Proyecto integrado curso 2024/2025

Ciclo Formativo de Grado Superior
Desarrollo de Aplicaciones Multiplataforma

CESUR Plaza Elíptica - Madrid

Gestor de Tareas

Autor: Iván Bacho Cifuentes
Tutora: María Jesús Rodríguez Sánchez

ÍNDICE

1 - Alcance del proyecto

Objetivo general

Mi objetivo con este proyecto ha sido crear una aplicación de escritorio con la que se pueda gestionar tareas personales de manera sencilla y ágil: una alternativa a los gestores “to-do” para poder llevar las tareas diarias de un estudiante.

Funcionalidades mínimas

Alta de tarea

Crear una tarea con un título, descripción opcional, fecha límite y otros datos que ayuden a su gestión.

Edición de tarea

Poder modificar cualquier campo de una tarea ya creada.

Marcado como “Hecha”

Poder cambiar su estado de “Pendiente” a “Hecha”

Posponer la tarea un día

Poder posponer la fecha de vencimiento de una tarea un día.

Borrado de la tarea

Eliminación de la tarea con una confirmación previa al paso definitivo.

Filtros

Poder filtrar las tareas por Todas / Pendientes / Hechas

Vista semanal

Poder filtrar de manera visual las tareas de Lunes a Viernes de la semana actual o mostrarlas Todas.

Persistencia

Almacenar todos los datos de manera local con SQLite

Portabilidad

Ejecución autónoma del JAR con un script .bat .

Tecnologías seleccionadas

Lenguaje: Java
Interfaz de usuario: JavaFX con SceneBuilder
Persistencia: SQLite con el driver jdbc
Pruebas: JUnit 5
Empaquetado: Maven y Shade
Planteado para poder ejecutarse en Windows con el script BAT. El JAR es multiplataforma, por lo que se podrían crear scripts para su ejecución en sistemas Linux y Macintosh.

Planteamiento del proyecto

El proyecto lo he dividido en nueve metas con las que ir avanzando poco a poco. Con cada una se han ido añadiendo funcionalidades y validando lo anterior con pruebas manuales. Han sido las siguientes:

1. Alcance
2. DAO + Modelo
3. Interfaz de usuario básica
4. Filtros y fechas
5. Vista semanal
6. Pruebas con JUnit
7. Pulido de la Interfaz de usuario
8. Empaquetado
9. Memoria y finalización

2 - Capa de datos y modelo

Diseño del esquema SQLite

Para que la aplicación se ejecute de manera cien por cien local, elegí SQLite debido a que no requiere servidor y está soportado por el driver sqlite-jdbc.

```
CREATE TABLE IF NOT EXISTS tareas (
    id          INTEGER PRIMARY KEY AUTOINCREMENT,
    titulo      TEXT    NOT NULL,
    descripcion TEXT,
    vencimiento DATE,
    estado      TEXT    DEFAULT 'PENDIENTE',
    prioridad   TEXT    DEFAULT 'NORMAL'
);
```

- **`id`** - Clave sustituta autoincremental
- **`titulo`** - campo obligatorio para identificación rápida
- **`descripcion`** - nota libre en la que se permite NULL
- **`vencimiento`** - texto con formato ISO de `AÑO-MES-DIA` para facilitar el parseo con `LocalDate`
- **`estado`** `y` **`prioridad`** - texto en mayúsculas para mapear los enums de `Estado` y `Prioridad`

Clase **`Tarea`**

Esta clase representa una fila de la tabla con sus atributos, constructores, getters y setters y un `toString()` para depurar.Tiene un constructor vacío que es necesario para JavaFX y JDBC. Cuando se crea desde la interfaz el atributo `id` y este tiene el valor de `null`, SQLite le asigna el valor definitivo. Por último, en lugar de `java.sql.Date` se usa `LocalDate` que es un tipo de Java más moderno.

Enumeraciones Estado y Prioridad

Estas dos enumeraciones, al estar como texto, permiten que al añadir o renombrar valores en un futuro solo se requiera actualizar el código, manteniendo la base de datos compatible. `Estado` se usa para el filtro de la vista el cálculo del progreso, mientras que `Prioridad` se usa para resaltar filas de alta prioridad en el color rojo.

Clase DbUtil

Con esta clase se encapsula la cadena de conexión y la creación de la tabla. El método estático `getConnection()` abre la conexión y garantiza la existencia del esquema. `setUrl()` está ahí para las pruebas JUnit, pudiendo apuntar a una Base de Datos en memoria.

Clase TareaDao

En esta clase se introducen las operaciones CRUD:

- **`crear()`** , con el código SQL `INSERT`, convierte LocalDate a String si no es null.
- **`listar()`** , con el código SQL `SELECT * ORDER BY` vencimiento devuelve `List<Tarea>` parseando fecha y enums.
- **`actualizar()`** , con el código SQL `UPDATE - WHERE id=?` modifica todos los campos en una sola consulta
- **`borrar()`** , con el código `DELETE FROM tareasWHERE id=?` hace un borrado completo.

Todas estas consultas se realizan en bloques try-with-resources, lo que permite que se cierren automáticamente.

Relación con el resto de las capas

- El controlador JavaFX pide al DAO la lista inicial y la carga en una `ObservableList`
- Cada acción de la interfaz (como pueden ser alta, editar, borrar…) actualiza primero la base de datos a traves del DAO y luego refresca la lista en memoria.

Pruebas unitarias (desarrollo más adelante)

Creo `TareaDaoTest` para asegurar que alta inserte únicamente una fila, el update persista con los cambios de título y prioridad y `delete` elimine solo la fila indicada.

3 - Interfaz de usuario básica

Usé JavaFX porque está en el temario, permite el diseño con SceneBuilder y el componente `TableView` cubre la mayor parte de los requisitos CRUD sin el uso de bibliotecas externas.

Estructura de la ventana principal

La ventana principal se divide básicamente en cuatro zonas:

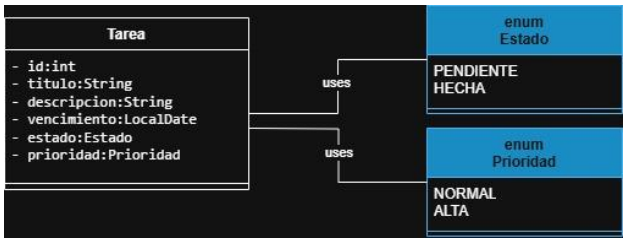
- Zona superior donde está el filtro con el estado de las tareas.
- Tabla central con el listado de las tareas con las columnas Título, Fecha, Estado y Prioridad.
- Panel Lateral con botones de la semana actual y TODOS.
- Zona inferior donde están los botones de gestionar las tareas.

Tabla principal

```
<TableView fx:id="tabla" prefWidth="600.0">
    <columns>
        <TableColumn fx:id="colTitulo"    text="Título"
prefWidth="220"/>
        <TableColumn fx:id="colFecha"     text="Fecha"
prefWidth="140"/>
        <TableColumn fx:id="colEstado"    text="Estado"
prefWidth="100"/>
        <TableColumn fx:id="colPrioridad" text="Prioridad"
prefWidth="100"/>
    </columns>
</TableView>
```

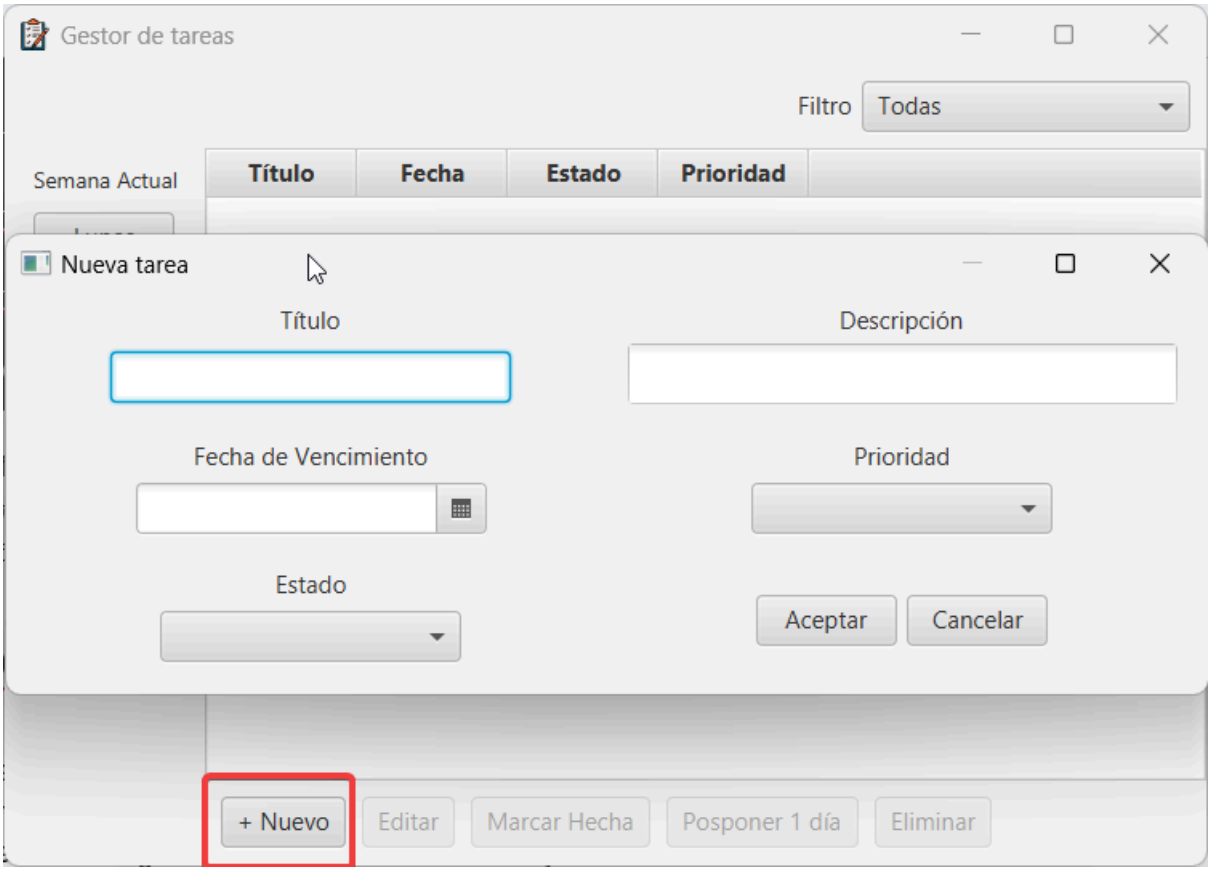
El formato de la fecha se aplica desde Java con `DateTimeFormatter`. He añadido un `RowFactory` para que las filas con la prioridad ALTA pasen a estar de color rojo claro (`#ffcccc`) para resaltar las que son urgentes.

TAREAS		
INT	id	PK
TEXT	titulo	
TEXT	descripcion	
DATE	vencimiento	
TEXT	estado	
TEXT	prioridad	



EditarTareaView.fxml

He creado una vista específica para el cuadro de diálogo de Nueva tarea / Editar tarea con una estructura de `GridPane` de dos columnas. El mismo formulario sirve tanto para la alta como para la edición: si se abre sin parámetro, es alta (constructor vacío), pero si recibe una tarea carga los campos y actualiza el objeto.



Controladores

En SceneBuilder se indicó el controlador `TareasController` para la vista principal y `EditarTareaController` para el cuadro de diálogo extra. Cada botón tiene su `onAction`, que es un método público del controlador, y para el filtro de la semana actual se le asigna el mismo `onFiltroDia` a los cinco botones equivalentes a los días de la semana porque dentro del método se identifica la fuente del evento.

Flujo de acciones CRUD

- **+Nueva** - Abre el diálogo vacío, lo valida, accede a `dao.crear()` y añade a `ObservableList`.
- **Editar** - Selecciona la fila, abre el diálogo con datos, accede a `dao.actualizar()` y reemplaza en la lista.

- **Marcar Hecha** - Cambia el enum Estado, accede a dao.actualizar() y refresca la tabla.
- **Posponer 1 día** - cambia el `setVencimiento(+1 día)`, accede al `dao.actualizar()` y refresca la tabla.
- **Borrar** - Aparece una alerta de confirmación, accede a `dao.borrar(id)` y lo quita de la lista.

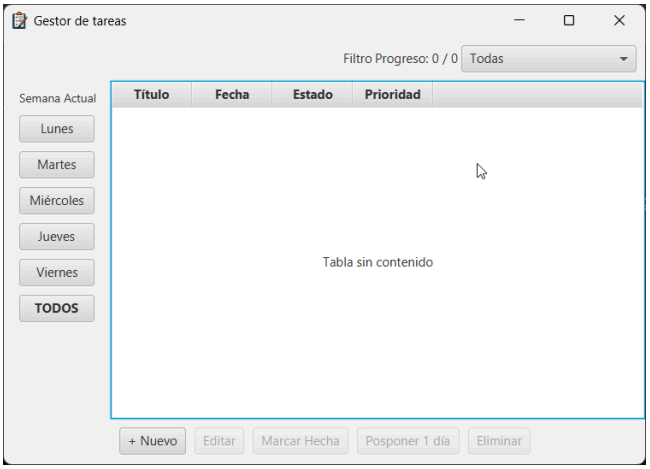
Filtros de visualización

El selector de Todas/Pendientes/hechas es un ComboBox que escucha cambios con y recalcula el `Predicate` de la `FilteredList`. Los filtros semanales marcan `diaActivo` y reutilizan el mismo método `aplicarFiltro()`, y el botón de Todos pone `diaActivo=NONE` y vuelve a ejecutar el filtro.

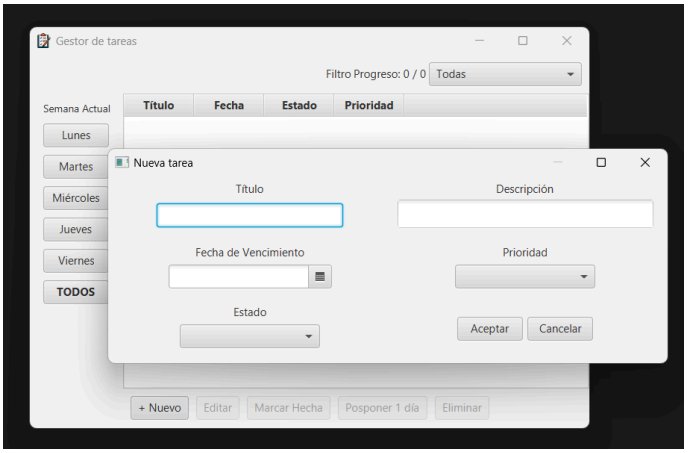
Mejoras de la experiencia de usuario

- Los botones de edición/borrado se deshabilitan cuando no hay fila seleccionada.
- Si el título se deja vacío al crear o editar una tarea, aparece una alerta.
- El progreso se actualiza tras cada operación.
- Se incluyen atajos de teclado, que están registrados en el `MainApp` para que sean globales a la ventana:
    - Ctrl + N -> nueva tarea
    - Supr (teniendo seleccionada una tarea en la tabla) -> borra la tarea con un aviso.
    - Ctrl + D (teniendo seleccionada una tarea en la tabla) -> es equivalente a pulsar “Posponer 1 día”

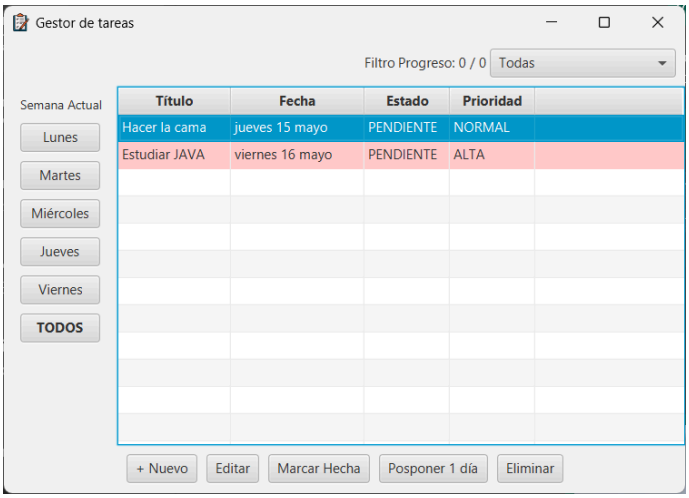
Interfaz principal



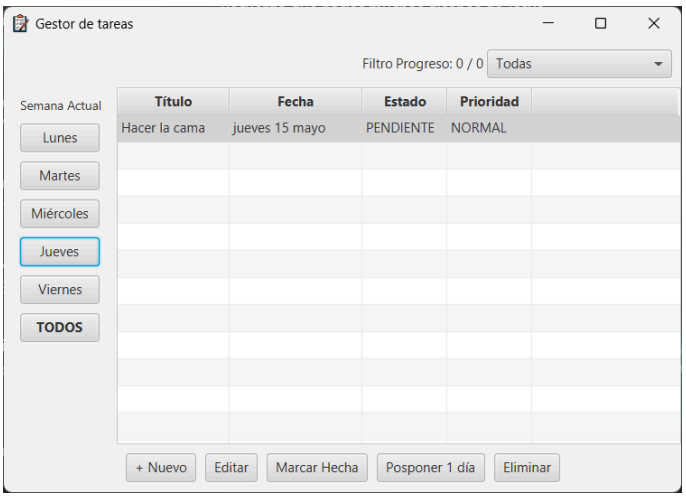
Diálogo de nueva Tarea



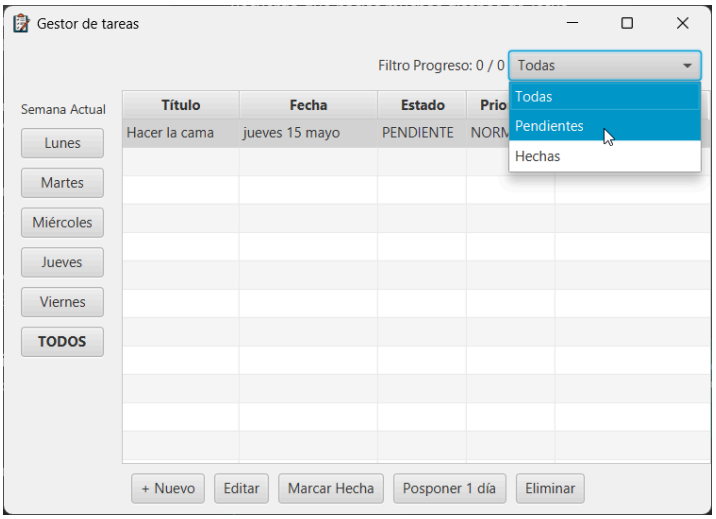
Ventana principal con actividades ya introducidas



Filtrado por dia de la semana



Diálogo de elección de filtro por estado



4 - Filtros y fechas

Durante las pruebas al inicio del desarrollo vi que habían dos problemas de usabilidad: las fechas eran poco legibles por el formato en el que se mostraban (aunque fueran correctas en España no las leemos en formato ISO) y el filtro pendientes / hechas era confuso. En esta fase refiné la presentación y unifiqué el mecanismo de filtrado.

Formato de fechas Español

```
private static final DateTimeFormatter FMT_FECHA = DateTimeFormatter
        .ofPattern("EEEE d MMMM", new Locale("es", "ES"));
```

- EEEE - Día completo (lunes, martes, miércoles…). No he conseguido que la primera letra sea en mayúscula.
- d - día del mes sin cero al inicio.
- MMMM - Nombre del mes (enero, febrero, marzo…). No he conseguido que la primera letra sea en mayúscula.

La conversión se aplica en el `CellValueFactory` de la columna `colFecha`:

```
colFecha.setCellValueFactory(c -> {
    LocalDate f = c.getValue().getVencimiento();
    String texto = (f != null) ? FMT_FECHA.format(f) : "";
    return new SimpleStringProperty(texto);
});
```

En el FXML se modificó la declaración:

```
<TableColumn fx:id="colFecha" text="Fecha" prefWidth="140"
             fx:type="java.lang.String"/>
```

y en el controlador la firma pasa de `TableColumn<Tarea, LocalDate>` a `TableColumn<Tarea, String>`.

ComboBox

```
comboFiltro.getItems().addAll("Todas", "Pendientes", "Hechas");
comboFiltro.valueProperty().addListener((obs, ant, act) -> {
    diaActivo = DiaFiltro.NONE;   // resetea filtro semanal
    aplicarFiltro();              // recalcula predicate
});
```

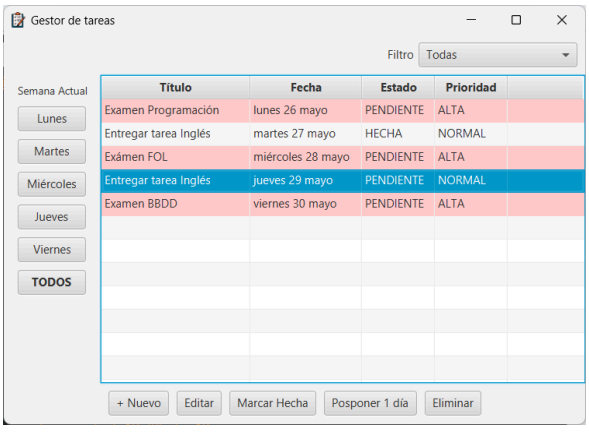
De esta manera cada vez que se cambia el estado se desactiva el filtro semanal para evitar combinaciones confusas.

Lógica única de filtrado

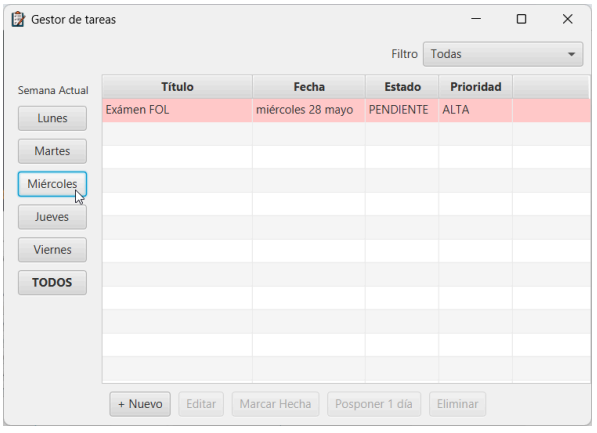
Para evitar duplicar código y garantizar que ambos filtros trabajen juntos se refactoriza a:

```
filtrada.setPredicate(t -> {
    /* --- filtro por estado --- */
    boolean estadoOK = switch (comboFiltro.getValue()) {
        case "Pendientes" -> t.getEstado() == Estado.PENDIENTE;
        case "Hechas"     -> t.getEstado() == Estado.HECHA;
        default           -> true;          // "Todas"
    };

    /* --- filtro por día de la semana --- */
    boolean diaOK = switch (diaActivo) {
        case LUNES      -> esDia(t, MONDAY);
        case MARTES     -> esDia(t, TUESDAY);
        case MIERCOLES  -> esDia(t, WEDNESDAY);
        case JUEVES     -> esDia(t, THURSDAY);
        case VIERNES    -> esDia(t, FRIDAY);
        default         -> true;            // NONE
    };

    return estadoOK && diaOK;
});
```

Vista de tareas sin filtrar



Vista filtrada por el día miércoles



5 - Vista Semanal

Con la introducción de este apartado se pretendía que el usuario pueda, además de filtrar por estado, ver las tareas por día de vencimiento de la semana actual, de manera inmediata.

Botonera lateral

Se añade un VBox anclado a la derecha del BorderPane principal con seis botones: uno por día de la semana de Lunes a Viernes y otro que quita el filtro semanal para mostrar todos.Todos los botones tienen el mismo tamaño `prefWidth = 75` para conservar una estética compacta.
Para evitar cinco métodos que eran casi iguales, en el FXML se pone el mismo `onAction` a los cinco botones y en el código se determina la fuente del evento:

```
@FXML
private void onFiltroDia(ActionEvent e) {
    Object src = e.getSource();
    diaActivo =
        (src == btnLun) ? DiaFiltro.LUNES :
        (src == btnMar) ? DiaFiltro.MARTES :
        (src == btnMie) ? DiaFiltro.MIERCOLES :
        (src == btnJue) ? DiaFiltro.JUEVES :
        (src == btnVie) ? DiaFiltro.VIERNES :
        DiaFiltro.NONE;        // nunca debería llegar aquí

    aplicarFiltro();           // recalcula la FilteredList
}
```

Para el botón Todos se crea otro handler llamado `onResetSemana`:

```
@FXML
private void onResetSemana() {
    diaActivo = DiaFiltro.NONE;
    aplicarFiltro();
}
```

Cuando se selecciona un día, `btnTodos` se desmarca y viceversa.

Actualización del método `aplicarFiltro()`

```
boolean diaOK = switch (diaActivo) {
```

```
    case LUNES      -> esDia(t, MONDAY);
    case MARTES     -> esDia(t, TUESDAY);
    case MIERCOLES  -> esDia(t, WEDNESDAY);
    case JUEVES     -> esDia(t, THURSDAY);
    case VIERNES    -> esDia(t, FRIDAY);
    default         -> true;           // NONE
};
```

La comprobación `esDia()` devuelve true si la fecha de vencimiento coincide con el `DayOfWeek` que se pide. Para tareas sin fecha el filtro semanal no las muestra, ya que no tiene sentido en la lista de lunes a viernes.

6 - Pruebas con JUnit

Antes de empaquetar había que comprobar que la capa de acceso a datos funcionase de forma fiable. Había que detectar fallos lógicos en el alta, actualización y borrado de tareas y garantizar que los cambios futuros no rompan otra funcionalidades que ya funcionasen.

Herramientas usadas

Se ha usado JUnit Jupiter y SQLite en memoria, por ser un framework estandar en java y por la rapidez y el aislamiento del fichero real. La dependencia de JUnit se declaró en el `pom.xml` con `scope=test`,

Adaptación de la aplicación

Para que el DAO pueda trabajar con una base de datos temporal se añade en `DbUtil`:

```
public static void setUrl(String nuevaUrl) { URL = nuevaUrl; }
```

Solo se usa desde los test, el código de producción mantiene la ruta `data/tareas.db`.

SQLite destruye la base de datos en memoria al cerrarse la última conexión. En la suite de pruebas se crea una conexión global (`keepAlive`) que se cierra en `@AfterAll`, compartiendo los tres test la misma tabla.

Estructura de la clase `TareaDaoTest`

```
@BeforeAll
static void initSuite() throws SQLException { ... }

@BeforeEach
void limpiarTabla() throws SQLException { ... }

@Test  void crear_y_listar()  { ... }
@Test  void actualizar()      { ... }
@Test  void borrar()          { ... }
```

- `initSuite()`
  - Define la URL en memoria
  - Ejecuta una sentencia `CREATE TABLE IF NOT EXIST` como la del código de producción
  - Instancia el `TareaDao`
- `limpiarTabla()`
  - ejecuta `DELETE FROM` tareas para garantizar que cada test empieza con una

tabla vacía

- `crear_y_listar()`
  - Inserta una tarea, llama a `dao.listar()` y comprueba que el tamaño es 1 y que el título coincide
- `actualizar()`
  - Inserta una tarea, recupera la fila real para obtener el `id`, cambia el título y su prioridad y llama a `dao.actualizar()` y vuelve a leer y verificar los valores nuevos.
- `borrar()`
  - Inserta dos tareas, borra la primera por `id` y lista de nuevo y comprueba que quede una y que corresponda a la segunda.

7 - Pulido de la interfaz de usuario

Una vez que las funcionalidades básicas de la aplicación estaban cubiertas y las validaciones realizadas, lo siguiente era revisar la experiencia de usuario al manejar la aplicación. Planteé varias opciones pero finalmente me quedé con cuatro: atajos de teclado, alertas y validaciones, señalización visual y personalización de la ventana.

Atajos de teclado

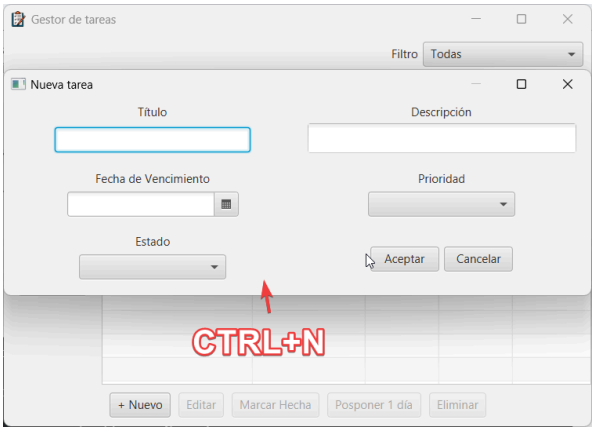
Introduje tres atajos para hacer más fácil la navegación en el programa:

- **Nueva tarea** - Ctrl + N - Alta rápida sin tocar el ratón
- **Borrar** - Supr (mientras se tiene una tarea seleccionada) - Atajo normal en windows para eliminar elementos.
- **Dejar para mañana** - Ctrl + D - Poder posponer una tarea sin tocar el ratón

En el MainApp:

```
Scene scene = new Scene(root);

scene.getAccelerators().put(
        new KeyCodeCombination(KeyCode.N, KeyCombination.CONTROL_DOWN),
        controller::onNueva);

scene.getAccelerators().put(
        new KeyCodeCombination(KeyCode.DELETE),
        controller::onBorrar);

scene.getAccelerators().put(
        new KeyCodeCombination(KeyCode.D, KeyCombination.CONTROL_DOWN),
        controller::onManiana);
```

Ejemplo de pulsación de atajo



Alertas y validaciones de entrada

Cuando se hacían determinadas acciones había que avisar al usuario para que confirmara:

- Confirmar borrado - `AlertType.CONFIRMATION` - Conlleva un texto para confirmar
- Título vacío - `AlertType.WARNING` - Con un texto avisando de que el título está vacío.

Señalización visual en la tabla

Prioridad alta:

```
tabla.setRowFactory(tv -> new TableRow<>() {
    @Override protected void updateItem(Tarea t, boolean empty) {
        super.updateItem(t, empty);
        setStyle(empty || t == null ? "" :
                 (t.getPrioridad() == Prioridad.ALTA ?
"-fx-background-color:#ffcccc;" : ""));
    }
});
```

El fondo rojo suave llama la atención sobre la prioridad ALTA.

Botones dependientes de selección:

```
tabla.getSelectionModel().selectedItemProperty().addListener((o,oldSel,n
ewSel)->actualizarBotones());

private void actualizarBotones() {
    boolean sel = tabla.getSelectionModel().getSelectedItem() != null;
    btnEditar.setDisable(!sel);
    btnHecha .setDisable(!sel);
    btnManiana.setDisable(!sel);
    btnBorrar.setDisable(!sel);
}
```

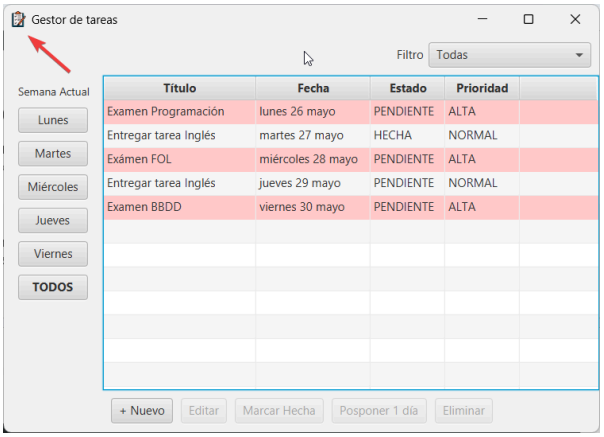
Personalización de ventana

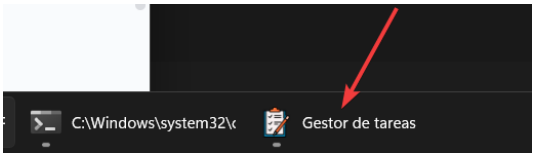
Para personalizar la ventana añadí un título fijo “Gestor de tareas” y un icono de 64px en `src/main/resources/es/dam/tareas/img/icono.png`

```
URL iconUrl = getClass().getResource("/es/dam/tareas/img/icono.png");
if (iconUrl != null)
    primaryStage.getIcons().add(new Image(iconUrl.toString()));
```

El icono aparece en la barra de título y en la barra de tareas de Windows.

Ejemplo de icono específico en la ventana principal





Ejemplo de icono en la barra de herramientas

8 - Empaquetado y distribución

Una vez que había depurado el código y la interfaz, y con las pruebas hechas, procedí a crear un archivo ejecutable que funcione en cualquier SO Windows con JRE 17 o posterior instalado, sin pasos extras de instalación.

Elección de formato

Tras ver distintas opciones, las que más me cuadraban eran:

- JAR “thin” + carpeta lib
    - Tiene un tamaño menor, pero requiere copiar varias DLL/JAR de JavaFX y leí que solía dar fallos, por la que lo descarté.
- Instalador MSI/EXE
    - Es la experiencia más cercana a un producto final, pero había que trabajar con WiX/Inno Setup y no me daba más tiempo a indagar en ello.
- JAR “fat”
    - Con este sistema todo va dentro del JAR y, aunque pesa más, ejecutando el script funciona rápido y además es multiplataforma (solo habría que hacer otro script que lo lance desde la plataforma deseada).

Usé maven-shade-plugin para crear un fat-jar que contiene clases de aplicación, dependencias JavaFX, driver SQLite y SLF4J y el manifest con Main-Class.

Configuración en pom.xml

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-shade-plugin</artifactId>
  <version>3.5.0</version>

  <executions>
    <execution>
      <phase>package</phase>
      <goals><goal>shade</goal></goals>

      <configuration>
        <!-- Nombre final del archivo -->
        <finalName>GestorTareas</finalName>

        <!-- Insertar Main-Class en el MANIFEST -->
        <transformers>
          <transformer implementation=

"org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
            <mainClass>es.dam.tareas.MainApp</mainClass>
          </transformer>
```

```
        </transformers>
      </configuration>
    </execution>
  </executions>
</plugin>
```

El plugin javafx-maven-plugin se mantiene para el desarrollo, pero no interviene en el empaquetado.

Generando el artefacto

```
mvn clean package
```

Con esto se crea un archivo de unos 32 MB.

Script de lanzamiento

Para simplificar el lanzamiento del programa se crea un run.bat para su ejecución con el siguiente contenido:

```
@echo off
REM Ejecuta el Gestor de Tareas
java -jar GestorTareas.jar
pause
```

Con lo que el usuario final tendría que descargar el archivo GestorTareas.zip, descomprimirlo y ejecutar el run.bat para correr el programa.

Posibles problemas

- El `JAR`, según he podido investigar,incluye librerias nativas solo para Windows x64, pero no se si es compatible con otros sistemas por la falta de acceso a ellos.
- Pese a que el tamaño es asumible, es elevado sobre todo si se quiere escalar el proyecto.
- No hay firma digital, por lo que Windows Defender podría bloquear su ejecución o mostrar una advertencia.

9 - Finalización del proyecto

Revisando el proyecto para hacer la memoria me doy cuenta de lo aprendido con la realización del mismo, entre otras cosas:

- Aplicar el patrón DAO con una base de datos, diseñadndo SQL limpio y encapsulado.
- Usar el bucle de escribir código -> probarlo -> refactorizar con JUnit y la base de datos en memoria.
- Hacer una interfaz en JavaFX que, después de los últimos pasos del proyecto, ofrece una experiencia de uso fluida con atajos, validaciones y que es agradable visualmente.
- Generar un JAR autoejecutable independiente de un IDE.

Los primeros pasos del proyecto me han obligado a tomar decisiones con respecto a qué hacer despues: elegir librerias, priorizar unas funcionalidades por encima de otras e incluso descartar algunas opciones que había elegido por poder llegar a tiempo a la entrega. Con ello creo haber llegado, al menos a una pequeña escala, a una situación de proyecto similar a lo que podría ser el ámbito profesional. Con estas elecciones, creo que podría llevar a cabo otros proyectos similares en características y requerimientos fuera del ámbito estudiantil.

Conclusiones y trabajos futuros

Los objetivos funcionales de la aplicación se han conseguido. Permite crear, editar, filtrar y conservar tareas en un archivo SQLite y se puede ejecutar en cualquier Windows con JRE.

Durante proceso del desarrollo tuve varios problemas, como escoger el flujo de trabajo y tecnologías que fueran adecuadas. Tras varias pruebas descarté CSS y use fat-jar para simplificar la puesta en producción. Me di cuenta, ya demasiado avanzado el proyecto, que mantener la documentación al día ahorra mucho en la fase de finalización del proyecto.

Enlo personal, este proyecto me ha supuesto el poder asimilar bien el patrón DAO, el uso de JUnit y la creación de la interfaz. Sobre todo me llevo la confianza de afrontar un proyecto desde cero.

En lo referente a las posibles mejoras que podría tener el programa, enumero:
· Añadir pruebas de UI con TestFX para evitar regresiones.
· Migrar a PostgreSQL si se necesitase escalabilidad o trabajo en grupo.
· Explorar las hojas de estilo y quizás un modo oscuro para mejorar la accesibilidad.
· Crear instaladores MSI para facilitar su distribución.

Bibliografía y referencias consultadas

1. **Oracle** - Java SE 17 Documentation: JDBC Guide
    - https://docs.oracle.com/en/java/javase/17/docs/api/java.sql/java/sql/package-summary.html
2. **OpenJFX Project** - JavaFX 20 Documentation & API Reference
    - https://openjfx.io/
3. **Gluon** - Scene Builder 17 User Guide
    - https://docs.gluonhq.com/scenebuilder/
4. **Xerial.org** - SQLite-JDBC Driver - User Guide
    - https://github.com/xerial/sqlite-jdbc
5. **JUnit 5 Team** - JUnit 5 User Guide
    - https://junit.org/junit5/docs/current/user-guide/
6. **Apache Maven Project** - Introduction to the POM
    - https://maven.apache.org/guides/introduction/introduction-to-the-pom.html
7. **Apache Maven Shade Plugin** - Shade Plugin – Usage.
    - https://maven.apache.org/plugins/maven-shade-plugin/
8. **DateTimeFormatter** - Documentación oficial.
    - https://docs.oracle.com/javase/17/docs/api/java.base/java/time/format/DateTimeFormatter.html
9. **Stack Overflow** - Preguntas y respuestas sobre FilteredList, combinaciones de teclas en JavaFX y empaquetado con jlink
    - https://stackoverflow.com
10. **Hilos** sobre buenas prácticas con DAO y pruebas en memoria con SQLite.
    - ForoProgramadores.net
11. **Consultas personales**
    - A. De La Fuente, desarrollador full-stack. Asesoramiento puntual sobre empaquetado fat jar y distribución en Windows.
    - M. Garrido, desarrollador full-stack. Revisión informal de los test JUnit y sugerencias de buenas prácticas de aserción.
12. **Inteligencia Artificial**
    - Uso ocasional de ChatGPT para aclarar dudas sintácticas de JavaFX, proponer ejemplos de DateTimeFormatter y validar comandos de Maven. Las respuestas se contrastaron siempre con la documentación oficial antes de incorporarlas al proyecto.