2. Modelo-Vista-Controlador

Los componentes de la arquitectura del patrón MVC están diseñados para manejar diferentes aspectos de una aplicación en desarrollo. El patrón de diseño MVC sirve para separar la capa de presentación de la lógica de la aplicación y es uno de los patrones de diseño de software más utilizados para el desarrollo web y de aplicaciones. Este patrón de diseño separa los distintos aspectos de nuestro proyecto en 3 grupos:

- **Modelo**: son todas las clases relacionadas con el dominio de nuestra aplicación. Clases que realizan la lógica del programa y clases que se utilizan para almacenar y gestionar los datos, a menudo, conectados con una base de datos.
- **Vista**: Interfaz Gráfica de Usuario. La vista contiene todas las funciones que interactúan directamente con el usuario, como hacer clic en un botón o en un evento de entrada. Además, se encarga de mostrar los datos almacenados al usuario.
- **Controlador**: conecta el modelo y la vista. Cuando un usuario interactúa con la IGU, solicita al controlador que se ejecute un evento. Este evento va a comunicar con el modelo de la aplicación e intercambiar datos. Estos datos serán devueltos al controlador, que los mostrará al usuario a través de la vista (IGU).

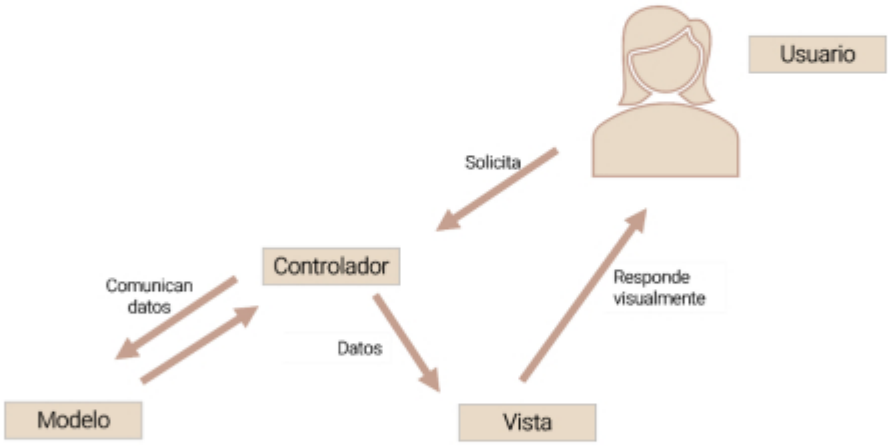

Fig. 13.4 Esquema del patrón Modelo-Vista-Controlador

El patrón MVC permite la separación de tareas, pues divide la lógica entre los 3 grupos de clases, de modo que cada grupo pueda actuar de forma independiente.

El modelo, la vista y el controlador no dependen el uno del otro. ¿Qué importancia tiene esto? ¡En realidad, mucha! Conviene recordar que, generalmente, el software lo trabajan distintos equipos: así, un equipo puede estar formado por un diseñador, un ingeniero y un arquitecto de base de datos. La separación de tareas implica que todos los miembros del equipo pueden trabajar en sus piezas al mismo tiempo, ya que la lógica del trabajo se ha dividido en grupos.

La separación de tareas también es excelente para el mantenimiento: los desarrolladores pueden corregir un error en una parte del código, sin tener que revisar las otras partes del mismo.

Hacer modelos, controladores y vistas independientes hace que la organización del código sea simple y fácil de entender, y facilita su mantenimiento. Gracias a eso, los programadores pueden corregir un error en la vista sin cambiar el código del modelo.

Crear un nuevo proyecto

Para crear nuestro primer proyecto JavaFX en Eclipse, en el menú *File - New*, escogeremos la opción *Other*. Dentro de la lista que aparece, en el apartado de JavaFX, seleccionaremos la opción de *JavaFX Project*.

En resumen:

```
File > New > Other > JavaFx Project
```

En la primera ventana que aparece, indicaremos el nombre del proyecto y dejaremos intactas el resto de opciones que se muestran por defecto en esta ventana y en la siguiente.

En la tercera ventana, debemos configurar los siguientes parámetros:

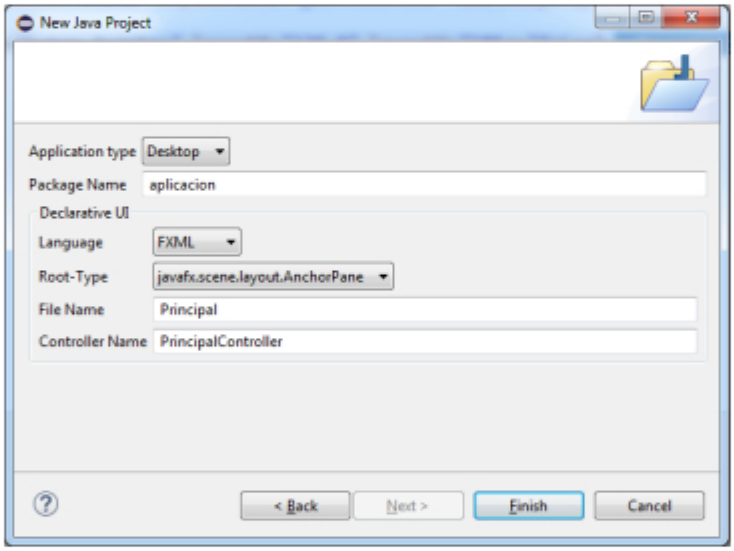

Fig13.5 Opciones en Eclipse para crear un nuevo proyecto JavaFX

- **Application type**: puedes escoger si es una aplicación para dispositivo móvil (*mobile*) o para escritorio (*desktop*).
- **Package Name**: nombre del paquete donde se crearán las clases del proyecto nuevo de nuestra aplicación.
- **Language**: indicaremos el lenguaje que queremos utilizar para guardar nuestras vistas. Nuestras IGU las programaremos con FXML.
- **Root-Type**: tipo de layout para el panel principal. Podemos escoger *AnchorPane* para nuestra primera aplicación, por su sencillez de uso.
- **File Name**: nombre del fichero de la vista. Eclipse añadirá automáticamente la extensión *.fxml*.
- **Controller Name**: nombre del fichero del controlador. Eclipse añadirá automáticamente la extensión *.java*.

Se aconseja separar en diferentes paquetes (*new - package*) las clases, dependiendo de la funcionalidad que tengan dentro del proyecto.

Puedes crear los siguientes paquetes:

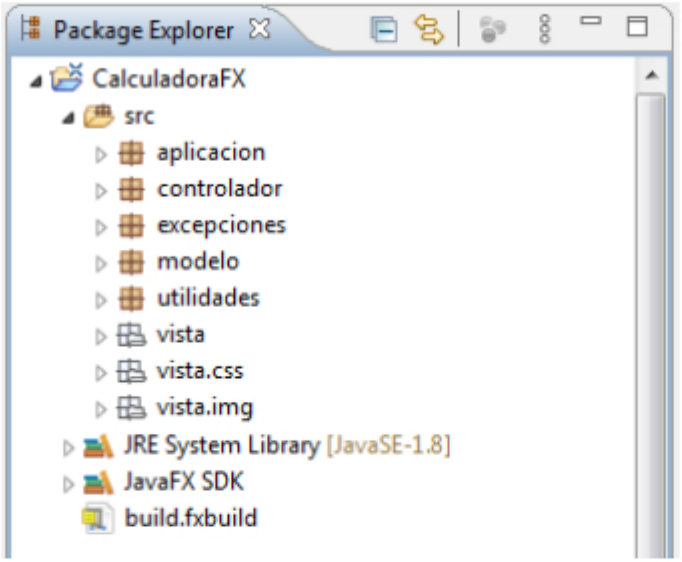

Fig. 13.6 Estructura de un proyecto con JavaFX e IGU

- **aplicación**: paquete donde estará la clase principal (*Main.java*). Es por donde empezará a ejecutarse nuestro programa.
- **controlador**: paquete donde estarán todos los controladores de cada una de las vistas de nuestra aplicación.
- **excepciones**: paquete donde crearemos nuestras propias excepciones de usuario.
- **modelo**: clases relacionadas con el modelo de nuestra aplicación.
- **utilidades**: paquete reservado para clases que no pertenecen al modelo de nuestra aplicación —por ejemplo, clases para guardar cadenas de texto, métodos de cálculos, etcétera—.
- **vista**: aquí guardaremos nuestros ficheros .fxml con las diferentes ventanas de nuestro programa.
- **css** (dentro de vista): se recomienda crear un paquete aparte para las hojas de estilo CSS.
- **img** (dentro de vista): también se recomienda tener un paquete para almacenar las imágenes utilizadas en las ventanas de nuestra aplicación.

Cuando empieza a ejecutarse la aplicación, el primer método que se va a ejecutar es el método *start()* de la clase ***Main.java.***

A continuación, se muestra un ejemplo de cómo se podría programar el método:

```
@Override
public void start(Stage primaryStage) {

	try {
		String fxml = "vista/Calculadora.fxml";

		// Cargar la ventana
		Parent root = FXMLLoader.load(getClass().getClassLoader().getResource(fxml));

		// Cargar la Scene
		Scene scene = new Scene(root);

		// Asignar propiedades al Stage
		primaryStage.setTitle("Calculadora FX");
		primaryStage.setResizable(false);

		// Asignar icono de la aplicación
		primaryStage.getIcons().add(new
		Image(getClass().getResource("/vista/img/icon.png").toExternalForm()));

		// Asignar la scene y mostrar
		primaryStage.setScene(scene);
		primaryStage.show();

	} catch(Exception e) {
		e.printStackTrace();
	}
}
```