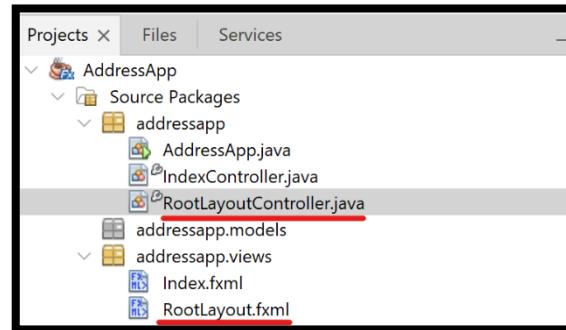
## 4. CREACIÓ DE LA VISTA CONTROLADORA

Haurem de veure la vista i el controlador de RootLayout.

Obrim el RootLayout.fxml

Esborrem el AnchorPane i afegeix el panell BorderPane.

Modifiquem la grandària amb la propietat "Pref Width" a 800 i "Pref Height" en 600.

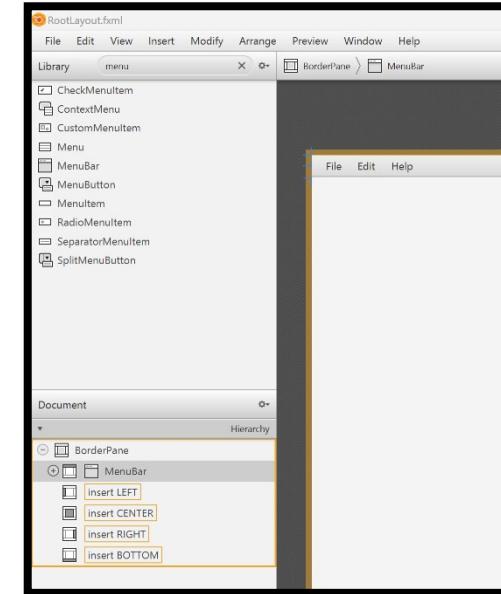
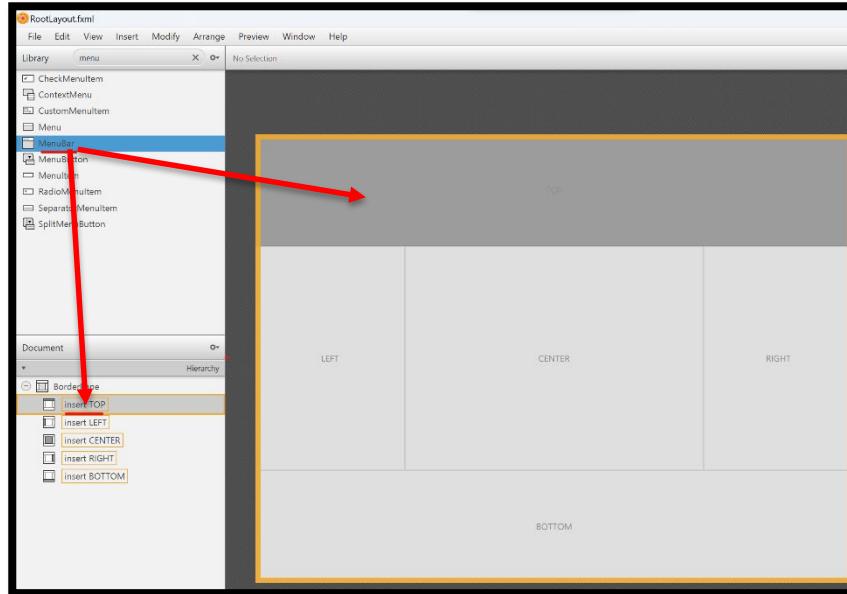


## 4. CREACIÓ DE LA VISTA CONTROLADORA

Afegim un **MenuBar** en la ranura superior de el **BorderPane** (insert TOP).

De moment no implementarem la funcionalitat del menú.

Hem de recordar-nos de modificar el nom en el controlador de la vista i recordem guardar la vista. Per últim, actualitzar els canvis amb el **Make Controller**.



## 5. LA CLASSE PRINCIPAL EN JavaFX

La classe generada (**UF12AddressApp.java**) estén a la classe **Application** i conté dos mètodes, main() i start().

El mètode main() és el mètode que s'executa inicialment. Aquest executa el mètode start(Stage stage) quan l'aplicació és llançada.

```
13  /* * @author guigar */  
14  public class AddressApp extends Application {  
15    
16  @Override  
17  public void start(Stage stage) throws Exception {  
18    
19    
20    
21    
22    
23    
24    
25    
26    
27    
28    
29    
30    
31    
32    
33    
34    
35    
36 }
```

Com es pot veure, el mètode start(...) pren un Stage com a paràmetre.

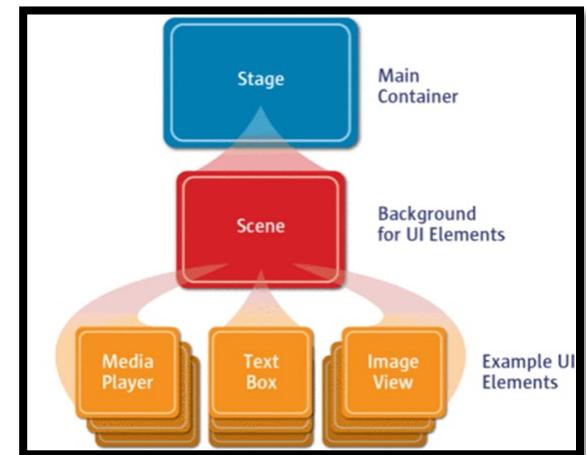
## 5. LA CLASSE PRINCIPAL EN JavaFX

El gràfic següent mostra l'estructura de qualsevol aplicació JavaFX:

És com una obra de teatre: El **Stage** (escenari) és el contenidor principal, normalment una finestra amb vora i els típics botons per a maximitzar, minimitzar o tancar la finestra.

Dins del **Stage** es pot afegir una **Scene** (escena), la qual pot canviar-se dinàmicament per una altra Scene.

Dins d'un Scene s'afegen els **nodos** JavaFX, com ara AnchorPane, TextBox, etc.



## 5. LA CLASSE PRINCIPAL EN JavaFX

Obrim l'arxiu [UF12AddressApp.java](#) i reemplacem el codi de la funció start() amb el codi següent:

```
14  /**
15  *
16  * @author GGarrido
17  */
18  public class AddressApp extends Application {
19      private Stage primaryStage;
20      private BorderPane rootLayout;
21
22      @Override
23      public void start(Stage stage) throws Exception {
24          //Assignem el primaryStage al stage inicial
25          this.primaryStage = stage;
26          //Canviem el títol.
27          //Heu de canviar Ggarrido el vostre nom i el primer cognom.
28          this.primaryStage.setTitle(value: "Activitat Avaluable 2 - GGarrido");
29
30          //La funcio initRootLayout inicialitza la Scene principal.
31          initRootLayout();
32
33          //La funcio showIndex inicialitza la Scene interna.
34          showIndex();
35      }
}
```

## 5. LA CLASSE PRINCIPAL EN JavaFX

Ara implementem el codi de initRootLayout

```
private void initRootLayout() {
    try{
        //Carreguem el FXML
        FXMLLoader loader = new FXMLLoader(
            url:(getClass().getResource(name: "views/RootLayout.fxml")));
        this.rootLayout = loader.load();
        //Creem una Scena amb el arxiu FXML
        Scene scene = new Scene(parent: this.rootLayout);
        //Assignem l'escena a l Stage.
        this.primaryStage.setScene(value: scene);
        //Mostrem el Stage
        this.primaryStage.show();
    }catch(IOException e){
        e.printStackTrace();
    }
}
```

## 5. LA CLASSE PRINCIPAL EN JavaFX

A continuació, implementem el codi de showIndex()

```
private void showIndex() {
    try{
        //Carreguem el FXML
        FXMLLoader loader = new FXMLLoader(
            url.getClass().getResource(name: "views/Index.fxml"));
        AnchorPane index = (AnchorPane) loader.load();
        this.rootLayout.setCenter(value: index);
    }catch(IOException e){
        e.printStackTrace();
    }
}
```

## 5. LA CLASSE PRINCIPAL EN JavaFX

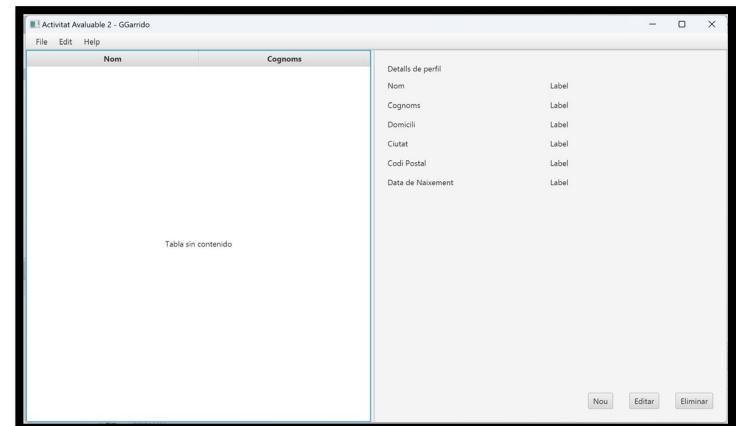
Per últim, netegem la aplicació per a que es creen els registres de la localització de les diferents vistes. Per a fer-ho anirem a Run -> Clean and Build Project.

Al finalitzar, tornarà un error de construcció, es normal perquè estem en un projecte de tipus ANT.

```
BUILD FAILED
E:\Users\quiga\Documents\NetBeansProjects\AddressApp\nbproject\jfx-impl.xml:3597: The following error occurred while executing this line:
C:\Users\quiga\Documents\NetBeansProjects\AddressApp\nbproject\jfx-impl.xml:1251: The following error occurred while executing this line:
C:\Users\quiga\Documents\NetBeansProjects\AddressApp\nbproject\jfx-impl.xml:1259: Unable to create javax script engine for javascript

Total time: 1 second
Result: 1
BUILD SUCCESSFUL (total time: 1 second)
```

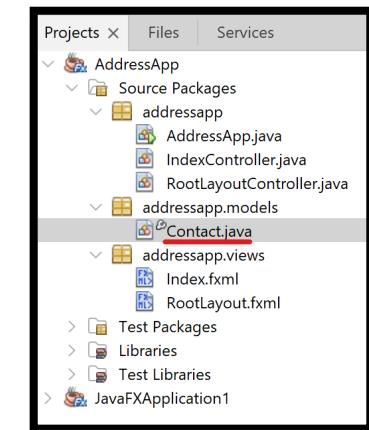
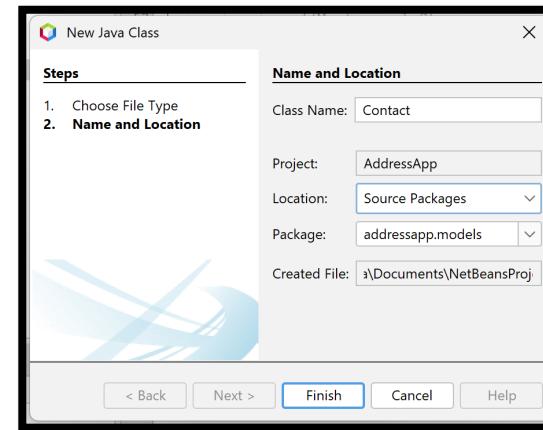
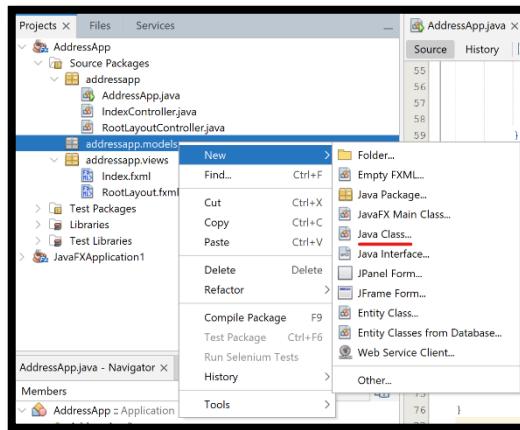
Ara provarem a executar (Run File) l'arxiu de la classe [UF12AddressApp.java](#).



## 6. ELS MODELS

Necessitem un model per a contindre la informació sobre els contactes de la nostra agenda. Afegim una nova classe al paquet encarregat de contindre els models (addressapp.model) denominada **Contact**.

La classe **Contact** tindrà atributs (instàncies de classe) per al nom, la direcció i la data de naixement.



## 6. ELS MODELS

Ara s'ha de crear les propietats del model.

Una Propietat permet, entre altres coses, rebre notificacions automàticament quan el valor d'una variable canvia (per exemple si canvia Cognoms).

Estes referencien a tots els atributs d'una classe.

Creem la classe amb els seus constructors. Un serà buit i l'altre amb variables.

```
5  package addressapp.models;
6  import java.time.LocalDate;
7  import javafx.beans.property.SimpleIntegerProperty;
8  import javafx.beans.property.SimpleObjectProperty;
9  import javafx.beans.property.SimpleStringProperty;
10 
11 /**
12  * 
13  * @author GGarrido
14  */
15 public class Contact {
16     private SimpleStringProperty nom;
17     private SimpleStringProperty cognoms;
18     private SimpleStringProperty domicili;
19     private SimpleStringProperty ciutat;
20     private SimpleIntegerProperty codi_postal;
21     private SimpleObjectProperty<LocalDate> data_de_naixement;
22 
23     public Contact(String nom, String cognoms, String domicili, String ciutat,
24         int cp, int dia, int mes, int any) {
25         this.nom = new SimpleStringProperty(string: nom);
26         this.cognoms = new SimpleStringProperty(string: cognoms);
27         this.domicili = new SimpleStringProperty(string: domicili);
28         this.ciutat = new SimpleStringProperty(string: ciutat);
29         this.codi_postal = new SimpleIntegerProperty(i: cp);
30         this.data_de_naixement = new SimpleObjectProperty<>(t: LocalDate.of(year: any, month: mes, dayOfMonth: dia));
31     }
32 
33     public Contact() {
34         this.nom = new SimpleStringProperty(string: "");
35         this.cognoms = new SimpleStringProperty(string: "");
36         this.domicili = new SimpleStringProperty(string: "");
37         this.ciutat = new SimpleStringProperty(string: "");
38         this.codi_postal = new SimpleIntegerProperty(i: -1);
39         this.data_de_naixement = new SimpleObjectProperty<>(t: null);
40     }
41 }
```

## 6. ELS MODELS

Ara creem els Getters i Setters de tots els atributs:

```
43     public SimpleStringProperty getNom() {
44         return nom;
45     }
46
47     public void setNom(SimpleStringProperty nom) {
48         this.nom = nom;
49     }
50
51     public SimpleStringProperty getCognoms() {
52         return cognoms;
53     }
54
55     public void setCognoms(SimpleStringProperty cognoms) {
56         this.cognoms = cognoms;
57     }
58
59     public SimpleStringProperty getDomicili() {
60         return domicili;
61     }
62
63     public void setDomicili(SimpleStringProperty domicili) {
64         this.domicili = domicili;
65     }
```

```
67     public SimpleStringProperty getCiutat() {
68         return ciutat;
69     }
70
71     public void setCiutat(SimpleStringProperty ciutat) {
72         this.ciutat = ciutat;
73     }
74
75     public SimpleIntegerProperty getCodi_postal() {
76         return codi_postal;
77     }
78
79     public void setCodi_postal(SimpleIntegerProperty codi_postal) {
80         this.codi_postal = codi_postal;
81     }
82
83     public SimpleObjectProperty<LocalDate> getData_de_naixement() {
84         return data_de_naixement;
85     }
86
87     public void setData_de_naixement(SimpleObjectProperty<LocalDate>
88                                         data_de_naixement) {
89         this.data_de_naixement = data_de_naixement;
90     }
```

## 6. ELS MODELS

Les principals dades que maneja la nostra aplicació són una col·lecció de contactes.

Crearem una llista d'objectes de tipus Contact dins de la classe principal [UF12AddressApp](#).

La resta de controladors obtindrà després accés a aquesta llista central dins d'[UF12AddressApp](#).

Per a crear aquest tipus de llista de contactes utilitzarem una llista observable (`ObservableList`) que és un tipus de llista que permeten avisar a les classes gràfiques de JavaFX sobre els canvis que es produeixen en la mateixa. D'aquesta manera la vista se sincronitzarà amb les dades.

## 6. ELS MODELS

Per a crear una nova ObservableList, afegim el codi següent al principi de la classe **UF12AddressApp**.

```
5  package addressapp;
6
7  import addressapp.models.Contact;
8  import java.io.IOException;
9
10 import javafx.application.Application;
11 import javafx.collections.FXCollections;
12 import javafx.collections.ObservableList;
13 import javafx.fxml.FXMLLoader;
14 import javafx.scene.Scene;
15 import javafx.scene.layout.AnchorPane;
16 import javafx.scene.layout.BorderPane;
17 import javafx.stage.Stage;
18
19 /**
20 *
21 * @author GGarrido
22 */
23 public class AddressApp extends Application {
24     private Stage primaryStage;
25     private BorderPane rootLayout;
26
27     private ObservableList<Contact> contactes =
28         FXCollections.observableArrayList();
```

## 6. ELS MODELS

També, afegirem un **constructor** ([UF12AddressApp](#)) per a crear dades d'exemple i un mètode públic de consulta de la llista ObservableList que simula el mètode get() anomenat **getContactes()**.

```
28
29
30     private ObservableList<Contact> contactes =
31         FXCollections.observableArrayList();
32
33     public AddressApp() {
34         this.contactes.add(new Contact(nom:"Guillermo", cognoms:"Garrido Portes",
35             domicili: "C/Albacete",
36             ciutat: "Valencia", cp: 47001, dia:11, mes:01, any:1995));
37         this.contactes.add(new Contact(nom:"Maria", cognoms:"Gómez Gil",
38             domicili: "C/Alzira",
39             ciutat: "Alacant", cp: 47002, dia:21, mes:02, any:2000));
40         this.contactes.add(new Contact(nom:"Diego", cognoms:"Gonzalez Cuenca",
41             domicili: "C/Manises",
42             ciutat: "Castello", cp: 47003, dia:31, mes:03, any:2005));
43         this.contactes.add(new Contact(nom:"Laura", cognoms:"Galiana Gutiérrez",
44             domicili: "C/Xativa",
45             ciutat: "Barcelona", cp: 47004, dia:01, mes:04, any:2010));
46         this.contactes.add(new Contact(nom:"Silvia", cognoms:"Gandia García",
47             domicili: "Plaza la Reina",
48             ciutat: "Valencia", cp: 47005, dia:12, mes:05, any:2015));
49     }
50
51     public ObservableList<Contact> getContactes(){
52         return this.contactes;
53     }
54 }
```

## 7. CONTROLADOR DE LA VISTA Index

Afegirem alguns atributs per a accedir a la taula i les etiquetes de la vista.

Els camps i mètodes on la vista necessita accès, aniran precedits per l'anotació @FXML.

Esta és necessària perquè la vista tinga accés als atributs i mètodes del controlador, fins i tot encara que siguen privats.

Una vegada definida la vista en fxml, l'aplicació s'encarregarà d'emplenar automàticament aquests atributs en carregar el fxml.

## 7. CONTROLADOR DE LA VISTA Index

Obrirem l'arxiu `IndexController.java` i afegirem el següent codi creant variables per a cada element de la interfície que volem modificar o utilitzar.

També, incloureml una variable per a la classe principal `UF12AddressApp`.

```
/*
 *
 * @author GGarrido
 */
public class IndexController implements Initializable {
    //Variables de la tabla
    @FXML
    private TableView<Contact> contact_table;
    @FXML
    private TableColumn<Contact, String> nom_column;
    @FXML
    private TableColumn<Contact, String> cognoms_column;
    //Variables de la vista de detalls
    @FXML
    private Label nom_label;
    @FXML
    private Label cognoms_label;
    @FXML
    private Label domicili_label;
    @FXML
    private Label ciutat_label;
    @FXML
    private Label codi_postal_label;
    @FXML
    private Label data_de_naixement_label;

    //Variable per a la clase principal
    private AddressApp address_app;
```

## 7. CONTROLADOR DE LA VISTA Index

A continuació, implementarem el mètode initialize() que és invocat automàticament després de carregar el fxml.

En setCellValueFactory(...) s'encarrega d'unificar el valor de cada cel·la amb la propietat de l'objecte. Per tant, per a cada objecte que li passem a la taula en la cel·la indicada assignarà el que retorna la funció get() que li indiquem.

```
@Override
public void initialize(URL url, ResourceBundle rb) {
    nom_column.setCellValueFactory(cellData->cellData.getValue().getNom());
    cognoms_column.setCellValueFactory(cellData->cellData.getValue()
        .getCognoms());
}

/**
 * Se obte, des de la classe principal, el array de contactes.
 * @param address_app
 */
public void setAddressApp(AddressApp address_app){
    this.address_app = address_app;
    contact_table.setItems(value: address_app.getContactes());
}
```

## 8. CONNECTAR UF12AddressApp AMB EL CONTROLADOR DE LA VISTA Index

El llistat de contactes han de ser creat en la classe principal per a poder ser utilitzats i modificats des de els diferents controladors que puguen existir en l'aplicació.

Per a poder fer aquesta connexió, el que es fa es:

- Generar el controlador en la classe principal
- Cridar a la funció `setAddressApp()` del controlador, passant-li al controlador la instància de la pròpia classe.
- D'aquesta manera el controlador, en la seua funció `setAddressApp()`, pot inicialitzar el llistat de contactes fent ús de la funció `getContactes()` de la instància de la classe principal.

## 8. CONNECTAR UF12AddressApp AMB EL CONTROLADOR DE LA VISTA Index

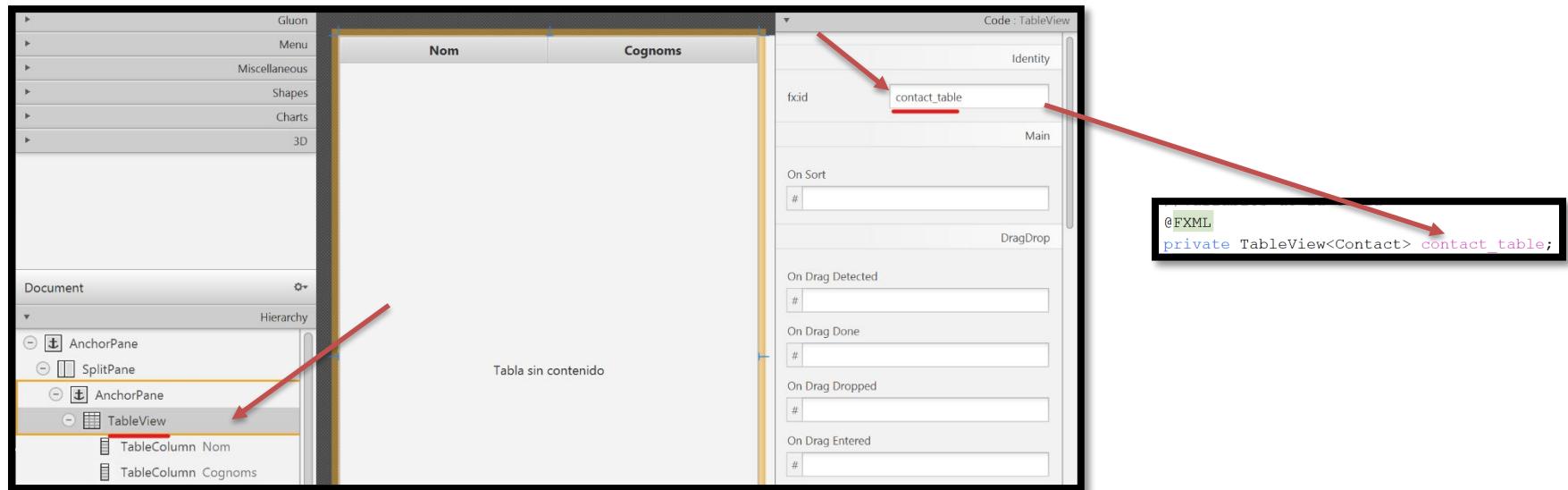
Per tant, modificarem la funció showIndex() de la classe AddressApp.

```
private void showIndex() {
    try{
        //Carreguem el FXML
        FXMLLoader loader = new FXMLLoader(
            url:getClass().getResource(name: "views/Index.fxml"));
        AnchorPane index = (AnchorPane) loader.load();
        this.rootLayout.setCenter(value: index);

        IndexController controller = loader.getController();
        controller.setAddressApp(address app:this);
    }catch(IOException e){
        e.printStackTrace();
    }
}
```

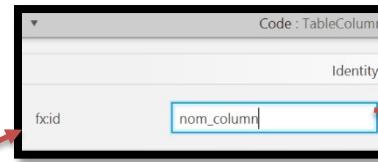
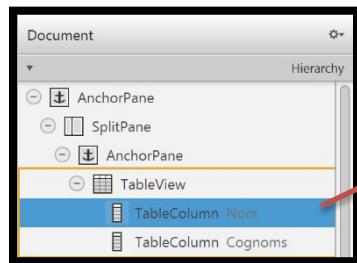
## 9. VINCULAR LES VARIABLES DEL CONTROLADOR DE LA VISTA Index AMB ELS SEUS ELEMENTS

Seleccionem **TableView** en la secció Hierarchy i en l'apartat Code escrivim **contact\_table** en la propietat **fx:id**.

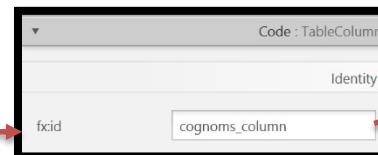
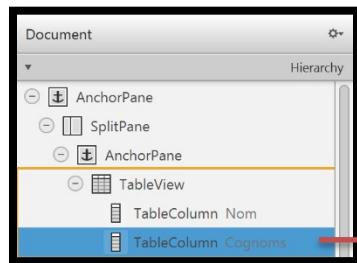


## 9. VINCULAR LES VARIABLES DEL CONTROLADOR DE LA VISTA Index AMB ELS SEUS ELEMENTS

Seleccionem les  **TableColumn** en la secció Hierarchy i fem el mateix amb els id's,



```
@FXML  
private TableColumn<Contact, String> nom_column;  
  
@FXML  
private TableColumn<Contact, String> cognoms_column;
```



## 9. VINCULAR LES VARIABLES DEL CONTROLADOR DE LA VISTA Index AMB ELS SEUS ELEMENTS

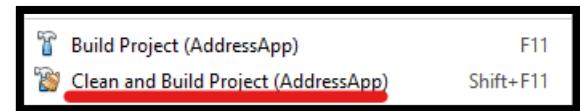
Per a cada etiqueta en la segona columna, introduïm el fx:id que corresponga.

The screenshot shows the JavaFX Scene Builder interface. The main area displays a GridPane with 2 columns and 6 rows. The first column contains labels for 'Nom', 'Cognoms', 'Domicili', 'Ciutat', 'Codi Postal', and 'Data de Naixement'. The second column contains empty labels. The 'Inspector' panel on the right shows the first label in the second column has an fx:id of 'nom\_label'. A red arrow points from the 'nom\_label' entry in the Inspector to the 'Assigned fxid' table in the bottom left, which also lists 'nom\_label'.

fxid	Component
contact_table	TableView
nom_column	TableColumn
cognoms_column	TableColumn
nom_label	Label
cognoms_label	Label
domicili_label	Label
ciutat_label	Label
codi_postal_label	Label
data_de_naixement_label	Label

```
@FXML  
private Label nom_label;
```

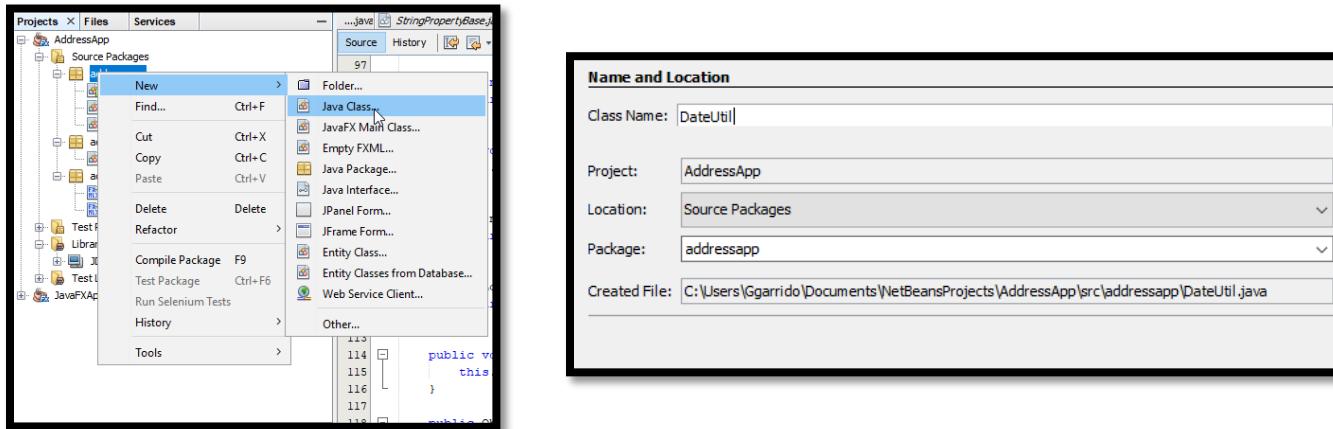
**IMPORTANT:** Si després de guardar els canvis, a l'executar dona algun error provarem a netejar el projecte.



## 10. LA CLASSE DateUtil

Per a poder treballar i mostrar la data de naixement que és un LocalDate necessitem comptar amb una classe que tinga alguns mètodes que ens ajuden a mostrar correctament les dates.

Per aquest motiu crearem en el nostre paquet [UF12AddressApp](#) una classe anomenada DateUtil.java



## 10. LA CLASSE DateUtil

Primer crearem les constants per a donar format a les dates.

Després, crearem tres mètodes estàtics per a convertir de LocalDate a String, String a LocalDate i validar dates.

```
public class DateUtil {  
    private static final String PATRO_DATA = "dd.MM.yyyy";  
    private static final DateTimeFormatter FORMATEJADOR_DATA =  
        DateTimeFormatter.ofPattern(pattern: PATRO_DATA);  
  
    /**  
     * Torna la data en format de cadena o null en cas de no existir  
     * @param data  
     * @return  
     */  
    public static String format(LocalDate data){  
        if(data == null)  
            return null;  
        return FORMATEJADOR_DATA.format(temporal:data);  
    }  
    /**  
     * Torna la data en format de LocalDate o null en cas de no existir  
     * @param data  
     * @return  
     */  
    public static LocalDate parse(String data){  
        //En cas de que la variable no siga una data vàlida  
        try{  
            return FORMATEJADOR_DATA.parse(text:data, LocalDate::from);  
        }catch(DateTimeParseException e){  
            return null;  
        }  
    }  
    /**  
     * Comproba que es vàlida la data  
     * @param data  
     * @return  
     */  
    public static boolean validDate(String data){  
        //En cas de que la variable no siga una data valida  
        return DateUtil.parse(data) != null;  
    }  
}
```

## 11. EMPLENAR ELS DETALLS DE LA VISTA I ACTUALITZAR-LOS

Afegim un nou mètode dins de IndexController que ens ajude a emplenar les etiquetes amb les dades d'una sola persona.

Creem un mètode anomenat showContactDetails(). Aquest mètode establirà el text de les etiquetes amb el valor de la propietat corresponent del contacte.

Si el paràmetre d'entrada es null eliminarà el text de totes les etiquetes.

```
public void showContactDetails(Contact contacte){  
    if(contacte != null){  
        this.nom_label.setText(value: contacte.getNom().get());  
        this.cognoms_label.setText(value: contacte.getCognoms().get());  
        this.domicili_label.setText(value: contacte.getDomicili().get());  
        this.ciutat_label.setText(value: contacte.getCiutat().get());  
        this.codí_postal_label.setText(value: Integer.toString(  
            i: contacte.getCodi_postal().get()));  
        this.data_de_naixement_label.setText(value: DateUtil.format(  
            data: contacte.getData_de_naixement().get()));  
    }else{  
        this.nom_label.setText(value: "");  
        this.cognoms_label.setText(value: "");  
        this.domicili_label.setText(value: "");  
        this.ciutat_label.setText(value: "");  
        this.codí_postal_label.setText(value: "");  
        this.data_de_naixement_label.setText(value: "");  
    }  
}
```

## 11. EMPLENAR ELS DETALLS DE LA VISTA I ACTUALITZAR-LOS

Per a assabentar-se que l'usuari ha seleccionat a un contacte en la taula de contactes, necessitem escoltar els canvis.

Això s'aconsegueix mitjançant la implementació d'un interface de JavaFX que es diu ChangeListener amb el mètode anomenat addListener().

Aquest mètode només té tres paràmetres: observable, oldValue, i newValue.

# 11. EMPLENAR ELS DETALLS DE LA VISTA I ACTUALITZAR-LOS

En el mètode initialize() de IndexController afegirem les següents línies:

```
@Override  
public void initialize(URL url, ResourceBundle rb) {  
    nom_column.setCellValueFactory(cellData->cellData.getValue().getNom());  
    cognoms_column.setCellValueFactory(cellData->cellData.getValue()  
        .getCognoms());  
  
    //Al carregar la vista esborrem les dades  
    showContactDetails(contacte: null);  
  
    /* Creem un escoltador per a la taula que quan es seleccione un element  
    de la mateixa, cride a la funció showContactDetails() passant-li el nou  
    contacte*/  
    contact_table.getSelectionModel().selectedItemProperty().addListener(  
        (observable, old_value, new_value) -> showContactDetails(contacte: new_value)  
    );  
}
```

A hores d'ara hauríem de ser capaços de veure la nostra aplicació en funcionament i veure com canvia segons premem a un contacte o un altre.

Activitat Available 2 - GGarrido		
		File Edit Help
Nom	Cognoms	Detalls de perfil
Guillermo	Garrido Portes	Guillermo
Maria	Gómez Gil	
Diego	González Cuenca	Garrido Portes
Laura	Galiana Gutiérrez	C/Albacete
Silvia	Gandia García	Valencia
		47001
		11.01.1995

## 12. EVENTS I US DE FUNCIONS DES DE LA VISTA

La nostra interfície d'usuari ja conté un botó d'esborrar, però sense funcionalitat.

Podem seleccionar l'acció a executar en prémer un botó des del Scene Builder.

Qualsevol mètode del nostre controlador anotat amb @FXML (o declarat com public) és accessible des de Scene Builder.

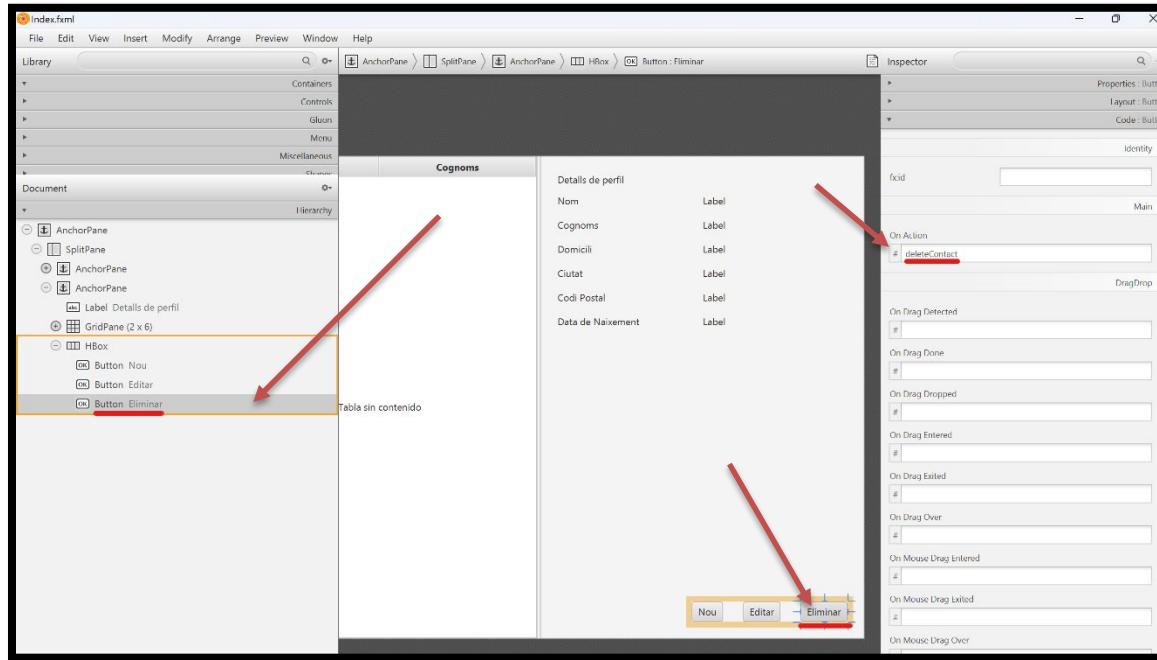
## 12. EVENTS I US DE FUNCIONS DES DE LA VISTA

Afegim el mètode d'esborrat al final de la nostra classe IndexController.

```
/**  
 * Elimina un item de la taula de contactes.  
 */  
@FXML  
public void deleteContact(){  
    if(!contact_table.getItems().isEmpty()) {  
        int selected_index = contact_table.getSelectionModel()  
            .getSelectedIndex();  
        contact_table.getItems().remove(index: selected_index);  
    }  
}
```

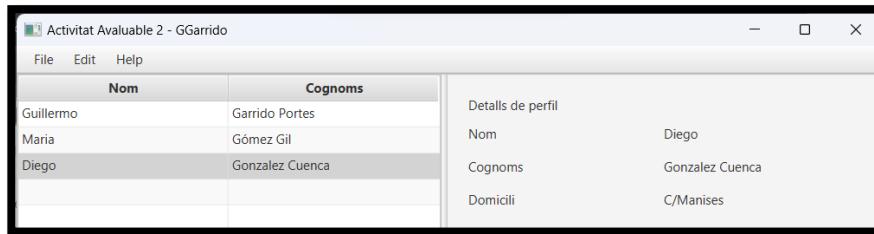
## 12. EVENTS I US DE FUNCIONS DES DE LA VISTA

Assignem la funció deleteContact() al botó en el menú denominat **On Action** en la vista **Index.fxml**



## 12. EVENTS I US DE FUNCIONS DES DE LA VISTA

Si després de guardar (i netejar la aplicació si es necessari) executem la aplicació podem veure com el boto Eliminar elimina el item que tenim seleccionat.



Però, que ocorre si no tenim cap element seleccionat?

```
Caused by: java.lang.IndexOutOfBoundsException: Index -1 out of bounds for length 5
    at java.base/jdk.internal.util.Preconditions.outOfBounds(Preconditions.java:100)
    at java.base/jdk.internal.util.Preconditions.outOfBoundsCheckIndex(Preconditions.java:106)
    at java.base/jdk.internal.util.Preconditions.checkIndex(Preconditions.java:302)
    at java.base/java.util.Objects.checkIndex(Objects.java:385)
    at java.base/java.util.ArrayList.remove(ArrayList.java:504)
    at javafx.base/com.sun.javafx.collections.ObservableListWrapper.doRemove(ObservableListWrapper.java:116)
    at javafx.base/javafx.collections.ModifiableObservableListBase.remove(ModifiableObservableListBase.java:186)
    at addressapp.IndexController.deleteContact(IndexController.java:103)
    at java.base/jdk.internal.reflect.DirectMethodHandleAccessor.invoke(DirectMethodHandleAccessor.java:104)
    ... 57 more
```

## 12. EVENTS I US DE FUNCIONS DES DE LA VISTA

Mètode `getSelectedIndex()` retorna -1 si no hi ha res seleccionat. Pel que al intentar cridar a la funció `remove()` amb index -1 mostra el error anterior.

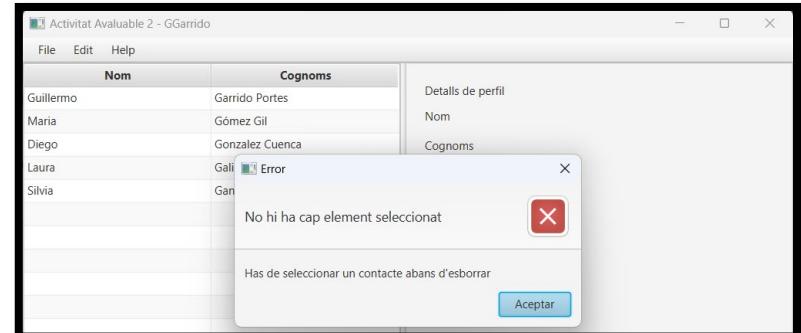
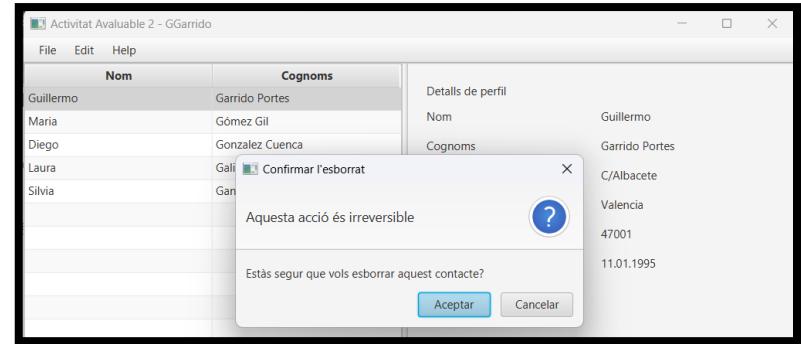
Aquest error es pot previndre afegint unes alertes per a evitar que es puga prémer a eliminar si no s'ha seleccionat cap persona.

També, hauríem de mostrar una alerta en cas que es vaja a produir l'esborrat per a confirmar el mateix.

## 12. EVENTS I US DE FUNCIONS DES DE LA VISTA

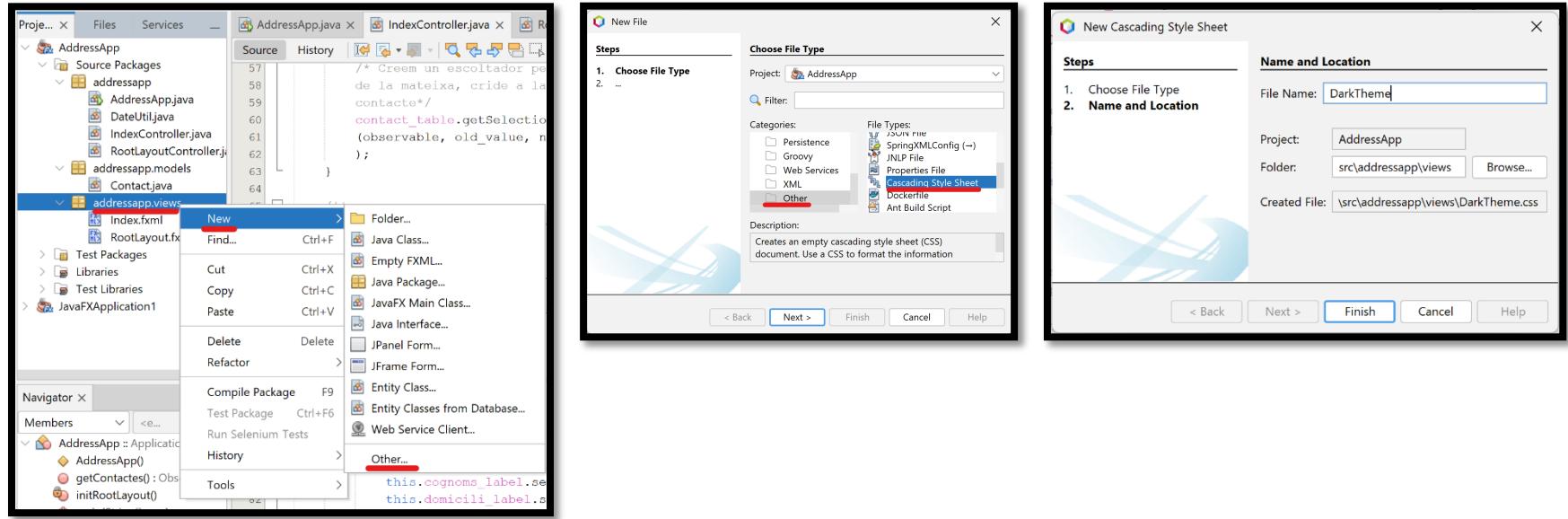
Ampliem el mètode deleteContact() afegint un dialeg per a validar el esborrat.

```
/**  
 * Elimina un item de la taula de contactes.  
 */  
@FXML  
public void deleteContact() {  
    Alert alert;  
    int selected_index = contact_table.getSelectionModel()  
        .getSelectedIndex();  
    if(!contact_table.getItems().isEmpty() && selected_index != -1){  
        alert = new Alert(at: Alert.AlertType.CONFIRMATION);  
        alert.setTitle(title: "Confirmar l'esborrat");  
        alert.setHeaderText(headerText: "Aquesta acció és irreversible");  
        alert.setContentText(  
            contentText:"Estàs segur que vols esborrar aquest contacte?");  
        Optional<ButtonType> result = alert.showAndWait();  
        if(result.get() == ButtonType.OK){  
            contact_table.getItems().remove(index: selected_index);  
        }  
    }else{  
        alert = new Alert(at: Alert.AlertType.ERROR);  
        alert.setTitle(title: "Error");  
        alert.setHeaderText(headerText: "No hi ha cap element seleccionat");  
        alert.setContentText(  
            contentText:"Has de seleccionar un contacte abans d'esborrar");  
        alert.showAndWait();  
    }  
}
```



## 13. ESTILS MIJANÇANT CSS

Ara que tot comença a funcionar li donarem estils perquè siga mes visual.  
En primer lloc crearem un nou fitxer DarkTheme.css en el paquet addressapp.views



# 13. ESTILS MIJANÇANT CSS

Afegim en el arxiu el següent codi:

```
/*
    Author      : GGarrido
*/
.background {
    -fx-background-color: #1d1d1d;
}

.label {
    -fx-font-size: 11pt;
    -fx-font-family: "Segoe UI Semibold";
    -fx-text-fill: white;
    -fx-opacity: 0.6;
}

.label-bright {
    -fx-font-size: 11pt;
    -fx-font-family: "Segoe UI Semibold";
    -fx-text-fill: white;
    -fx-opacity: 1;
}

.label-header {
    -fx-font-size: 32pt;
    -fx-font-family: "Segoe UI Light";
    -fx-text-fill: white;
    -fx-opacity: 1;
}

.table-view {
    -fx-base: #1d1d1d;
    -fx-control-inner-background: #1d1d1d;
    -fx-background-color: #1d1d1d;
    -fx-table-cell-border-color: transparent;
    -fx-table-header-border-color: transparent;
    -fx-padding: 5;
}
```

```
.table-view .column-header-background {
    -fx-background-color: transparent;
}

.table-view .column-header, .table-view .filler {
    -fx-size: 35;
    -fx-border-width: 0 0 1 0;
    -fx-background-color: transparent;
    -fx-border-color:
        transparent
        transparent
        derive(-fx-base, 80%)
        transparent;
    -fx-border-insets: 0 10 1 0;
}

.table-view .column-header .label {
    -fx-font-size: 20pt;
    -fx-font-family: "Segoe UI Light";
    -fx-text-fill: white;
    -fx-alignment: center-left;
    -fx-opacity: 1;
}

.table-view:focused .table-row-cell:filled:focused:selected {
    -fx-background-color: -fx-focus-color;
}

.split-pane:horizontal > .split-pane-divider {
    -fx-border-color: transparent #1d1d1d transparent #1d1d1d;
    -fx-background-color: transparent, derive(#1d1d1d, 20%);
}

.split-pane {
    -fx-padding: 1 0 0 0;
}

.menu-bar {
    -fx-background-color: derive(#1d1d1d, 20%);
}
```

```
.context-menu {
    -fx-background-color: derive(#1d1d1d, 50%);
}

.menu-bar .label {
    -fx-font-size: 14pt;
    -fx-font-family: "Segoe UI Light";
    -fx-text-fill: white;
    -fx-opacity: 0.9;
}

.menu .left-container {
    -fx-background-color: black;
}

.text-field {
    -fx-font-size: 12pt;
    -fx-font-family: "Segoe UI Semibold";
}

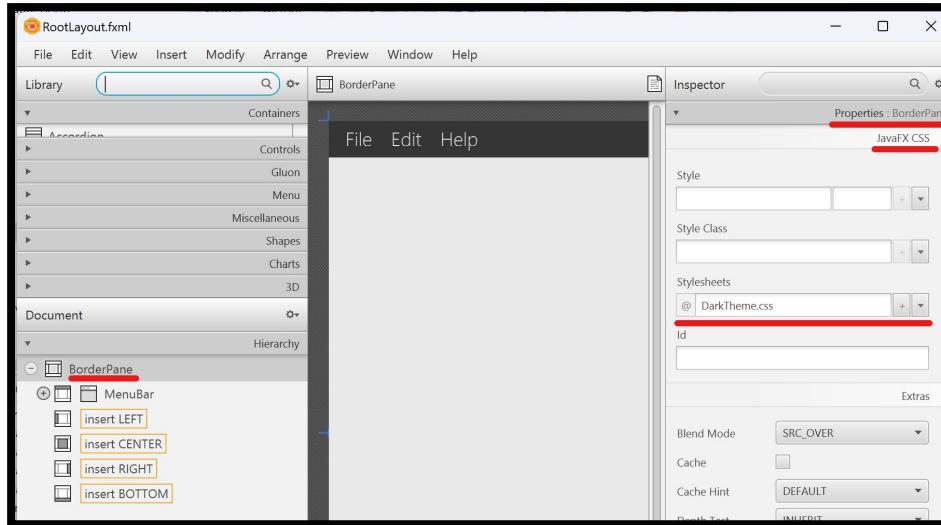
.button {
    -fx-padding: 5 22 5 22;
    -fx-border-color: #eze2e2;
    -fx-border-width: 2;
    -fx-background-radius: 0;
    -fx-background-color: #1d1d1d;
    -fx-font-family: "Segoe UI", Helvetica, Arial, sans-serif;
    -fx-font-size: 11pt;
    -fx-text-fill: #d8d8d8;
    -fx-background-insets: 0 0 0 0, 0, 1, 2;
}

.button:hover {
    -fx-background-color: #3a3a3a;
}

.button:pressed, .button:default:hover:pressed {
    -fx-background-color: white;
    -fx-text-fill: #1d1d1d;
}
```

## 13. ESTILS MIJANÇANT CSS

Seleccionem el BorderPane de RootLayout en la secció Hierarchy i en la vista Properties afegim en **Stylesheets** l'arxiu DarkTheme.css.

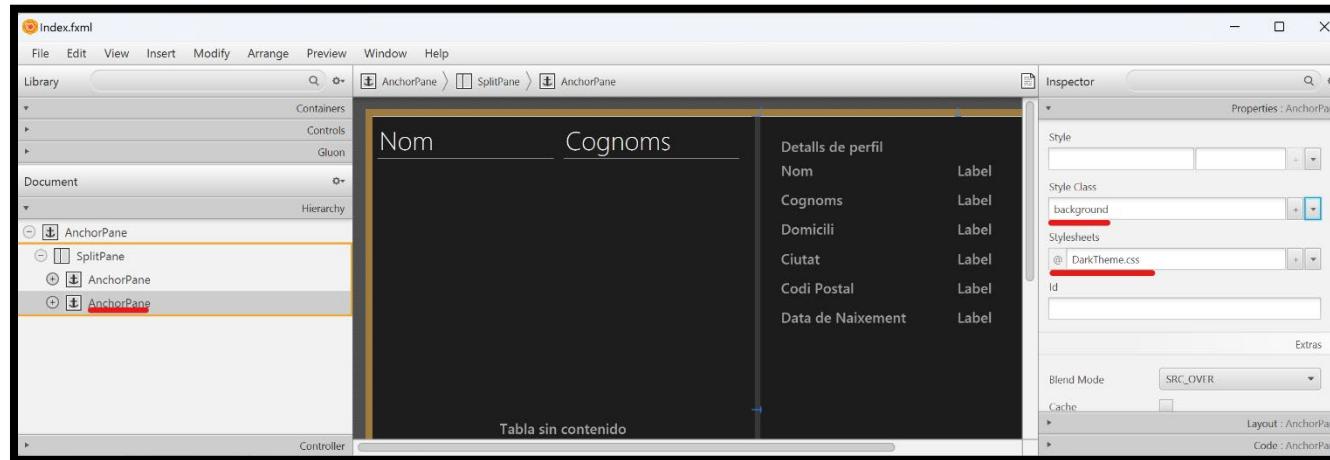


Fem el mateix amb el AnchorPane de Index.fxml.

## 13. ESTILS MIJANÇANT CSS

Seleccionem el panell AnchorPane de la dreta, dins del SplitPane de Index.fxml.

En este cas a més de asignar-li el full de estils, anem a asignar-li una classe. Per a fer-ho en la vista Properties i triem background com a classe d'estil. El fons haurà de tornar-se negre.



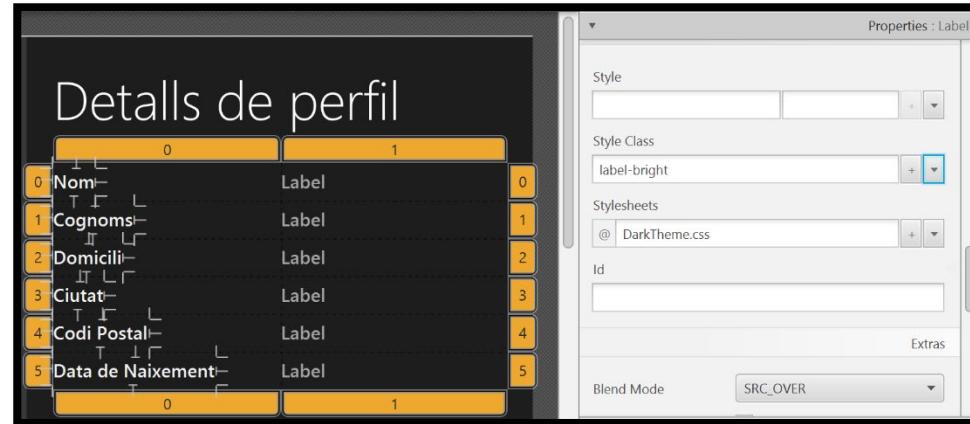
## 13. ESTILS MIJANÇANT CSS

En aquest moment, totes les etiquetes en el costat dret tenen la mateixa grandària.

Seleccionem l'etiqueta “Detalls de perfil” i afegim **label-header** com a classe d'estil.

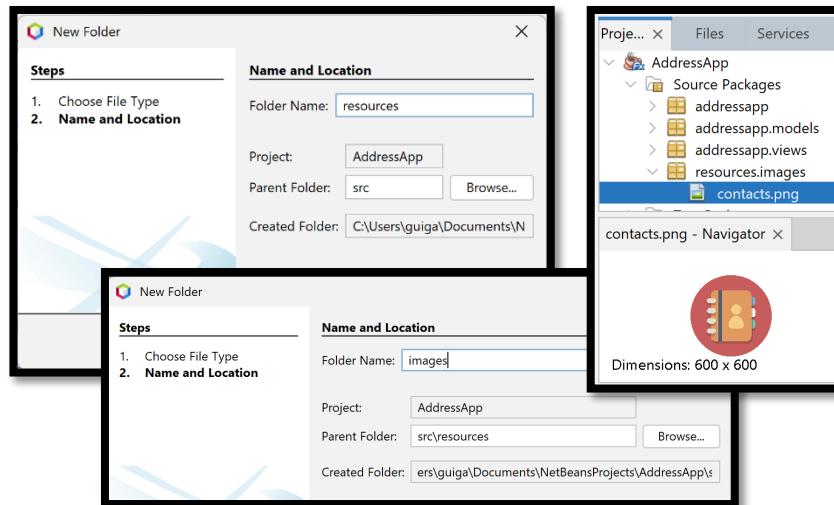
I a les etiquetes de la columna esquerra assignarem la classe **label-bright**.

Després, haurem d'adaptar la vista perquè no és sobreposen els elements.



## 14. ICONA DE L'APLICACIÓ

Per a establir la icona de la nostra escena hem de crear una carpeta en src/resources/images i dins de la carpeta afegirem la nostra imatge. Després hem d'assignar-li-la a la Stage en el mètode start() de UF12AddressApp.java.



```
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

@Override
public void start(Stage stage) throws Exception {
    //Assignem el primaryStage al stage inicial
    this.primaryStage = stage;
    //Canviem el títol.
    //Heu de canviar Ggarrido el vostre nom i el primer cognom.
    this.primaryStage.setTitle(value: "Activitat Avaluable 2 - GGarrido");

    Image icona = new Image(string: "resources/images/contacts.png");
    this.primaryStage.getIcons().add(e: icona);
}
```

## 14. PERSISTÈNCIA DE DADES

Actualment l'aplicació funciona de la següent manera:

- L'aplicació s'inicia amb l'execució del mètode `main(...)` de la classe `UF12AddressApp`.
- El constructor public `UF12AddressApp()` és invocat i afeg algunes dades d'exemple.
- El mètode `start(...)` de la classe `UF12AddressApp` és invocat, el qual al seu temps invoca a `initRootLayout()` per a inicialitzar la vista principal utilitzant l'arxiu `RootLayout.fxml`. L'arxiu FXML té informació sobre quin controlador utilitzar i enllaça la vista amb el seu controlador `RootLayoutController`.

## 14. PERSISTÈNCIA DE DADES

Per a poder guardar les dades des de les opcions del menú, hem de poder comunicar-nos amb la classe principal de l'aplicació.

Modifarem la classe **UF12AddressApp** per a crear una instància del **RootLayoutController** i poder passar-li una referència a si mateix. Amb aquesta referència el controlador podrà després accedir als mètodes (públics) d'**UF12AddressApp**.

```
private void initRootLayout() {  
    try{  
        //Carreguem el FXML  
        FXMLLoader loader = new FXMLLoader(  
            url getClass().getResource(name: "views/RootLayout.fxml"));  
        this.rootLayout = loader.load();  
        //Creem una Scena amb el arxiu FXML  
        Scene scene = new Scene(parent: this.rootLayout);  
        //Assignem l'escena a l Stage.  
        this.primaryStage.setScene(value: scene);  
        /* Creem el controlador per a poder assignar-li una instància de  
           AddressApp */  
        RootLayoutController controller = loader.getController();  
        controller.setAddressApp(address app:this);  
        //Mostrem el Stage  
        this.primaryStage.show();  
    }catch(IOException e){  
        e.printStackTrace();  
    }  
}
```

## 14. PERSISTÈNCIA DE DADES

Ara afegirem al RootLayoutController les diferents funcions que és podran utilitzar des del menú de RootLayout.fxml.

També, declararem una variable privada:

```
private UF12AddressApp address_app;
```

```
/*
 * S'obte, des de la classe principal.
 * @param address_app
 */
public void setAddressApp(AddressApp address_app) {
    this.address_app = address_app;
}

/**
 * Tanca la aplicació
 *
 */
@FXML
public void exit(){
    System.exit(status: 0);
}

/**
 * Es carrega el observableList des de l'arxiu.
 *
 */
@FXML
public void openFile(){
    File arxiu = this.mostraDialeg(tipus: "open");
    //Si s'ha obtingut el nom de un arxiu
    if(arxiu != null){
        this.address_app.loadContactDataFromFile(arxiu);
    }
}
```

## 14. PERSISTÈNCIA DE DADES

A continuació, afegirem les funcions per a poder emmagatzemar la informació.

```
/*
 * Es guarda en l'arxiu l'observableList que existeix o crida a saveAsFile().
 */
@FXML
public void saveFile() {
    File arxiu = this.address_app.getContactFilePath();
    if(arxiu != null){
        this.address_app.saveContactDataToFile(arxiu);
    }else{
        this.saveAsFile();
    }
}

/**
 * Es mostra l'explorador per a posar-li nom a l'arxiu i el guarda.
 */
@FXML
public void saveAsFile() {
    File arxiu = this.mostraDialog(tipus: "save");
    // Si s'ha obtingut el nom i ruta d'un arxiu
    if (arxiu != null) {
        // Asegurem la extensió per si l'usuari ha pogut evitar el filtre
        if (!arxiu.getPath().endsWith(suffix: ".txt")) {
            // Creem el arxiu amb la extensió ".txt"
            arxiu = new File(arxiu.getPath() + ".txt");
        }
        this.address_app.saveContactDataToFile(arxiu);
    }
}
```

```
/*
 * Carrega l'explorador per a moder elegir el nom o el arxiu.
 */
private File mostraDialog(String tipus) {
    File arxiu;
    //Classe per a poder seleccionar el arxiu desde el explorador
    FileChooser seleccionador = new FileChooser();
    //Filtre per a evitar que se seleccionen arxius d'extensions no permeses
    FileChooser.ExtensionFilter extensio = new FileChooser.ExtensionFilter(
        string: "Archivos de texto", strings:"*.txt");

    //Afegim al seleccionador la restricció de l'extensió
    seleccionador.getExtensionFilters().add(e: extensio);
    //Mostrem el explorador amb un dialog que apareixerà sobre primary_stage
    if(tipus.equals(anobject: "save")){
        //Si el tipus es "save" obri l'explorador per a guardar
        return arxiu = seleccionador.showSaveDialog(
            ownerWindow:address_app.getPrimaryStage());
    }else{
        /* Si el tipus es diferent a "save" obri l'explorador per a
        seleccionar l'arxiu.*/
        return arxiu = seleccionador.showOpenDialog(
            ownerWindow:address_app.getPrimaryStage());
    }
}
```

## 14. PERSISTÈNCIA DE DADES

Java ens permet guardar una certa informació mitjançant una classe anomenada Preferences, pensada per a guardar les preferències d'usuari d'una aplicació. Depenent del sistema operatiu, aquestes preferències són guardades en un lloc o un altre (per exemple el registre de Windows).

No podem usar un arxiu de Preferences per a guardar la nostra llibreta d'adreses completa, però ens serveix per a guardar informació d'estat molt simple.

Un exemple del tipus de coses que podem guardar en aquestes preferències és la ruta a l'últim arxiu obert. Amb aquesta informació podem recuperar l'últim estat de l'aplicació quan l'usuari torne a executar l'aplicació.

## 14. PERSISTÈNCIA DE DADES

Afegirem a [UF12AddressApp.java](#) les següents funcions per a poder guardar i carregar des de Preferences la ruta de l'arxiu que volem utilitzar.

```
public void setContactFilePath(File arxiu) {
    Preferences prefs = Preferences.userNodeForPackage(c: getClass());
    if(arxiu != null){
        prefs.put(key:"ruta_arxiu", value: arxiu.getPath());
    }else{
        prefs.remove(key:"ruta_arxiu");
    }
}

public File getContactFilePath(){
    Preferences prefs = Preferences.userNodeForPackage(c: getClass());
    String ruta_arxiu = prefs.get(key:"ruta_arxiu", def:null);
    if(ruta_arxiu != null){
        return new File(pathname:ruta_arxiu);
    }else{
        return null;
    }
}
```

## 14. PERSISTÈNCIA DE DADES

A més, des del RootLayoutController volem poder obrir un quadre de diàleg sobre la pantalla de l'aplicació que ens permeta seleccionar els arxius.

Per a poder conèixer la instància de la pantalla de l'aplicació hem de poder tindre un mètode `getPrimaryStage()` en la classe AddressApp.java

```
public Window getPrimaryStage() {  
    ...  
    return this.primaryStage;  
}
```

## 14. PERSISTÈNCIA DE DADES

Una vegada tenim tots els mètodes, cal guardar i carregar la informació des de un fitxer.

```
public void saveContactDataToFile(File arxiu){  
    try {  
        FileWriter fitxer = new FileWriter(file:arxiu);  
        fitxer.write(str:"");  
        fitxer.close();  
        fitxer = new FileWriter(file:arxiu, append: true);  
        for (Contact contact : this.contactes) {  
            String str = contact.getNom().get() + ","  
                + contact.getCognoms().get() + ","  
                + contact.getDomicili().get() + ","  
                + contact.getCiutat().get() + ","  
                + String.valueOf(contact.getCodi_postal().get()) + ","  
                + DateUtil.format(data:contact.getData_de_naixement())  
                    .get());  
            fitxer.write(str);  
            fitxer.write(str: System.lineSeparator());  
        }  
        fitxer.close();  
        this.setContactFilePath(arxiu);  
    } catch (Exception ex) {  
        System.err.println("Error al guardar els contactes en l'arxiu: "  
            + arxiu.getName());  
    }  
}
```

```
public void loadContactDataFromFile(File arxiu) {  
    this.contactes.clear();  
    try {  
        FileReader fr = new FileReader (file:arxiu);  
        BufferedReader br = new BufferedReader(in: fr);  
        // Lectura del fitxer  
        String linea;  
        while((linea = br.readLine())!=null){  
            String[] contacte = linea.split(regex: ",");  
            String[] fecha = contacte[5].split(regex: "\\\\".  
            this.contactes.add(new Contact(  
                contacte[0],  
                contacte[1],  
                contacte[2],  
                contacte[3],  
                cp: Integer.parseInt(contacte[4]),  
                dia:Integer.parseInt(fecha[0]),  
                mes:Integer.parseInt(fecha[1]),  
                any:Integer.parseInt(fecha[2])));  
        }  
        fr.close();  
        this.setContactFilePath(arxiu);  
    } catch (Exception ex) {  
        System.err.println("No s'ha trobat l'arxiu: " + arxiu.getName());  
    }  
}
```

## 15. ACCIONS DE MENÚ

En el nostre RootLayout.fxml ja hi ha un menú, però encara no l'hem utilitzat. Abans d'afegir accions al menú crearem tots els ítems del menú.

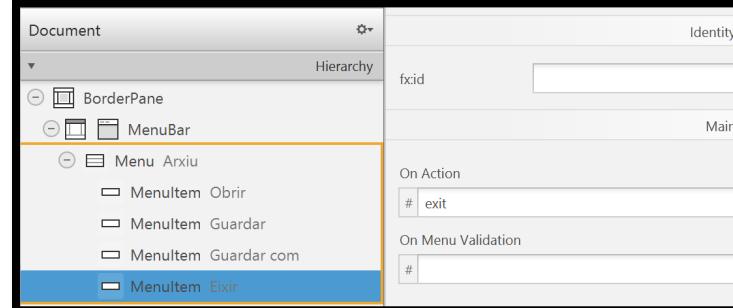
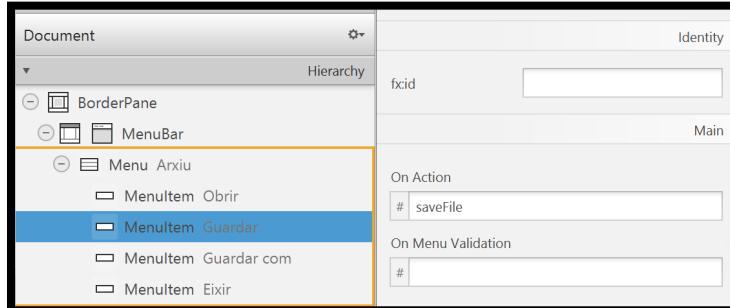
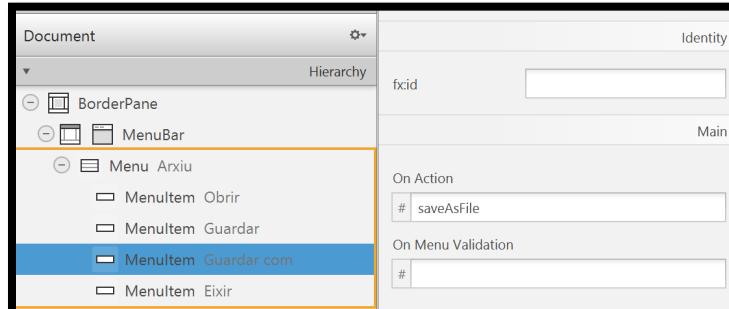
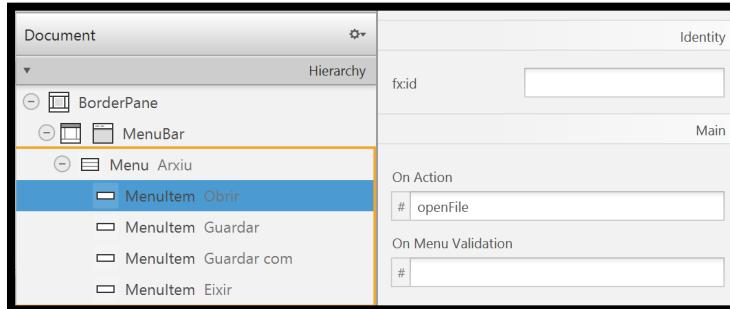
Obrim l'arxiu RootLayout.fxml en Scene Builder i arrossega els ítems de menú necessaris des de la secció library a la barra de menús (componentMenuBar en la hierarchy).

Crea els ítems com es veuen en la imatge:



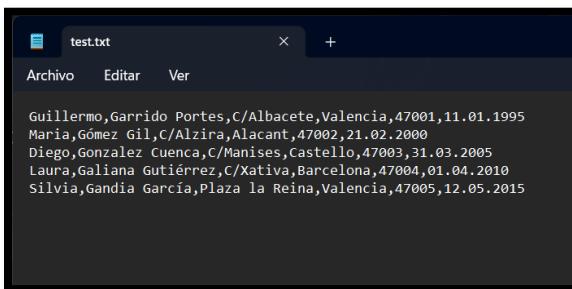
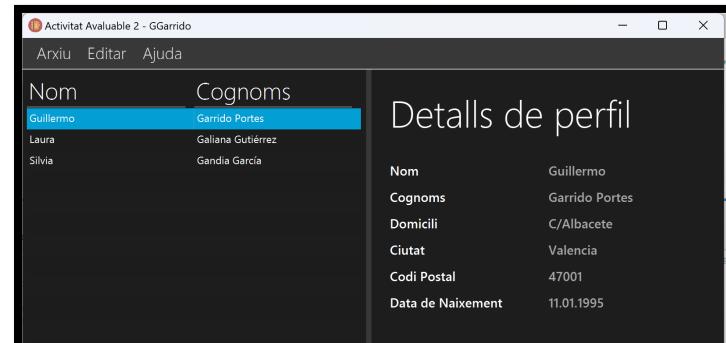
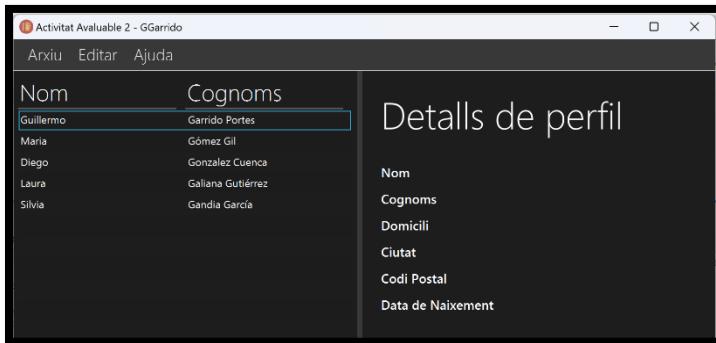
## 15. ACCIONS DE MENÚ

Per últim sols queda cridar des del menú de la vista a cada una de les funcions que s'han implementat.



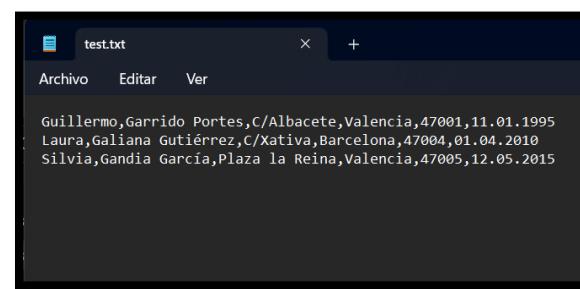
## 16. EXECUTAR L'APLICACIÓ

Ara que ja tenim tot en funcionament s'ha de provar l'execució de la aplicació i comprovar que tot guarda, carrega i mostra la informació correctament.



The screenshot shows a terminal window titled "test.txt" with the following content:

```
Guillermo,Garrido Portes,C/Albacete,Valencia,47001,11.01.1995
Maria,Gómez Gil,C/Alzira,Alacant,47002,21.02.2000
Diego,Gonzalez Cuenca,C/Manises,Castello,47003,31.03.2005
Laura,Galiana Gutiérrez,C/Xativa,Barcelona,47004,01.04.2010
Silvia,Gandia García,Plaza la Reina,Valencia,47005,12.05.2015
```



The screenshot shows the same terminal window after changes were made, reflecting the updated data from the application:

```
Guillermo,Garrido Portes,C/Albacete,Valencia,47001,11.01.1995
Laura,Galiana Gutiérrez,C/Xativa,Barcelona,47004,01.04.2010
Silvia,Gandia García,Plaza la Reina,Valencia,47005,12.05.2015
```

**Activitat Avaluable**

