PRÁCTICA 10. Programación Orientada a Eventos: GUI | ***2 H (1 Ses)***

Objetivos:

- Afianzar los conceptos vistos en la sesión anterior sobre desarrollo de aplicaciones gráficas
- Verificar el funcionamiento de otros componentes como lista desplegables, jList.
- Incluir un menú en una aplicación gráfica

Contenido:

El objetivo de esta práctica es continuar con el proceso de creación de aplicaciones con interfaces Gráfica así como experimentar los contenidos teóricos expuestos en el tema de teoría: Graphical User Interfaces (GUI).

Ejercicio Gestión de desguace

Se pide desarrollar una aplicación que permita gestionar un desguace como continuación de las prácticas anteriores. Se debe tener las siguientes clases: Desguace, Persona, Empleado, Proveedor, Pieza, Vehiculo y Persona.

Fase 0: Estructuración de los paquetes del proyecto

Crearemos 4 paquetes en el proyecto:

- Vistas: es.unex.cum.mdp.sesion10.vistas donde se incluirán todos los ficheros .fxml
- Controller. es.unex.cum.mdp.sesion10.vistas donde se incluirán todos los ficheros controladores
- Modelo: es.unex.cum.mdp.sesion10.modelo donde se copiará la práctica del desguace
- En es.unex.cum.mdp.sesion10 se tendrá el main de la aplicación.
- Directorio de imágenes: es.unex.cum.mdp.sesion10.resource
- Añadir la librería de JavaFX al buildPath así como eliminar modelo-info.

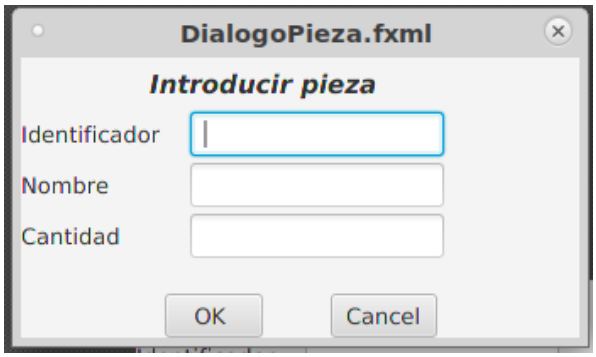
Importante: Copiamos la práctica del desguace en su paquete

Fase 1: Creación del dialógo para añadir una nueva Pieza (DialogoPieza.java)

Pasos:

1. Crear una nueva clase Controller (new-->file--> class) → **DialogoPiezaController** en el paquete controller
    1. Añadir en la definición de la clase implements Initializable
2. Crear un fichero de vista (new-->other--> fxml) de tipo AnchorPane → **DialogoPieza** en el paquete vista

1. Abrir con SceneBuilder
2. Añadir el controlador **es.unex.cum.mdp.sesion10.controller.DialogoPiezaController**
3. Añadir dos paneles: un GridPane y un HboxPane
4. Sobre el GridPane
    1. Crear 3 columnas y 4 filas
    2. En la primera columna fila introducir una etiqueta con texto: **Introducir Pieza**
    3. Problema: No se ve todo el texto.
    4. Seleccionamos el texto y pinchamos en el menú Modify → GridPane → Increase ColumnSpan (para ocupar más columnas)
    5. Cambiamos el formato (mayor tamaño de letra y negrita)
    6. Cambiamos la alineación al centro
    7. Introducir tres etiquetas con valor de text (Identificador, nombre y cantidad)
    8. Introducir tres cuadros de texto con ningún valor.
        1. **Importante**: Como se quiere recuperar sus valores deberemos dar valor a fx:id. (id, nombre, cantidad) para tener acceso desde el Controller
5. Sobre el HboxPane
    1. Añadir dos botones denominado: OK y Cancel
    2. Sobre el botón OK hacer un margin de 40 pixel a la derecha para que deje espacio.
6. Añadimos al botón OK un evento para onAction denominado **PulsadoOk** y otro al Cancel **PulsadoCancel**
7. Salvamos copiamos el código en el Controller → Menú View → Show SampleControler



Fase 2: Creación del dialógo para añadir un nuevo vehiculo (IntroducirVehiculo.java)

Realizamos los mismos pasos del caso anterior (incluso se puede copiar y pegar cambiando la referencia al controller y las etiquetas oportunas).

Fase 3: Pantalla principal

Pasos:

1. Crear una nueva clase denominada MainController en el paquete Controller
2. Crear un nuevo fichero fxml denominado Main.fxml en el paquete vista
3. Crear una nueva clase JavaFX main class denominado Main en el paquete sesion10

```
public class Main extends Application {
	@Override
	public void start(Stage primaryStage) {
		try {
			FXMLLoader loader = new FXMLLoader(getClass().
						getResource("vistas/Main.fxml"));
			//Recuperamos el controller
			Parent root = loader.load();
			MainController controller = (MainController)
						loader.getController();

			Scene scene = new Scene(root);
			primaryStage.setScene(scene);
			//Cuando se cierre se llama al método shutdown de controller
			//primaryStage.setOnHidden(e -> { //Capturar al pulsar Exit
			//    controller.shutdown();
			//    Platform.exit();
			//});
			primaryStage.show();

		} catch (Exception e) {
			e.printStackTrace();
		}
	}
	public static void main(String[] args) {
		launch(args);
	}
}
```

4. Configuración del Fichero fxml → Abrimos SceneBuilder
    1. Establecemos un tamaño de trabajo del stage de 600x600 (luego se podrá modificar)
    2. Toda aplicación gráfica funciona con menú o con botones de acceso de rápido. Para crear un menu añadimos un MenuBar al BorderPane.
        1. Archivo (Menu)→ Salir (MenuItem)
        2. Operaciones (Menu): InsertarPieza, InsertarVehiculo, InsertarProveedor, InsertarEmpleado, VerVehículos, Ver piezas,
        3. Ayuda (Menu) → Acerca (MenuItem)

        4. A continuación capturaremos los eventos al pulsar InsertarPieza e InsertarVehiculo. Para ello sobre el menuItem InsertarPieza e InsertarVehiculo configuramos onAction con nombre (insertarPieza e InsertarVehiculo).

Después seguiremos configurando el resto de la interfaz. A continuación vamos a realizar la operación de insertarPieza y de insertarVehiculo pues sus diálogos ya se han realizado. Toda esta operabilidad se realiza en MainController.

1. Añadimos a MainController un atributo Desguace para poder trabajar con la lógica del programa.
2. En el método initialize(URL arg0, ResourceBundle arg1) cargaremos los datos del fichero que se hizó en la sesion 8 (sino lo habéis realizado, no se podrá cargar y salvar).

```
@Override
public void initialize(URL arg0, ResourceBundle arg1) {
	try {
		CargarSerializable("datos.dat");
		d.cargarPiezas("piezas.txt");
	} catch (ClassNotFoundException | IOException e) {
		Alert alert3 = new Alert(AlertType.INFORMATION);
		alert3.setTitle("No hay fichero!!! ");
		alert3.setContentText("El desguace funcionará sin datos");
		alert3.showAndWait();
		d= new Desguace();
	}

}
```

Ahora que ya tenemos desguace podemos salvar también la información. En el main pusimos una llamada al evento Shutdown. El código será el siguiente:

```
public void shutdown() {
    d.salvarPiezas("piezas.txt");
    try {
        SalvarSerializable(d, "datos.dat");
    } catch (IOException e) {
        //Alert con un mensaje de error
    }
}
```

3. A continuación añadiremos la funcionalidad de InsertarPiezas introduciendo el siguiendo código en el evento oportuno. La idea es crear un objeto FXMLLoader para cargar la ventana de diálogo de InsertarPieza.

```
@FXML
void InsertarPiezas(ActionEvent event) throws IOException {
  //Se establece el fichero fxml que se va a cargar
  FXMLLoader fxmlLoader = new FXMLLoader(getClass().
                          getResource("../vistas/DialogoPieza.fxml"));
  Parent parent = fxmlLoader.load();
  //Se establece su escena
  Scene scene = new Scene(parent, 300, 200);
  //Se establece su stage
  Stage stage = new Stage();
  stage.initModality(Modality.APPLICATION_MODAL); //Modal
  stage.setScene(scene);
  //Se muestra
  stage.showAndWait();
}
```

IMPORTANTE: En este código existe un problema: los datos que se introducen en el diálogo de InsertarPieza quedan en el controlador de InsertarPieza. Por tanto cualquier dato introducido no se puede almacenar en el desguace pues el objeto Desguace se encuentran en un controlador distinto.

Para solventar este problema se pueden optar por varias soluciones, de las cuales veremos:

- Devolver un objeto (Pieza en este caso) del controlador hijo (DialogoPiezaController) al controlador Padre (MainController) con los datos que se han capturado del diálogo. Posteriormente en el controlador principal lo almacenaremos en Desguace.
- Pasar el controlador principal al controlador hijo y de este modo se tendría acceso al objeto Desguace

Solución basada en devolución de un objeto (InsertarPiezas)

En primer lugar se debe adaptar el Controlador de Pieza: En primer lugar tendremos un atributo Pieza que será el retornado y se establecerá el código cuando se pulsa Ok (se crea la pieza con los datos) o Cancel (la pieza a devolver vale null). En este caso la ventana de Dialogo crea el objeto pero toda la lógica se queda en el controlador principal.

```
public class DialogoPiezaController implements Initializable {
	@FXML
	private TextField id;
	@FXML
	private TextField nombre;
	@FXML
	private TextField cantidad;

	private Pieza p = null;
	public Pieza getP() {return p;}
	public void setP(Pieza p) {this.p = p;    }

	@Override
	public void initialize(URL arg0, ResourceBundle arg1) {
		id.setText("");
		nombre.setText("");
		cantidad.setText("");
	}
	@FXML
	void PulsadoCancel(ActionEvent event) {
		p = null;
		closeStage(event);
	}
	@FXML
	void PulsadoOk(ActionEvent event) {
		try {
			p = new Pieza(id.getText(), nombre.getText(),
					Integer.parseInt(cantidad.getText()));
			closeStage(event);
		} catch (java.lang.NumberFormatException e) {
			p = null;
		}
	}
	private void closeStage(ActionEvent event) {
		Node source = (Node) event.getSource();
		Stage stage = (Stage) source.getScene().getWindow();
		stage.close();
	}
}
```

En segundo lugar, una vez que carguemos el diálogo y se introduzca los datos y se pulse OK o Cancel, tendremos que modificar el Controlador del Main para recuperar la pieza e insertarla en el Desguace.

```
@FXML
void InsertarPiezas(ActionEvent event) throws IOException {
	// Se establece el fichero fxml que se va a cargar
	FXMLLoader fxmlLoader = new FXMLLoader(getClass().
			getResource("../vistas/DialogoPieza.fxml"));
	Parent parent = fxmlLoader.load();

	// Se recupera el objeto controlador asociado al diálogo
	DialogoPiezaController dialogController = fxmlLoader.getController();

	// Se establece la escena
	Scene scene = new Scene(parent, 300, 200);
	// Se establece el stage
	Stage stage = new Stage();
	stage.initModality(Modality.APPLICATION_MODAL);
	stage.setScene(scene);
	stage.showAndWait();


	//Una vez finalizado, se recupera la pieza (null o un objeto)
	Pieza p = dialogController.getP();
	if (p == null) {
		Alert alert3 = new Alert(AlertType.INFORMATION);
		alert3.setContentText("Has pulsado cancel o datos incorrectos");
		alert3.showAndWait();
	} else {
		Alert alert3 = new Alert(AlertType.INFORMATION);
		if (d.addPieza(p.getXXXX())) {
			alert3.setContentText("Añadido correctamente");
		} else {
			alert3.setContentText("No se ha añadido");
		}
		alert3.showAndWait();
	}
}
```

Solución basada en stablecer el controlador principal en los hijos(InsertarVehiculos)

En este caso, el controlador principal se pasa al controlador secundario (diálogo) y es esté el que realiza toda la lógica. Haremos esta opción para la operación de InsertarVehículo.

```
@FXML
void InsertarVehiculo(ActionEvent event) throws IOException {
	// Se establece el fichero fxml que se va a cargar
	FXMLLoader fxmlLoader = new FXMLLoader(getClass().
				getResource("../vistas/DialogoVehiculo.fxml"));
	Parent parent = fxmlLoader.load();

	// Se crear el controlador de diálogo y se pasa el controlador principal
	DialogoVehiculoController dialogController = fxmlLoader.getController();
	dialogController.setM(this); //Se pasa el controlador

	// Se establece la escena
	Scene scene = new Scene(parent, 300, 200);
```

```
        // Se establece el stage
        Stage stage = new Stage();
        stage.initModality(Modality.APPLICATION_MODAL);
        stage.setScene(scene);
        stage.showAndWait();

}
```

A continuación se debe modificar el diálogo de Vehículo para que se encargue de hacer toda la lógica asociada a este procedimiento.

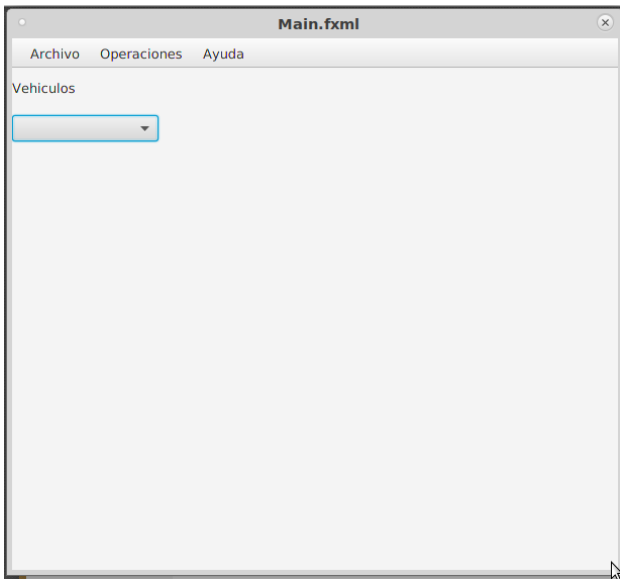
```
public class DialogoVehiculoController implements Initializable {
        @FXML
        private TextField marca;
        @FXML
        private TextField modelo;
        @FXML
        private TextField bastidor;

        //Atributo del mainController para cuando se hace el setter
        private MainController mc = null;
        //Solo el setter, así se evita que se devuelva
        public void setM(MainController mc) {this.mc = mc;}


        @FXML
        void PulsadoCancel(ActionEvent event) {
                closeStage(event);
        }
        @FXML
        void PulsadoOk(ActionEvent event) {
           try {
                Vehiculo v= new Vehiculo ();
                v.setMarca(marca.getText());
                v.setModelo(modelo.getText());
                v.setBastidor(Integer.parseInt(bastidor.getText()));
                if (mc!=null) {
                        Alert alert3 = new Alert(AlertType.INFORMATION);
                        if (mc.getD().addVehiculo(v)) {
                                alert3.setContentText("Añadido correctamente");
                        }else {
                                alert3.setContentText("No se ha añadido");
                        }
                        alert3.showAndWait();
                }
                closeStage(event);
           } catch (java.lang.NumberFormatException e) {}
        }
        private void closeStage(ActionEvent event) {
                Node source = (Node) event.getSource();
                Stage stage = (Stage) source.getScene().getWindow();
                stage.close();
        }
        @Override
        public void initialize(URL arg0, ResourceBundle arg1) {
                // TODO Auto-generated method stub
                marca.setText("");
                modelo.setText("");
                bastidor.setText("");
        }
```

```
}
```

Fase 4: Resto de la interfaz de Pantalla principal

A continuación desarrollaremos el resto de la interfaz de la ventana principal. En la parte izquierda se podrá seleccionar las piezas o vehículos de forma rápida y en la parte central se dará información de los mismos.

1. Añadir dos paneles (Oeste y Centro)
    3. Oeste(FlowPane):
        1. Añadir una etiqueta (Vehiculos)
        2. Añadir un cuadro comboBox
        3. Probar con distintos layout → Problema no se consigue una buena distribución
        4. Solución
            1. Añadir un panel → GridLayout con 1º columna
            2. Sobre él añadir la etiqueta y el comboBox's
    4. Centro (Vbox). **Importante**: Dadle propiedad fx:id para acceder a él desde el Controller.
        1. De momento no añadiremos nada más y se hará de forma dinámica dependiendo si se selecciona un vehiculo o una pieza.



Fase 5: Mostrar Piezas/Vehiculo

Todas las operaciones relacionadas con la visualización de los datos de Piezas, Vehículos, Empleados y Proveedores se realizará en la parte cenral de la interfaz. Para ello utilizaremos un control TableView que muestra todos los datos en formato Tabla. Se realizará en tiempo de ejecución pues las columnas del Tableview serán distintas dependiendo del caso. Esta funcionalidad se realizará en dos pasos:

Paso 1:

- Creación del tableview: Se añaden dos atributos en el MainController
- Permitir que se puede bajar con una barra de desplazamiento por los datos

- Permitir que se puede ordenar por los valores de lascolumnas
- Seleccionar una fila

Paso 2: No se implementará en nuestro caso

- Editar las celdas de las tablas
- Añadir nuevos elementos
- Borrar elementos

Para mostrar los datos, capturaremos el evento del Menu → Ver Piezas

- Añadir dos atributos en MainController (tableview y ObservableList

```
private TableView table;  //El control Tabla
private ObservableList data; //Donde están los datos
```

- A continuación se establece la configuración del Tableview

```
@FXML
void VerPiezas(ActionEvent event) {
    // Etiqueta de listar Piezas
    Label label = new Label("Listado de Piezas");
    label.setTextFill(Color.DARKBLUE);
    label.setFont(Font.font("Calibri", FontWeight.BOLD, 24));
    vboxCentral.getChildren().add(label);
    // Table view
    table = new TableView();

    //Cargar los datos iniciales. No de momento
    ??????????

     //Se definen las columnas. Se deben llamar igual que los atributos
    TableColumn titleCol = new TableColumn("Id");
    titleCol.setCellValueFactory(new PropertyValueFactory("Id"));
    TableColumn authorCol = new TableColumn("Nombre");
    authorCol.setCellValueFactory(new PropertyValueFactory("Nombre"));
    TableColumn authorCant = new TableColumn("Cantidad");
    authorCant.setCellValueFactory(new PropertyValueFactory("contador"));
    //Se añaden las columnas
    table.getColumns().setAll(titleCol, authorCol, authorCant);
    //Se fija el tamaño
    table.setPrefWidth(350);
    table.setPrefHeight(300);
    table.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
      //Se añade al vbox
    vboxCentral.getChildren().add(table);
}
```

- Solo nos queda proporcionar los datos. Sustituimos la interrogación al cargar los datos por:

```
//Si fuera una lista
//data = FXCollections.observableList(d.getVehiculos());
//table.setItems(data);

//Al ser un set, se debe convertir a Lista
List<Pieza> l = new ArrayList(d.getCatalogo());
data = FXCollections.observableList(l);
table.setItems(data);
```

Fase 7: Mostrar los datos de un vehículo concreto

La idea es mostrar la información de un vehículo cuando se pulse sobre el combobox de un Vehiculo. Para ello tendremos que cargarlo cuando se carga el fichero y por otra parte añadir cuando se añade un nuevo vehículo. Además tendremos que capturar el evento ocAction del combobox.

Paso 1: Cargar los vehículos al arrancar el programa

```
@Override
	public void initialize(URL arg0, ResourceBundle arg1) {
		// TODO Auto-generated method stub
		try {
			CargarSerializable("datos.dat");
			d.cargarPiezas("piezas.txt");
			cargarDatosComboBox();
		} catch (ClassNotFoundException | IOException e) {
			Alert alert3 = new Alert(AlertType.INFORMATION);
			alert3.setTitle("No hay fichero!!! ");
			alert3.setContentText("El desguace funcionará sin datos");
			alert3.showAndWait();
			d = new Desguace();
		}
	}
```

Si añadieramos directamente la lista de Vehículo al ObservableList en el desplegale veríamos el toString de Vehículo. Para evitar eso debemos implementar la clase StringConverter que te permitirá decidir que quieres mostrar en el desplegable (método toString) así como fromString que te permite a partir del String visualizado, devolver un objeto. El código mostrado en esta última función utiliza el método **filter** que nos ayuda con las búsquedas en listas (internamente es como si hiciera un iterator sobre la lista para buscar).

Como atributo:

```
private  ObservableList<Vehiculo> list = FXCollections.observableArrayList();
```

Como código:

```
private void cargarDatosComboBox() {
	// combobox.
	list.addAll(d.getVehiculos());
	combobox.setItems(list);
	combobox.setEditable(true);
	//combobox.getSelectionModel().selectFirst();
	combobox.setConverter(new StringConverter<Vehiculo>() {
		@Override
		public String toString(Vehiculo object) {
			if (object!=null) return null;
			else return object.getBastidor().toString();
		}

		@Override
		public Vehiculo fromString(String string) {
			return (Vehiculo) combobox.getItems().stream().filter(ap ->
```

```
ap.getBastidor().toString().equals(string))
                                .findFirst().orElse(null);
            }
      });

}
```

Paso 2: Al insertar un vehículo se debe añadir al combobox. En DialogoVehiculoController en el método PulsadoOk cuando se añade el vehículo deberemos añadirlo a la lista de ObservableList.

mc.getList().add(v);

Paso 3: Recuperar el vehículo.

1. Capturaremos el evento onAction sobre el comboBox
2. Deberíamos usar el método getSelectedItem pero como hemos usado StringConverter este método ya te devuelve el objeto sobre el que has pulsado.

```
@FXML
void seleccionVehiculo(ActionEvent event) {
      Vehiculo v=combobox.getValue();
      System.out.println(v.toString());
}
```

Fase 8: Operabilidad: Añadir Coche, Camión o Moto

En este caso se debe modificar DialogoVehiculoController para poder introducir Coche, Camión o Moto. La forma más sencilla es incluir un combobox con las tres opciones y dependiendo de la seleccionada, crear un objeto Coche, Moto o Camión.

Otra mejora sería también incluir la persona que es propietaria del Vehículo pero se necesitaría incluir las personas del sistema. Otra opción es añadir un nuevo menú de asignación de Propietario a Vehiculo.

Fase 9: Operabilidad: añadir un nuevo Empleado y Proveedor

Se deberá implementar un nuevo cuadro de dialogo con su respectivo controller para añadir un empleado y un proveedor.

Fase 10: Resto de funcionalidades

1. Hacer la operabilidad que añada al vehiculo seleccionado la pieza seleccionada
2. Hacer la operabilidad de devolver todos los Vehiculos de una determinada marca