

# 1. Vuelos FX

---

**Unidad 1. Ejercicio final**

**Programación de Servicios y Procesos**

Nacho Iborra  
Álvaro Pérez  
Javier Carrasco



Esta obra está bajo una [Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional](https://creativecommons.org/licenses/by-nc-sa/4.0/).

## Tabla de contenido

<b>1. Introducción y primeros pasos.....</b>	<b>3</b>
1.1. Configurar el proyecto.....	3
<b>2. Estructura de las clases.....</b>	<b>4</b>
2.1. La clase <i>Flight</i> .....	4
2.2. Clase <i>FileUtils</i> .....	4
2.2.1. Estructura del archivos de vuelos.....	4
2.3. Otras clases útiles.....	5
<b>3. La aplicación JavaFX.....</b>	<b>6</b>
3.1. Diseño de la vista principal.....	6
3.2. ¿Cómo debería funcionar?.....	7
3.2.1. Operaciones insertar/eliminar.....	7
3.2.2. Aplicar filtros.....	7
3.2.3. Guardando cambios.....	8
<b>4. Mejoras opcionales.....</b>	<b>8</b>
4.1. Agregar gráficos.....	8
4.2. Añadir estilos CSS.....	9
4.3. Actualizar vuelos.....	9
<b>5. Reglas de evaluación.....</b>	<b>10</b>
5.1. Parte obligatoria.....	10
5.2. Acerca de las mejoras opcionales.....	10

# 1. Introducción y primeros pasos

En el ejercicio final de esta unidad deberás implementar una aplicación JavaFX para gestionar una lista de vuelos de un aeropuerto. Con esta aplicación se podrá filtrar vuelos según unos criterios. La apariencia de la aplicación será más o menos así:

Flight Number	Destination	Departure	Duration	
IB601N	Oviedo	28/10/2017 11:33	0:50	
RY112A	Edinburgh	31/10/2017 16:05	2:35	
AA225H	New York	01/11/2017 10:10	8:12	
FE811Y	Dubai	12/11/2017 15:55	6:17	
IB113U	Santander	30/09/2017 08:55	0:55	
RY131P	London	20/12/2017 11:15	2:28	
AA392Y	Seattle	30/11/2017 06:45	12:14	

Flight Number:

Destination:

Add

Departure:

Duration:

Delete

▼

Apply Filter

## 1.1. Configurar el proyecto

Crear un proyecto JavaFX (una aplicación FXML, para ser más precisos), llamada **FlightsFX**:

Una vez creado el proyecto, en paquete principal de la sección `src>main>java` crea la siguiente estructura:

- El paquete principal *com.example.flightsfx*, contendrá la clase principal y los controladores de JavaFX.
- El paquete *com.example.flightsfx.model*, para almacenar el modelo (clase *Flight*, se explica más adelante).
- El paquete *com.example.flightsfx.utils*, para almacenar algunas clases útiles.

**NOTA:** sustituye *example* por tu nombre y primer apellido.

## 2. Estructura de las clases

Además de la aplicación principal JavaFX con el archivo FXML y el controlador (se verá en la siguiente sección), se necesitan algunas clases adicionales para almacenar la información sobre los vuelos y ayudar a realizar algunas operaciones básicas.

### 2.1. La clase *Flight*

Añade una nueva clase dentro del paquete *model* llamada ***Flight***. Esta clase tendrá los siguientes atributos:

- El número de vuelo (un *String*, como “IB601N”).
- El destino (el nombre de una ciudad).
- La hora y fecha de salida (usa un objeto *LocalDateTime*).
- La duración del vuelo, en horas y minutos (por ejemplo: 2:30 significará 2 horas y 30 minutos). Utiliza un objeto *LocalTime* para almacenar esta información.

Además, deberás añadir dos constructores:

- Un constructor con el número de vuelo.
- Un constructor con todos los atributos.

Añade los *getters* y *setters* para cada atributo.

### 2.2. Clase *FileUtils*

Para recuperar y guardar la información en un archivo de texto, deberás crear una clase llamada ***FileUtils*** en el paquete *utils*. Esta clase tendrá un conjunto de métodos estáticos de utilidad como:

- ***static List<Flight> loadFlights()*** para cargar los vuelos desde un archivo.
- ***static void saveFlights(List<Flight>)*** para guardar una lista de vuelos en un archivo.

Para ambas operaciones, habrá un archivo llamado ***flights.txt*** en la carpeta raíz del proyecto.

#### 2.2.1. Estructura del archivos de vuelos

La estructura del archivo de texto que contiene la información del vuelo (*flights.txt*) será la siguiente:

```
flight_number;destination;departure;duration
```

Por ejemplo:

```
IB601N;Oviedo;28/10/2017 11:33;0:50  
RY112A;Edinburgh;31/10/2017 16:05;2:35  
...
```

Los atributos estarán separados por “;”.

## 2.3. Otras clases útiles

Aunque no es obligatorio, puede ser útil añadir algunas otras clases. Por ejemplo, una clase llamada *MessageUtils* (en el paquete *utils*) para mostrar diferentes mensajes aviso (*Alert*). Podría contener los siguientes métodos estáticos:

- ***static void showError(String message)*** para mostrar mensajes de error.
- ***static void showMessage(String message)*** para mostrar mensajes de información.
- ...

Puedes añadir tantas clases como necesites dentro del paquete *utils*.

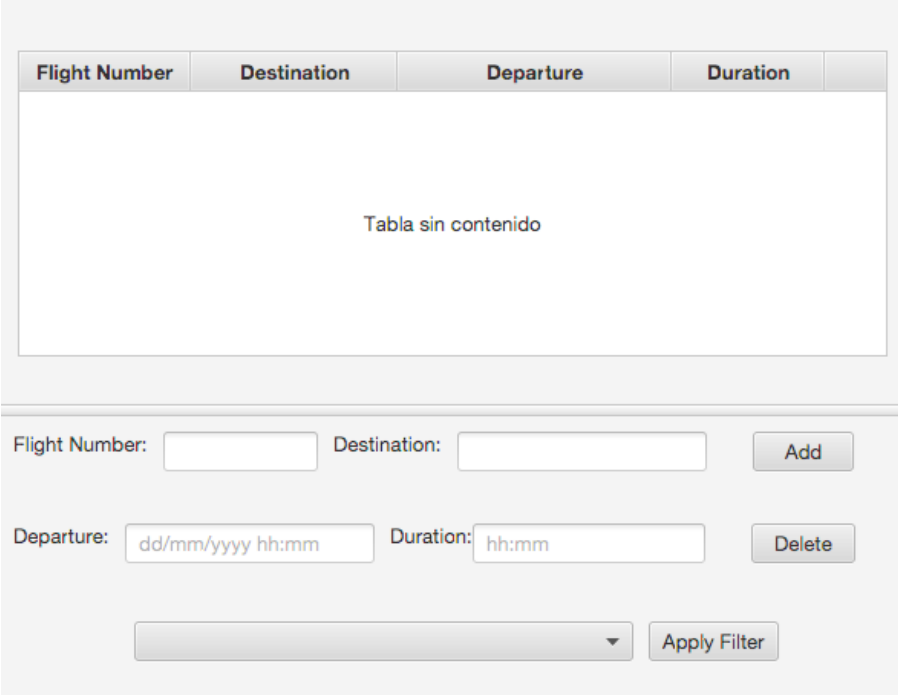
### 3. La aplicación JavaFX

Crea las clases de la aplicación JavaFX siguiendo estos pasos:

1. La aplicación principal de JavaFX se llamará **FlightsFX** dentro paquete principal (*com.example.flightsfx*).
2. Habrá un archivo FXML llamado **FXMLMainView.fxml** con su controlador asociado (*FXMLMainViewController.java*). El archivos FXML se colocará en el paquete *com.example.flightsfx* de la carpeta *resources*, el controlador se ubicará en el paquete principal de la carpeta *java*.
3. Asegúrate de que la clase principal *FlightsFX* carga el contenido del archivo FXML.

#### 3.1. Diseño de la vista principal

Utilice ahora *SceneBuilder* para diseñar la escena principal y obtener una apariencia similar a esta:



Flight Number	Destination	Departure	Duration
Tabla sin contenido			

Flight Number:

Destination:

Add

Departure:

Duration:

Delete

Apply Filter

Puedes, por ejemplo, utilizar un *SplitPane* vertical como contenedor principal, y luego:

- Coloca un *TableView* con su correspondiente *TableColumns* en la sección superior.
- Coloca los controles del formulario debajo de la tabla en la sección inferior (utiliza los contenedores apropiados para mostrar estos controles).

Recuerda establecer un *fx:id* para cada elemento al que pueda ser necesario acceder desde el controlador.

## 3.2. ¿Cómo debería funcionar?

Tan pronto como se inicie la aplicación, deberá cargar el archivo *flights.txt* desde la raíz del proyecto en una lista de objetos *Flight*, después asociará esta lista con el *TableView*, de modo que se muestre un vuelo en cada fila:

Flight Number	Destination	Departure	Duration	
IB601N	Oviedo	28/10/2017 11:33	0:50	
RY112A	Edinburgh	31/10/2017 16:05	2:35	
AA225H	New York	01/11/2017 10:10	8:12	
FE811Y	Dubai	12/11/2017 15:55	6:17	
IB113U	Santander	30/09/2017 08:55	0:55	
RY131P	London	20/12/2017 11:15	2:28	
AA392Y	Seattle	30/11/2017 06:45	12:14	

Flight Number:

Destination:

Add

Departure:

Duration:

Delete

Apply Filter

### 3.2.1. Operaciones insertar/eliminar

Una vez cargada la lista, se podrán realizar las siguientes operaciones:

Si se completan los datos de los campos *Flight number*, *Destination*, *Departure* y *Duration* en el formulario inferior (ninguno puede estar vacío), y haces clic en el *Add*, se añadirá un nuevo vuelo a la lista de vuelos (y a la tabla).

El botón *Delete* debe estar deshabilitado al inicio (usa el método *setDisable* para hacer esto). Cuando se haga clic en cualquier vuelo de la tabla, se activará y se podrá eliminar el vuelo de la lista.

### 3.2.2. Aplicar filtros

Bajo del formulario de vuelo, habrá un desplegable (o cuadro combinado, si lo prefieres), con una lista de filtros:

- **Show all flights:** mostrará todos los vuelos de la lista en la tabla.
- **Show flights to currently selected city:** mostrará todos los vuelos a la ciudad del vuelo seleccionado actualmente en la tabla. Si no se selecciona ningún vuelo, esta opción generará una alerta de error.
- **Show long flights:** mostrará todos los vuelos cuya duración sea superior a 3 horas.

- **Show next 5 flights:** mostrará solo los próximos 5 vuelos que saldrán del aeropuerto. Para hacer esto, debes excluir todos los vuelos pasados, ordenar la lista filtrada por fecha y hora de salida y luego limitar el resultado a 5 vuelos (usa filtros para hacer todo esto).
- **Show flight duration average:** calculará la duración media de todos los vuelos, y mostrará el resultado en un *Alert* con el formato hh: mm.

Todos estos filtros deben implementarse utilizando *streams* y expresiones lambda siempre que sean necesarios.

### 3.2.3. Guardando cambios

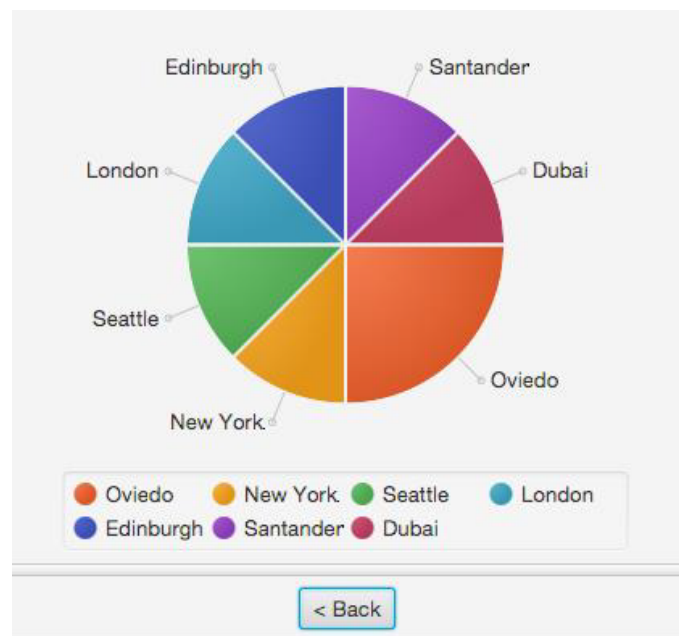
Siempre que se intente cerrar la aplicación, deberás capturar el evento y guardar en el archivo de vuelos la lista de vuelos actual (no la que se muestra en la tabla, ya que puede estar filtrada).

## 4. Mejoras opcionales

Además de la parte obligatoria de este ejercicio, puedes intentar implementar estas mejoras opcionales para obtener una mejor nota.

### 4.1. Agregar gráficos

En la parte izquierda de *SceneBuilder* hay una sub-sección llamada *Charts*, con una lista de gráficos que se pueden añadir a la aplicación JavaFX. Intente añadir una opción para mostrar un gráfico circular (*pie chart*) con el número de vuelos por destino. Debes mostrarlo en una *scene* separada dentro del mismo *stage* (ver en el Anexo V cómo crear aplicaciones con múltiples vistas).





## 4.2. Añadir estilos CSS

Añade una hoja de estilo CSS con algunos estilos predeterminados para la vista principal, como:

- *Background color*: elige entre cualquier color que no sea gris, negro o blanco.
- *Font color*: según el fondo elegido, puede ser blanco o gris.
- Cada botón deberá tener un fondo negro con color de texto blanco.
- Otros estilos que quieras añadir.

## 4.3. Actualizar vuelos

Añade una nueva opción al formulario que permita actualizar los datos de un vuelo sin tener que eliminarlo y volver a añadirlo a la lista.

## 5. Reglas de evaluación

### 5.1. Parte obligatoria

Para obtener la nota final, se aplicarán los siguientes criterios:

- Estructura de clases (paquetes *model* y *utils*), con las clases *Flight* y *FileUtils* con el código correspondiente, y todas las demás clases útiles adicionales que puedas necesitar: 1 punto.
- Diseño de la aplicación JavaFX, similar al mostrado en las figuras anteriores: 1 punto. Debes utilizar los contenedores de diseño adecuados para organizar los controles correctamente en la vista (*HBox* y/o *VBox* son muy útiles).
- Cargar vuelos en la lista y la tabla, y añadir/eliminar vuelos de la lista y la tabla (deshabilitar el botón *Delete* cuando no se puede realizar esta operación): 3 puntos.
- Aplicar filtros de la lista inferior: 0,75 puntos cada filtro (Mostrar todos los vuelos no cuenta).
- Mostrar mensajes de error siempre que no se pueda hacer algo (por ejemplo, eliminar un vuelo sin seleccionar uno de la tabla o añadir un nuevo vuelo con algún campo vacío): 1 punto.
- Documentación del código (comentarios de *Javadoc* para cada clase y método público o constructor), limpieza y eficiencia: 1 punto.

### 5.2. Acerca de las mejoras opcionales

Si decides implementar alguna mejora opcional para este ejercicio, deberás tener en cuenta que:

- Si tu parte obligatoria es inferior a 10, puedes obtener hasta 10 puntos implementando una (o muchas) de las mejoras opcionales (hasta 1 punto por mejora).
- Si tu parte obligatoria es perfecta (10 puntos), puedes conseguir hasta 11 puntos implementando TODAS las mejoras opcionales.