Examen Programación III - noviembre 2023

- Tienes en ALUD la base para realizar el examen. Descarga el ZIP, descomprímelo y crea un proyecto en Eclipse con la estructura de carpetas suministrada. Trabaja en una ruta que puedas localizar fácilmente en el ordenador (por ejemplo el Escritorio).
- Para entregar tu respuesta a través de ALUD, comprime la carpeta raíz del proyecto que has creado en Eclipse, asegúrate de que tiene los ficheros que has editado y súbelo en la tarea de ALUD.
- Tiempo de examen: **2,5h.**

Este examen se centra en las 2 primeras competencias de la asignatura:

- CE-01. Construir interfaces gráficas complejas personalizando el comportamiento y el aspecto visual de los componentes estándar de Java Swing.
- CE-02. Diseñar y manejar estructuras de datos basadas en Java Collections y Generics.

Una vez terminada la implementación de todas las tareas, podrás ver el estado de ocupación de los asientos del avión y además, podrás hacer el check-in de varios asientos de una misma clase. Para ello debes pulsar la combinación de teclas CTRL + C y desde el cuadro de diálogo emergente buscar el número de asientos que desees de una determinada clase pulsando el botón "Find Seats". Si hay disponibilidad de ese número de asientos, la aplicación te hará una propuesta que podrás confirmar con el botón "Confirm". Si confirmas la propuesta, los asientos quedan reservados y aparecen ocupados.

1. Tareas

Sobre este código, debes realizar una serie de tareas. Entre corchetes encontrarás la puntuación de cada tarea sobre 10 (+1 extra) , las clases en las que debes añadir código y el número aproximado de líneas.

T1. Crea un modelo de datos para un JTable [2 puntos] [Debes crear una nueva clase, 40-50 líneas y cargar ese modelo en JFrameCheckIn.java]

Si ejecutas el programa principal, verás que el JTable seatsTable se inicializa completamente vacío. Debes crear un modelo de datos para mostrar la información de los asientos fila a fila. Adapta la carga del modelo a los asientos del avión: el que se usa como referencia tiene 38 filas, cada una de ellas con 6 asientos identificados con letras de la A a la F: **suponemos que todos los aviones tienen estos 6 asientos por cada fila**. Por lo tanto, el modelo de datos debe permitir cargar los asientos por filas (en el ejemplo, desde la 1 a la 38) y devolver la información en 8 columnas: la columna 0 devuelve el número de fila (Integer), las columnas 1-3 los asientos de la A a la C (Seat), la columna 4 será una columna vacía para simular el pasillo (String vacío) y finalmente las columnas 5-7 los asientos de la D a la F (Seat). Haz que el constructor del modelo reciba una lista List<Seat> de los asientos para inicializarlo.

La fila de tabla 0 corresponde a la fila 1 del avión, y así sucesivamente. Todas las celdas deben ser no editables.

Fíjate cómo se crean los asientos en el método initPlane() de la clase Main.java para saber el orden en que están almacenados en la lista List<Seat> seats de la clase Aircraft.java.

Define las cabeceras con los textos que se muestran en esta figura:

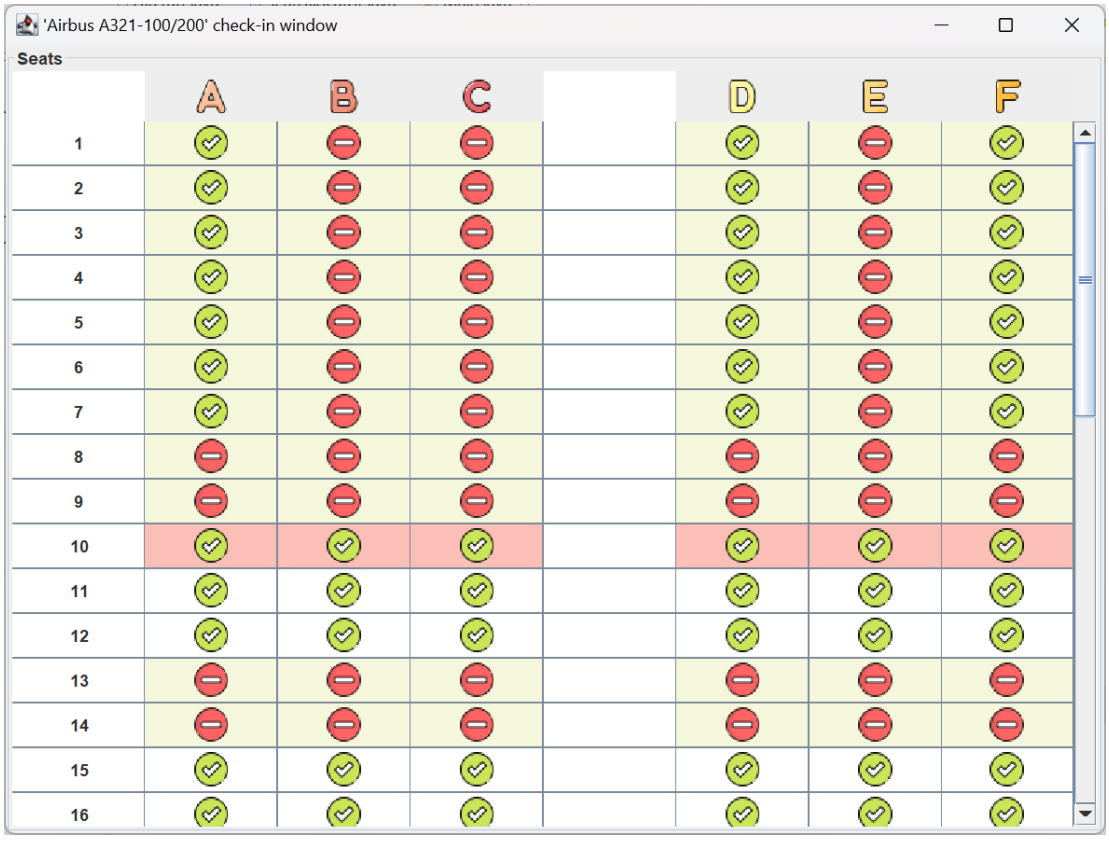
	A	B	C		D	E	F
1	1A	1B	1C		1D	1E	1F

2	2A	2B	2C		2D	2E	2F
…	…	…	…	…	…	…	…

T2. Implementa un Renderer personalizado para un JTable [2 puntos] [JFrameCheckIn.java, 50-60 líneas]

Debes modificar la manera en que se dibuja la tabla para que los datos de los asientos se visualicen como en la imagen:

- En la cabecera debe aparecer la imagen de las letras con color de fondo RGB(237, 237, 237). Encontrarás las imágenes en resources/images/A.png (B.png, C.png, etc.)
- La 1.ª y 4.ª columna de la cabecera aparecen vacías y con el color de fondo por defecto de la tabla.
- En la 1.ª columna de datos aparecen los números de fila con el color de fondo por defecto de la tabla.
- Los asientos en las filas FIRST_CLASS y EMERGENCY tienen colores de fondo personalizados RGB(245, 247, 220) y RGB(252, 191, 183) respectivamente.
- Cada asiento se pinta con la imagen Occupied.png o Available.png según se encuentre (en resources/images/Available.png y Occupied.png).
- La columna 4ª, que representa el pasillo, debe aparecer vacía con el color de fondo por defecto de la tabla.
- La altura por defecto de todas las filas de la tabla debe ser 32 píxeles.



T3. Implementa un evento de teclado [1 punto + 1 extra] [JFrameCheckIn.java, 10 líneas]

Al pulsar la combinación de teclas **CTRL + C** cuando el foco se encuentre en la tabla, debes abrir el cuadro de diálogo JDialogSeatAllocator para buscar y confirmar asientos (será suficiente con llamar al constructor de esta ventana de diálogo. El cuadro de diálogo tiene toda la funcionalidad implementada y hace uso de los métodos de la clase SeatAllocator.

- **Extra. Fuerza el repintado de un componente** [1 punto] [JDialogSeatAllocator.java, 5 líneas]: Al cerrar el cuadro de diálogo, la ocupación de los asientos que se hayan reservado no se actualiza

automáticamente. Añade el código necesario en el cuadro de diálogo para que cada vez que se confirme la sugerencia de asientos, se actualice la tabla de manera automática.

NOTA: Para comprobar la implementación correcta de cada una de las siguientes tareas utiliza los programas principales que tienes disponibles en *es.deusto.ingenieria.prog3.checkin.domain.test*. Los 3 programas funcionan aunque haya errores en las tareas relacionadas con la interfaz gráfica.

T4. Inicializa un mapa a partir de una lista [1 punto] [SeatAllocator.java, 10 líneas]

A partir de la lista de asientos `List<Seat> seats` debes crear el mapa `Map<Integer, List<Seat>>` que agrupa los asientos por número de fila. Este mapa se utilizará como una estructura de datos auxiliar para buscar los asientos recomendados al realizar el Check-in.

T5. Crea una estructura de datos compleja a partir de otra [2,5 puntos] [SeatAllocator.java, 35-40 líneas]

A partir de una clase de asiento (`FIRST_CLASS / EMERGENCY / ECONOMY`) y un número de asientos, debes crear una lista que contenga listas de asientos contiguos (no ocupados) en una misma fila.

Debes procesar los asientos de cada fila (usando el mapa `seatsMap` que has inicializado en la tarea anterior) para ir agrupando los asientos libres que estén juntos. Tienes que agrupar los asientos contiguos de cada fila e ir guardando los grupos cuyo tamaño sea mayor o igual al número de asientos necesarios. Es decir, si necesitas 3 asientos pero en una fila todos los asientos están libres, guardas un grupo con los 6 asientos libres. Por otro lado, si necesitas dos asientos y en una fila están ocupados los dos asientos del pasillo (C y D), en ese caso, guardas dos grupos de asientos: (A, B) y (E, F). Para realizar este proceso, ten en cuenta que los asientos de cada fila son todos de la misma clase, están ordenados de la A a la F y que los asientos del pasillo (C y D) se consideran contiguos.

Si no existe ningún grupo de asientos que cumpla las condiciones de la clase y el número de asientos, la lista devuelta estará vacía.

Por ejemplo, si se piden 2 asientos de FIRST_CLASS, se devolvería la lista de listas [[1C,1D], [2C,2D], [3C,3D], [4C,4D], [5C, 5D], [6C, 6D], [7C,7D]], y si se piden 3 de FIRST_CLASS, lista vacía.

T6. Ordena una lista usando 2 criterios [1,5 puntos] [SeatAllocator.java, 10-15 líneas]

Debes ordenar una lista de listas de asientos List<List<Seat>> siguiendo estos criterios:

- Criterio 1: De menor a mayor número de asientos de la lista.
- Criterio 2: Si dos listas coinciden en el número de asientos, ordenarlas de mayor a menor número de fila de los asientos de la lista.