Deusto
Facultad de Ingeniería
Faculty of Engineering

Programación III & Programación de Aplicaciones

Curso 2025/2026

9 de enero de 2026

Examen parcial - 4,5 puntos (Duración: 2,5h)

Antes de comenzar a resolver el examen debes seguir los pasos indicados en la hoja de instrucciones

Este examen se enmarca en el ámbito de las estaciones de esquí, en concreto en la información del estado (apertura y clima), y el transporte en autobús entre ellas. En el examen se evalúan las competencias CE3 - JDBC y CE4 - Recursividad. Además, incluye tareas correspondientes a las competencias CE1 - Swing y CE2 - Threads, a modo de recuperación, para las personas que no obtuvieron al menos un 45% en cada una de ellas en el examen parcial de noviembre.

Para cada tarea, se indica su puntuación (sobre 4,5 puntos), la clase en la que debes añadir el código y una estimación del número de líneas de código. El código base está organizado en varios paquetes, cada uno de ellos relacionado con una competencia. En cada paquete encontrarás un método `main()` para probar tu código y ver el resultado: `MainJDBC`, `MainRecursividad`, `MainJSwing` y `MainThreads`. Por otro lado, dentro de la carpeta `resources/doc` encontrarás trazas de ejecución, imágenes y un vídeo que te ayudarán a ver con mayor detalle el resultado esperado de las tareas.

Tarea 1: Actualizar datos de una BBDD [1 punto] [GestorBD] [50-70 líneas]

El objetivo de esta tarea es la actualización de la información del estado de una estación de esquí. Usaremos una BBDD que está en `resources/db/esqui.db`. La BBDD tiene datos de ciudades, estaciones, estado de estaciones e itinerarios de autobuses. La tarea se centra únicamente en las siguientes tablas:

`ESTACION`: • `ID`: Integer (PK) • `NOMBRE`: Text • `CIUDAD_ID`: Integer (FK) • `KM_ESQUIABLES_TOTALES`: Real	`ESTADO_ESTACION`: • `ESTACION_ID`: Integer (PK & FK) • `CLIMA`: Text • `APERTURA`: Text • `TEMPERATURA`: Integer • `KM_ESQUIABLES_ACTUALES`: Real • `ACTUALIZACIÓN`: Integer (timestamp)

La clase **MainJDBC**, muestra una ventana en la que se puede seleccionar una estación de esquí en un JTree. Una vez seleccionada una estación, se muestran los datos de su estado en el panel central. Al pulsar el botón "Actualizar estado", se invoca el método **actualizarEstadoEstacion()** cuyo contenido debes implementar en la clase **GestorBD**. El método recibe como parámetros el nombre de una estación y un objeto **EstadoEstacion**. Con esos datos, debes actualizar la tabla **ESTADO_ESTACION**. Además, podría ocurrir que para una estación no existan datos en la tabla **ESTADO_ESTACION**; en ese caso, en vez de actualizar debes insertar los datos. Tienes que implementar toda la lógica necesaria dentro del método propuesto y para resolver la tarea **no puedes** invocar ningún otro método de la clase **GestorBD**.

Tarea 2: Recursividad [1 punto] [MainRecursividad] [30-50 líneas]

El objetivo de esta tarea es la generación de viajes de ida y vuelta tomando como referencia una estación de esquí. Un viaje es una sucesión encadenada de itinerarios. Los viajes deben cumplir estas restricciones:

- El viaje comienza y termina en una ciudad, y esa ciudad no puede repetirse en medio del viaje.
- El viaje debe pasar por los 3 países disponibles (Andorra, España y Francia).
- No se pueden repetir itinerarios en un mismo viaje.
- El número máximo de itinerarios de un viaje debe ser menor o igual que el número máximo indicado (*maxItinerarios*).

Debes implementar el proceso de generación de viajes en la clase **MainRecursividad**. Para ello, tienes un método denominado **generarViajes()** que recibe como parámetros una lista de itinerarios, el nombre de la ciudad origen y el número máximo de itinerarios que puede incluir un viaje.

Para comprobar que tu implementación es correcta, tienes el resultado esperado de la ejecución del método **main()** en el fichero **resources/doc/T2 – Recursividad.txt**.

La traza de ejecución muestra el método toString() de los itinerarios de cada viaje.

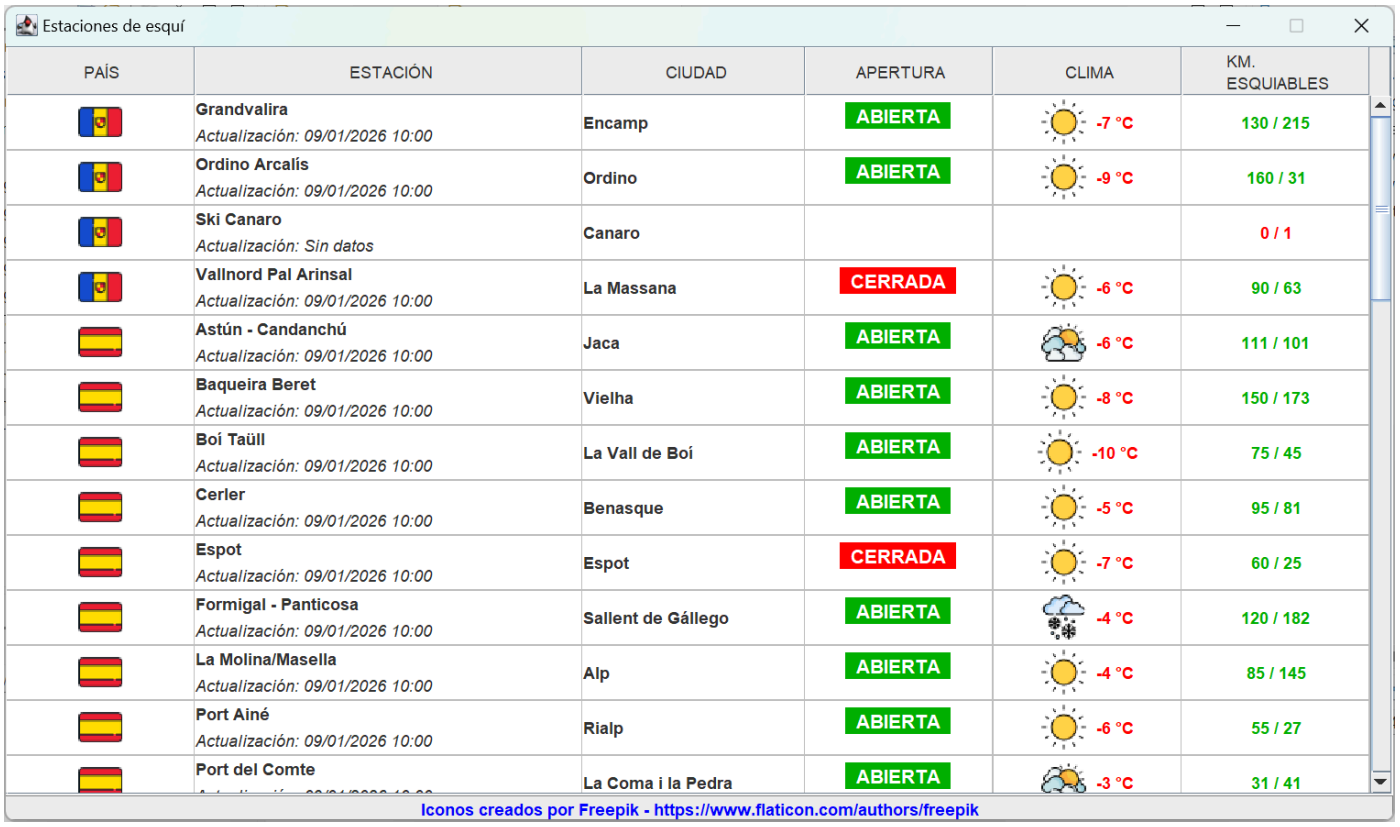
Recuperación de las competencias CE1 y CE2

A continuación, se describen 2 tareas que permiten evaluar de nuevo las 2 competencias evaluadas en el examen parcial. A pesar de que hayas liberado alguna de las dos competencias puedes intentar realizar las tareas para mejorar el resultado. La calificación definitiva para cada una de las dos competencias será la más alta entre la obtenida en el examen parcial de noviembre y este examen.

Tarea 3.1: JTable [1 punto] [MainJTable y EstacionTableModel] [50-60 líneas]

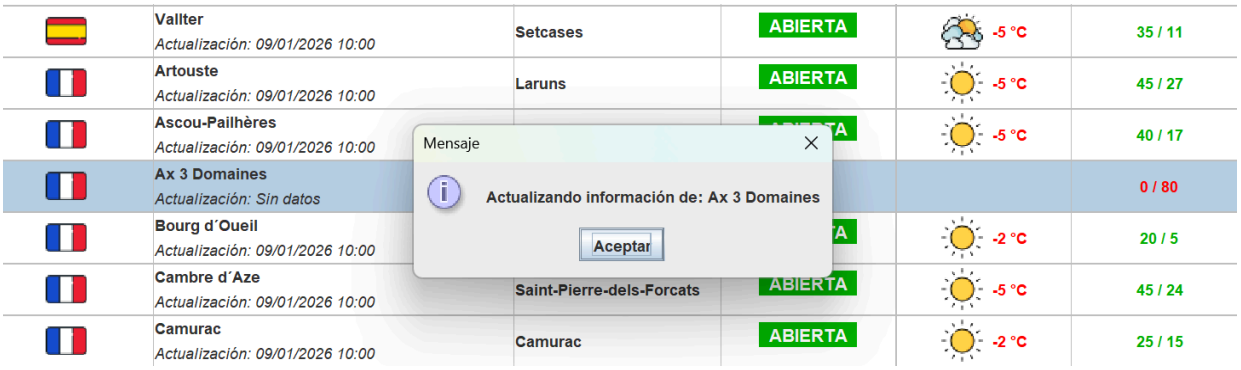
El objetivo de esta tarea es modificar el contenido y la visualización de la información de las estaciones de esquí que se muestran en un JTable. Para ello debes modificar el modelo de datos **`EstacionTableModel`** y ajustar el renderizado del JTable **`tablaEstaciones`**. El aspecto de la tabla debe parecerse lo más posible al mostrado en la siguiente imagen. Encontrarás los iconos en la carpeta **`resources/images`**.

Fíjate bien en las columnas ESTACIÓN y APERTURA no son simples etiquetas de texto.



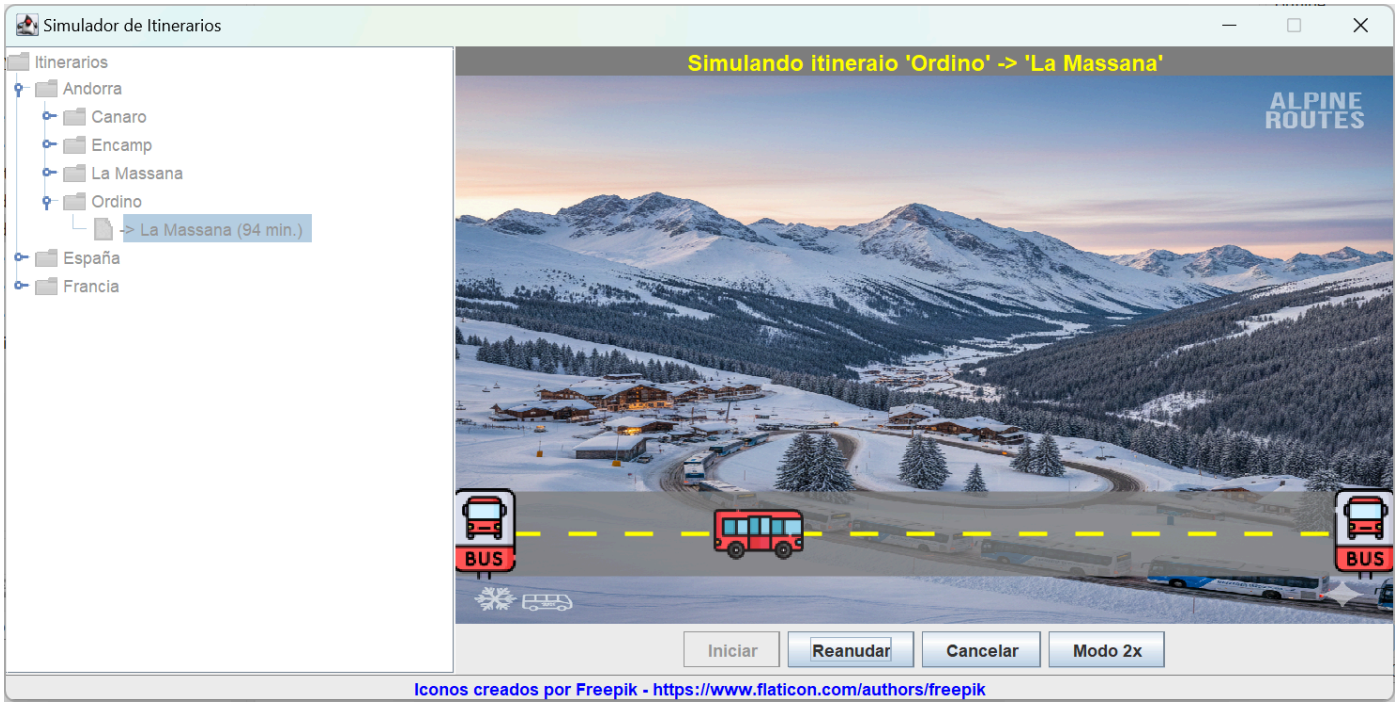
Tarea 3.2: Gestión de eventos [0,5 puntos] [MainJTable] [10 líneas]

El objetivo de esta tarea es mostrar un mensaje para simular la actualización del estado de una estación al pulsar la tecla "+". El mensaje sólo se muestra si hay una estación seleccionada en la tabla.



Tarea 4: Hilos [1 punto] [MainThreads] [50 líneas]

El objetivo de esta tarea es implementar un simulador de itinerarios. Hay una interfaz gráfica que permite la selección de un itinerario y muestra el desplazamiento del autobús.



Debes implementar la lógica de la simulación (utilizando hilos) que haga que el autobús se desplace de izquierda a derecha. Para ello el método **`actualizarProgreso()`** (que recibe un valor entre 0 y 100) coloca el autobús a lo largo del trayecto según el valor de progreso recibido. Con el fin de que el tiempo de la simulación sea proporcional a la duración del itinerario, el autobús se desplazará de acuerdo a un incremento de progreso que es igual a **`100 / duración`** (debes tener en cuenta este incremento para aumentar el valor de progreso). La velocidad siempre será constante y se simula durmiendo el hilo durante 200 mseg. en Modo 1x; y 100 mseg. en Modo 2x.

Además de implementar la lógica de la simulación, tienes que incorporar código en los Listeners de los 4 botones que controlan la simulación:

- **Iniciar**: activa la simulación.
- **Pausar** / **Reanudar**: pausa y reactiva la simulación.
- **Cancelar**: interrumpe la simulación.
- **Modo 2x** / **Modo 1x**: modifica la velocidad (alterando el tiempo que el hilo se duerme entre cada incremento del progreso).

Al terminar la simulación debes actualizar el mensaje de la parte superior indicando si la simulación se ha cancelado o ha terminado satisfactoriamente. Para ello tienes que usar el método **`actualizarMensaje()`**.

En la carpeta **`resources/doc`** tienes un vídeo que muestra el comportamiento esperado.