CASO PRÁCTICAS

Pizarra de navegación

Interfaces Persona Computador

IPC – DSIC

UPV

Curso 2021-2022

Índice

Tablas

Figuras

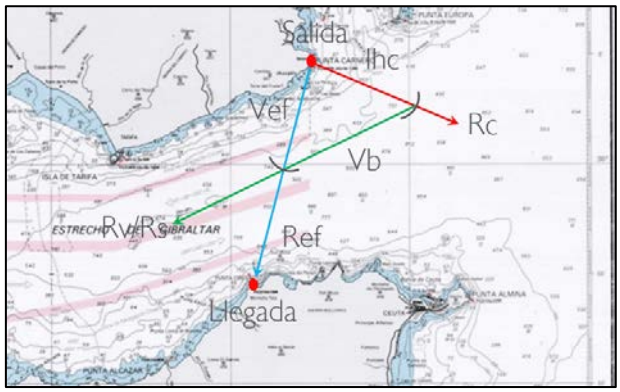
1. Caso de Estudio

Se desea desarrollar una herramienta para facilitar la preparación de los ejercicios de navegación sobre la carta náutica del estrecho de Gibraltar en los exámenes de patrón de embarcación de recreo. A continuación, describimos como es este proceso.

Los exámenes de esta titulación son de tipo test, al alumno se le propone un enunciado de un problema y se les dan 4 respuestas, siendo una de ellas cierta.

> 32.- Desde la situación l= 35º45,0´N L= 006º09,8´W navegamos al rumbo efectivo Ref = 042º durante una hora, hasta la nueva situación l= 35º 52,6´N – L= 006º 01,5´W ¿Qué distancia hemos navegado entre las dos situaciones?
>
> A) d= 12,6´
> B) d= 10,2´
> C) d= 9,2´
> D) d= 11,2´

El alumno utiliza la carta náutica para hacer los cálculos que necesite y así poder escoger la respuesta correcta.



Sobre la carta náutica el alumno marca puntos, traza líneas entre dos puntos, circunferencias (cuyo centro está en un punto marcado). También puede medir distancias entre dos puntos que después utilizará para evaluar su correspondencia en la escala vertical de la carta. Además, puede escribir sobre la carta su cálculos intermedios o anotaciones útiles para resolver el problema. Para todo ello dispone de una carta, un lápiz, una goma de borrar, una regla para trazar líneas y tomar distancias entre dos puntos, un compás con el que dibujar arcos, y un transportador de ángulos cuadrado.

Así pues, la aplicación a desarrollar tendrá dos grupos de funcionalidades. Por una parte, facilitará al alumno diversos enunciados de problemas con su correspondiente batería de respuestas, y recogerá los aciertos y errores que cometa el alumno al responderlos. Para ello el usuario creará un perfil, y cada vez que inicie una sesión la aplicación le propondrá problemas que el usuario deberá de resolver. Al finalizar la sesión se guardarán los aciertos y errores que se hayan cometido durante la sesión. Esta información almacenada se podrá visualizar a posteriori para ver la evolución en el aprendizaje del alumno.

Por otra parte, se dotará de la funcionalidad necesaria para que pueda pintar sobre una carta digital puntos, líneas, arcos, y escribir anotaciones. Con la herramienta queremos dotar de un entorno digital similar al entorno físico en el que el alumno realizará el examen, es decir disponer de las mismas herramientas físicas que utilizará el día del examen, pero en formato digital.

Seguidamente se detallan los escenarios de uso que se han obtenido tras el análisis de requisitos del sistema y que deben utilizarse para diseñar e implementar adecuadamente la aplicación requerida. Estos escenarios se han agrupado en dos categorías, los que hacer referencia al usuario registrado y aquellos que se derivan del uso de las herramientas de dibujo sobre la carta.

2. Escenarios sobre el usuario

2.1. Registrarse en la aplicación

Juan se quiere preparar el examen de patrón de embarcación de recreo. En la academia que se ha apuntado, el profesor utiliza una herramienta digital que está disponible para descargar. Tras instalarla y para poder acceder al conjunto de problemas resueltos disponibles accede a la opción de registrarse.

El sistema le solicita que introduzca un nombre de usuario, que será utilizado para identificarle, una cuenta de correo electrónico válida, una contraseña, su fecha de nacimiento y si lo desea, una imagen para ser utilizada como avatar. Juan introduce como nombre de usuario "jgarcia". Como contraseña, elige "passPER21!" y proporciona la cuenta de correo: "jgarcia@gmail.com". Finalmente, introduce una fecha de nacimiento unos años menor a su edad real. Como no le apetece buscar un avatar en ese momento, se queda con el que se le ofrece por defecto. Tras introducir toda la información pedida, solicita finalizar el proceso.

El sistema comprueba los datos introducidos y le indica que ese nombre ya está siendo utilizado y debe seleccionar un nuevo valor que contenga entre 6 y 15 caracteres o dígitos si espacios, pudiendo usar guiones o sub-guiones. Juan decide seleccionar el *nickname* "jpgarcia".

El sistema comprueba que el resto datos introducidos son válidos. La contraseña contiene entre 8 y 20 caracteres, incorpora al menos una letra en mayúsculas y minúsculas, así como algún dígito y algún carácter especial (!@#$%&*()-+=). El correo electrónico tiene un formato válido y el usuario tiene más de 16 años. El sistema registra al usuario y le informa de ello.

2.2. Autentificarse

Juan tiene un rato libe y piensa en hacer unos ejercicios de navegación antes de ir a la academia. Abre la aplicación y selecciona la opción de autenticarse en el sistema. El sistema le solicita que introduzca su nombre de usuario o *nickname* y su contraseña. Juan introduce su nickname y su contraseña. El sistema comprueba que existe un usuario con esos datos y lo autentifica dándole acceso al resto de funciones.

2.3. Cerrar sesión

Juan están cansado de hacer ejercicios, y Susana quiere probar a ver si es capaz de hacer bien 4 ejercicios seguidos, en lugar de cerrar la aplicación, Juan cierra su sesión para que Susana pueda arrancar la suya.

2.4. Realizar un problema

Juan, tras autenticarse, le pide al sistema que le proponga un problema. Hoy no quiere pensar mucho y escoge que el sistema la proponga un problema de manera aleatoria. El sistema le propone un problema, le muestra el enunciado y las 4 respuestas. Antes de mostrar las respuestas el sistema siempre las ordena de manera aleatoria.

Después de realizar sus cálculos sobre la carta, Juan marca la respuesta que considera la correcta y le pide al sistema que verifique la solución.

Otro día Juan le pedirá al sistema que le muestra la lista de problemas y escogerá uno el mismo.

2.5. Modificar perfil

Juan no eligió un avatar cuando se dio de alta, pero se ha cansado de aparecer con el avatar por defecto, así que decide acceder a la opción de modificar los datos de su perfil. El sistema le ofrece la información que tiene actualmente: su nombre de usuario, su contraseña, su dirección de correo electrónica y su fecha de nacimiento, así como el avatar que tiene asignado. Juan se da cuenta que puede modificar cualquier dato excepto su nombre de usuario. Decide cambiar su avatar entre algunas imágenes que tiene en su máquina. Tras realizar el cambio, solicita que se actualice la información.

El sistema comprueba que todos los cambios introducidos cumplen con los requisitos establecidos. La contraseña contiene entre 8 y 20 caracteres, incorpora al menos una letra en mayúsculas y minúsculas, así como algún dígito y algún carácter especial (!@#$%&*()-+=). El correo electrónico tiene un formato válido y el usuario tiene más de 12 años. Por tanto, el sistema actualiza la información e informa al usuario.

2.6. Mostrar resultados

Juan quiere saber cómo está evolucionando en la materia así que, después de autenticarse, le pide a la aplicación que le muestre la información sobre los aciertos y errores que ha cometido. El sistema le muestra sus datos de forma estructurada. Juan quiere ver esa información, pero solo en los últimos días, así que filtra los datos a mostrar indicando el día a partir del cual quiere que se muestren los resultados.

3. Escenarios sobre la carta

Al igual que ocurre en el entorno real en el que el usuario dispone de lápices de diferentes colores y puntas de diferente grosor, en las acciones que se van a describir a continuación y que así lo requieran, debe de estar la opción de escoger color y anchura del trazo.

3.1. Realizar zoom

Juan está realizando un ejercicio, pero no ve con detalle en nombre de un faro sobre la carta, así que hace una aproximación al detalle o zoom para acercar la vista. Después de leer el dato, decide volver a otra vista más alejada y hace un zoom para alejar la vista.

3.2. Marcar un punto

Juan decide marcar un punto sobre la carta. Utiliza la herramienta de puntos y selecciona la posición sobre la carta. El sistema muestra el punto en la forma y color previamente establecido.

3.3. Trazar una línea

Juan quiere trazar una línea que pasa por dos puntos, así que accede a la herramienta de trazado de líneas. Tras seleccionar el punto de origen y el punto final, la línea queda marcada con el grosor y color preestablecido.

3.4. Trazar un arco

Juan quiere trazar un arco así que accede a la herramienta de trazado de arcos y, tras seleccionar el centro del arco y su radio, este queda marcado con el grosor y color preestablecido.

3.5. Anotar texto

Juan quiere anotar texto sobre la carta. Selecciona la herramienta de texto y, tras escoger el lugar donde se mostrará, escribe el texto correspondiente. Al finalizar, el sistema muestra el texto con el color y tamaño preestablecido.

3.6. Cambiar el color de una marca

Juan quiere cambiar el color de una marca (punto, línea, arco, texto), así que selecciona el color y la marca, y el sistema muestra la marca con la nueva apariencia.

3.7. Eliminar una marca

Juan no está contento con la última línea pintada, así que la selecciona y la elimina de la carta. En otro momento elimina también un punto, un arco y un texto.

3.8. Limpiar la carta

Juan ha resuelto el problema y quiere limpiar la carta para poder resolver otro problema. Escoge la opción de limpiar y el sistema le presenta una carta sin ninguna marca.

3.9. Desplazar el transportador para medir ángulos. (Tomar un ángulo / trazar línea con un ángulo)

Juan quiere saber el ángulo que forma una línea, para ello selecciona el transportador, desplaza el transportador por la carta hasta situar su centro sobre la línea. Una vez visualiza el punto de corte de la línea sobre el transportador, ya sabe cuál es el ángulo que forma esta línea sobre el norte. Poco después quiere trazar una línea desde un faro con un ángulo determinado. Para ello sitúa el centro del transportador en el faro. Para trazar una línea solo tendrá que marcar un punto en la carta próximo a la medida deseada del transportador

Tomar un ángulo:

Podemos indicar que tomar un ángulo es una tarea compuesta de varios de los escenarios descritos en el punto 3, es decir, para anotar el ángulo de una línea primero tendrá que pintar la línea, después desplazar el transportador y por último anotar el texto que considere. En el siguiente enlace podéis acceder a un video en el se muestra un ejemplo de esta tarea de alto nivel: anotar ángulo

Trazar línea ángulo:

De manera similar al caso anterior, en el siguiente video mostramos una tarea de alto nivel que utiliza varios escenarios: trazar línea ángulo

3.10. Marcar extremos de un punto en la carta

Juan quiere saber la latitud y la longitud de un punto marcado en la carta. Después de seleccionar la herramienta adecuada y seleccionar el punto el sistema, pinta sobre la carta dos líneas ortogonales que pasan por el punto y que cortan perpendicularmente los bordes de la carta.

4. Recursos para realizar la práctica

Para realizar la práctica se proporciona el fichero "carta_nautica.jpg" que se utilizará como base para realizar los diferentes casos de uso.

También se proporciona el fichero "transportador.jpg" como transportador cuadrado de ángulos. Para que este transportador resulte útil, es necesario que se le dote de transparencia. Los nodos en javaFX tienen la propiedad **Opacity** que permite variar su opacidad y hacerlos transparentes. No es obligado utilizar este recurso, el alumno puede utilizar otras estrategias o fuentes para disponer de un transportador de ángulos transparente.

5. Librería proporcionada para la persistencia de los datos

La persistencia de la información se va a realizar a través de una base de datos *SQLite*[1]. En la librería de acceso (*Navegacion.jar*) se define el modelo de datos a utilizar en la práctica. Esta librería os permitirá almacenar y recuperar toda la información que necesite persistencia. En concreto la aplicación mantendrá un conjunto de problemas de navegación con sus respuestas. También guardará la información de los usuarios, sus datos y de cada sesión iniciada se guardará también el número de problemas resueltos correctamente y el número de problemas no resueltos adecuadamente.

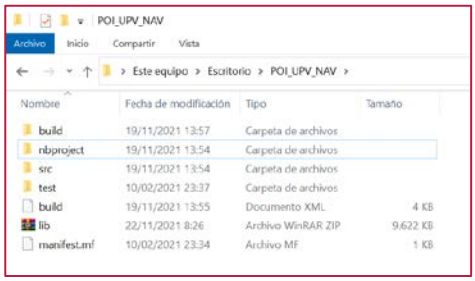
La librería gestiona todo el acceso a la base de datos, creándola de forma automática si es necesario dentro del directorio de vuestro proyecto, en un fichero denominado "database.db". Si el sistema crea la base de datos esta estará vacía por lo que no se dispondrán de problemas que mostrar al usuario. Os vamos a dejar una base de datos que contiene un conjunto de problemas y un usuario genérico para poder probar vuestro trabajo, en el caso de que todavía no tengáis definido el registro de usuarios.

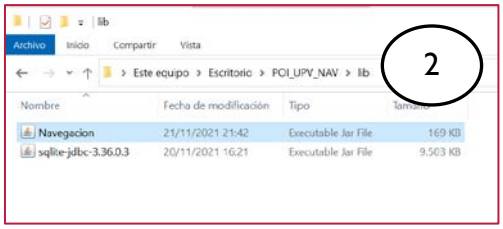
Tal y como se explicará a continuación, se os proporciona un método capaz de volcar a un fichero de texto el contenido de la base de datos, por si necesitáis verificar su contenido durante el desarrollo del proyecto. No obstante, existen numerosas aplicaciones que os permiten abrir una base de datos *SQLite* y ver el contenido de sus tablas. Por ejemplo, podéis instalaros la aplicación gratuita *"DB Browser for SQLite"*[2].

5.1. Agregar la librería Navegacion.jar a tu proyecto

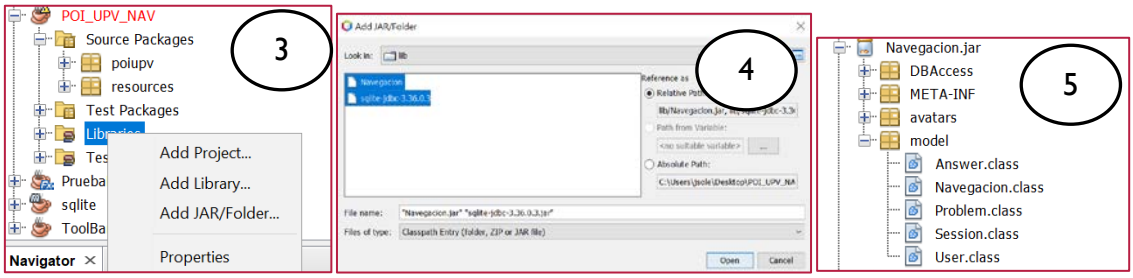
En primer lugar, descargad y guardad dentro de la carpeta de vuestro proyecto el fichero *lib.zip,* que contiene las librerías necesarias(1). Descomprimid el contenido en ese directorio, de forma que se creará una nueva carpeta *lib*, con dos ficheros .jar: *navegacion.jar y sqlite-jdbc-3.30.1.jar.* (2).

[1] https://www.sqlite.org/index.html
[2] https://sqlitebrowser.org/



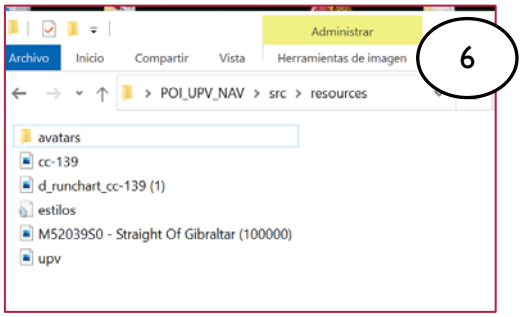


A continuación, añadiremos estas librerías a vuestro proyecto. Los proyectos creados por Netbeans para aplicaciones java tienen una carpeta `Libraries` donde añadir las librerías externas que se desee. Para ello, basta situarse sobre esa carpeta, y desde el menú contextual (botón derecho del ratón), seleccionar la opción `Add JAR/Folder`, tal y como se muestra en (3).



En el diálogo Add Jar/Folder(1), selecciona la carpeta lib que hemos creado anteriormente. Selecciona ambos ficheros .jar y pulsa sobre el botón *Open (4)*. Netbeans añadirá las librerías al proyecto y como puedes observar en (5).

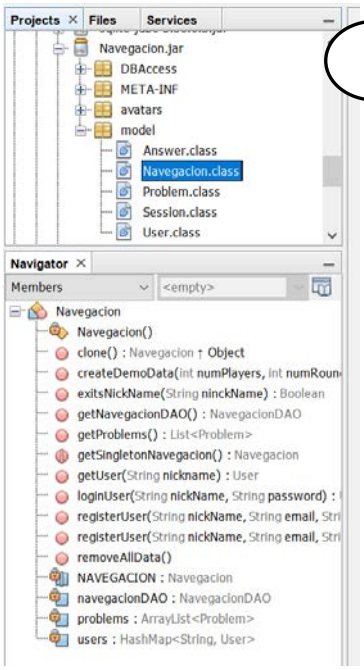
La librería necesita también que descarguéis el fichero avatars.zip y lo descomprimáis dentro de la carpeta src del proyecto, tal y como se muestra en (6).



5.2. API proporcionado por *Navegacion.jar*

La librería proporciona diferentes clases, de las que solo necesitaréis usar aquellas que están dentro de la carpeta *model* (7). Si os situáis sobre cualquiera de las clases proporcionadas y hacéis doble clic, se desplegará en la pestaña *Navigator* todos los métodos públicos que proporciona, tal y como podéis apreciar en la Figura 4. A

continuación, describiremos los aspectos más importantes de las clases proporcionadas.



5.2.1. Clases del modelo

Navegacion

La clase *Navegación* permite la inserción y recuperación de toda la información que necesitaréis en vuestro proyecto. Esta clase se encarga de conectarse a la base de datos (BD) y cargar en memoria toda la información que contiene, así como de almacenar la nueva información y cambios que se realicen. Esta clase implementa el patrón *Singleton*, por lo que sólo puede instanciarse un objeto de esta clase en una aplicación. Para obtener este objeto debéis llamar al método estático **getSingletonNavegacion()**.

getSingletonNavegacion

```
public static Navegacion getSingletonNavegacion ()
```

A partir de este objeto podéis acceder a los métodos que permiten acceder a la información del sistema. La clase Navegación mantiene una estructura del tipo Map con los usuarios registrados en la aplicación (objetos del tipo User). Para acceder a los datos de un usuario o almacenar un usuario nuevo hay que hacer uso de los métodos públicos que se describen continuación.

loginUser

```
public User loginUser(String nickName, String password)
```

Este método recupera de la BD el objeto user con el *nikcName* y *password* introducidos como parámetros, si no existe retorna null. Antes de invocar a este método es conveniente comprobar que existe el usuario en el sistema.

existsNickName

```
public Boolean exitsNickName(String ninckName)
```

Este método comprueba si existe en el sistema el usuario con nickName introducido como parámetro.

registerUser

```
public User registerUser(String nickName, String email, String password, Image avatar,
LocalDate birthdate)
```

Este método se utiliza para crear un objeto del tipo User y registrarlo en el sistema, de tal manera que se guardará en la BD, en este caso hay que proporcionar todos los atributos del usuario. Hay un método alternativo en el que no se introduce el avatar. Aunque puedes crear objetos del tipo User invocando al constructor de la clase, estos objetos no se almacenarán en la BD, es por ello que recomendamos que no crees usuarios sino que registres usuarios.

getProblems

```
public List<Problem> getProblems()
```

Devuelve una estructura de tipo lista con los problemas de navegación almacenados en la BD (objetos del tipo Problem). En este caso la gestión es muy sencilla pues la única funcionalidad disponible en la librería es la de obtener la lista inmutable con todos los problemas almacenados. Ya es trabajo vuestro como mostráis los problemas al usuario, de manera aleatoria o dejando que este escoja un problema de la lista.

User

La clase User permite manejar la información de los usuarios del sistema, ofreciendo métodos *getters* y *setters* para acceder a sus atributos. Los métodos *setters* actualizan la información en la base de datos, **siempre y cuando** el objeto haya sido creado usando el método Navegacion.*registerUser()*. Por tanto, **no debe utilizarse el constructor público** ofrecido por la clase si se desea que el usuario esté almacenado en la base de datos y sea manejado por el sistema.

Los atributos de User son:

- String **nickName**: nombre de usuario. No puede ser actualizado.
- String **email**: dirección de correo electrónico del usuario.
- String **password:** contraseña del usuario.
- Image **avatar**: imagen de avatar del usuario.
- LocalDate **birthdate**: fecha de nacimiento del usuario.
- ArrayList<Session> **sessions**: una lista con todas las sesiones que ha iniciado el usuario y en la que se guarda el número de problemas resueltos correctamente y el número de problemas no resueltos correctamente.

Además de los setter y getters, la clase *User* proporciona una serie de métodos útiles para validar información y actualizar la información de sus atributos que os mostramos en la Tabla I.

Tabla 1. API de la clase User

`public Boolean checkCredentials(String nickName, String password)`	Devuelve true si el nombre de usuario y la contraseña proporcionados coinciden con la del usuario.
`public String toString()`	Extrae la información del usuario y la devuelve en una cadena de texto con el formato atributo = valor
`public static Boolean checkEmail (String email)`	Método estático que permite comprobar si el dato proporcionado corresponde con una cuenta de correo válida.
`public static Boolean checkNickName(String nickname)`	Método estático que permite comprobar si el dato proporcionado corresponde con un nombre de usuario válido. Un *nickname* es válido si tiene entre 6 y 15 caracteres y contiene letras mayúsculas, minúsculas o los guiones '–' y ' _ ' .
`public static Boolean checkPassword(String password)`	Método estático que permite comprobar si el dato proporcionado contiene una contraseña válida. Una contraseña es válida si: - contiene ente 8 y 20 caracteres - contiene al menos una letra mayúscula - contiene al menos una letra minúscula - contiene al menos un dígito - contiene un carácter especial del conjunto: !@#$%&*()-+= - no contiene ningún espacio en blanco

Session

La clase Session almacena la información de cada sesión realizada por el usuario, ofreciendo métodos *getters* para acceder a sus atributos. Los atributos de un Session no pueden ser modificados tras haberse creado. Es por ello que el objeto se deberá de crear en el momento que el usuario cierra la sesión, bien de manera voluntaria o porque se cierra la aplicación, es en este momento cuando se conoce el número de aciertos o fallos durante la sesión. Para poder almacenar la sesión es necesario invocar al método de la clase User:

```
public void addSession(Session session)
```

Los atributos de la clase Session son:

- LocalDateTime **timeStamp**: día y hora en el que se inicia/registra la sesión.
- int **hits**: número de problemas resueltos correctamente
- int **faults**: número de problemas resueltos incorrectamente

Problem

La clase Problem se utiliza para guardar los problemas de navegación. En la base de datos se incluye un conjunto de problemas por lo que no es necesario que creéis objetos de este tipo. Los campos de la clase son:

- StringProperty **text**: propiedad con el texto del enunciado del problema.
- ArrayList<Answer> **answers**: lista con las cuatro respuestas, objetos del tipo answer.

Answer

La clase Answer se utiliza para guardar una respuesta de un problema de navegación. En la base de datos se incluye un conjunto de problemas con sus respuestas por lo que no es necesario que creéis objetos de este tipo. Los campos de la clase son:

- StringProperty **text**: propiedad con el texto de la respuesta.
- BooleanProperty **validity**: propiedad con el valor de certeza de la respuesta (es la correcta o no).

5.3. Uso de la librería desde el proyecto

Para acceder a los métodos de la librería, es necesario instanciar primero un objeto de la clase *Navegacion*, utilizando para ello el método estático *getSingletonNavegacion*(). Después, puede obtenerse la información registrada o modificarla a través del API proporcionada. Por ejemplo, en el siguiente código se añade un nuevo usuario:

```
String nickName = "nickName";
String email = "email@domain.es";
String password = "miPassword";
LocalDate birthdate = LocalDate.now().minusYears(18);
Navegacion navegacion = Navegacion.getSingletonNavegacion();
User result = navegacion.registerUser(nickName, email, password, birthdate);
```

6. Ayudas a la programación

6.1. Imagen como base de contenedor y zoom

Para realizar el trabajo se ha dejado un proyecto de Netbeans, **Poi_UPV**. Este proyecto muestra cómo se puede implementar la funcionalidad de zoom descrita en el escenario de tarea, y se puede utilizar como proyecto base para desarrollar el trabajo (no es obligatorio realizarlo a partir de este proyecto).

Para implementar el zoom se utiliza un ImageView que ocupa todo un contenedor. En el ImageView se muestra un mapa (similar a la carta náutica que se pide en el trabajo). El contenedor adecuado de javaFX para poder aplicar el zoom es el ScrollPane. En este proyecto se muestra cómo podemos incluir una imagen de un mapa en este contenedor y después aplicar un escalado. Cuando el tamaño de la imagen desborda el tamaño del contenedor debido a aplicar una función de escalado, aparecen de forma automática los scrolls que permiten movernos por todo el contenedor. Si hemos añadido objetos por detrás del ImageViewew, se mostrará por encima de este; y si aplicamos un escalado al contenedor, también se le aplicará a estos objetos. El escalado no afectará a la posición del nodo dentro del mapa, pues el escalado afecta a los dos nodos por igual, es decir mantendrá su posición (x,y) respecto al (0,0) de la imagen.

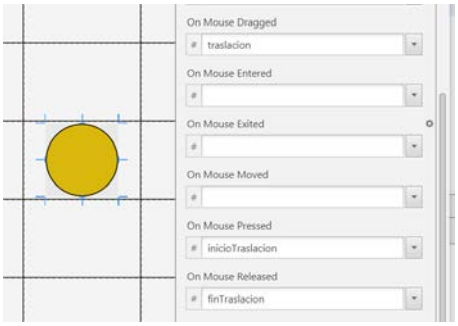
Para que esta estrategia funcione de manera adecuada es necesario utilizar objetos del tipo Group que no están disponibles en el SceneBuilder, el código necesario se incluye en el método initialize() de la clase controladora y esta explicado mediante comentarios.

Se recomienda utilizar este proyecto como base del trabajo, eliminado o cambiando aquellos elementos que no resulten adecuados para la funcionalidad requerida.

6.2. Desplazar un nodo dentro de un contenedor

Durante las prácticas ya hemos visto como desplazar un Circle sobre un GridPane mediante el ratón. Para ello hemos utilizado tres manejadores. El primero se registra

sobre el evento del tipo MousePressed y el segundo sobre el evento del tipo MouseDragged. Si se requiere hacer alguna acción especial al terminar de desplazar, se añade otro manejador sobre el evento MouseRealeased



Para pintar el nodo en otra posición fuera de la calculada se pueden utilizar los métodos setTranslateX(dx) para repintar el nodo desplazado en el eje X una distancia dx, y el método setTranslateY(dy) para hacerlo sobre el eje y. El efecto es similar a desplazar el nodo dx en el eje de las X y dy e el eje de la Y

Para desplazar un nodo dentro de un contenedor podemos seguir la misma estrategia.

1- MousePressed

Iniciamos la acción y guardamos la posición del ratón para poder calcular después en el drag el desplazamiento en X y en Y. En este caso en particular, vamos a dejar el nodo en una posición diferente a la original y por ende, cuando intentemos volver a desplazar el nodo debemos de tener en cuenta esta traslación ya aplicada sobre la posición por defecto. Así pues, es necesario al inicio del desplazamiento guardar el desplazamiento ya aplicado:

```
inicioXTrans = event.getSceneX();
inicioYTrans = event.getSceneY();
baseX = transportador.getTranslateX();
baseY = transportador.getTranslateY();
event.consume();
```

2- MouseDragged

Repintamos el nodo aplicando el desplazamiento del ratón:

```
double despX = event.getSceneX() - inicioXTrans;
double despY = event.getSceneY() - inicioYTrans;
transportador.setTranslateX(baseX + despX);
transportador.setTranslateY(baseY + despY);
event.consume();
```

3- MouseReleased

Si es necesario realizamos acciones como puede ser volver al cursor por defecto

El resultado final puede ser similar al mostrado en el siguiente video: arrastrar

6.3. Pintado de una línea

La estrategia más sencilla para pintar una línea es muy similar a la de mover el objeto con el drag de ratón. Para pintar una línea necesitamos un origen y un destino, el origen puede ser el punto en el que realizamos un presionamos del botón del ratón y el final puede ser el punto en el que soltamos el botón del ratón. Si queremos pintar la línea intermedia entre estos dos puntos, podemos tomar como final la posición del ratón mientras hacemos el arrastre. Es decir, mientras arrastramos el ratón con el botón pulsado, vamos modificando el punto final de la línea.

Para ello vamos a necesitar los dos manejadores de eventos. Con el MousePressed creamos la línea y con el MouseDragged modificamos su punto final. En el siguiente código suponemos que tenemos dos variables, *Line linePainting*, y Group *zoomGroup*

1- MousePressed

Creamos la línea:

```
linePainting = new Line(event.getX(), event.getY(),event.getX(), event.getY());
```

Añadimos la línea al contenedor:

```
zoomGroup.getChildren().add(linePainting);
```

Podemos añadir de eventos sobre la línea. Mediante el siguiente código añadimos un manejador de eventos sobre el evento ContextMenuRequested que se produce cuando se pulsa el botón derecho del ratón sobre la línea. En el manejador se crea un ContextMenu con una opción que permite borrar la línea. El menú tiene un MenuItem, eliminar. A este MenuItem se le registra un manejador para que elimine la línea al ser seleccionado. Para mostrar el menú contextual se invoca el método show(). Fíjate que dentro de la primera función lambda que tiene un parámetro "e" hay otra función lambda similar que en este caso necesitamos definir otro parámetro que llamamos "ev". Tanto "e", como "ev" son del tipo event

```
linePainting.setOnContextMenuRequested(e -> {
    ContextMenu menuContext = new ContextMenu();
    MenuItem borrarItem = new MenuItem("eliminar");
    menuContext.getItems().add(borrarItem);
    borrarItem.setOnAction(ev -> {
        zoomGroup.getChildren().remove((Node)e.getSource());
        ev.consume();
    });
    menuContext.show(linePainting, e.getSceneX(), e.getSceneY());
    e.consume();
});
```

2- MouseDragged

El paso final es muy sencillo, cuando arrastramos con el ratón lo único que tenemos que hacer es modificar el punto final de la línea con la posición actual del ratón.

```
linePainting.setEndX(event.getX());
linePainting.setEndY(event.getY());
event.consume();
```

Al soltar el botón del ratón la línea queda fija y ya no se puede modificar, aunque si la podremos eliminar si pulsamos con el botón derecho y escogemos la opción eliminar del menú contextual.

El resultado puede ser similar al mostrado en el siguiente video: trazar línea

6.4. Pintado de un arco/circulo

En este caso la estrategia más sencilla es pintar una circunferencia con la misma estrategia que hemos utilizado para pintar la línea. Con el MousePressed creamos la circunferencia en la posición del ratón y con el MouseDragged modificamos su radio. Cuando soltamos el botón del ratón la circunferencia queda fija y si hemos añadido el menú contextual de antes la podremos borrar.

Se necesitan dos pasos. En primer lugar, al pulsar el ratón, su posición será el centro del círculo. Tendremos que añadir las siguientes líneas de código al correspondiente manejador de eventos:

1- MousePressed

Crear el nodo del tipo Circle que tiene que ser transparente:

```
circlePainting = new Circle(1);
circlePainting.setStroke(Color.RED);
circlePainting.setFill(Color.TRANSPARENT);
```

Añadimos el circulo al contenedor:

```
zoomGroup.getChildren().add(circlePainting);
```

Posicionamos el circulo donde está el ratón y nos guardamos la posición X del ratón en la variable inicioXArc para calcular después el radio con el drag del ratón:

```
circlePainting.setCenterX(event.getX());
circlePainting.setCenterY(event.getY());
inicioXArc = event.getX();
```

Si queremos añadir comportamiento al círculo como el borrado o modificación del radio los podemos hacer como en el caso de la línea, añadiendo manejadores de eventos sobre el nodo

2- MouseDragged

Cuando arrastramos con el ratón modificamos el radio del círculo utilizando la posición del ratón. Este código hay que añadirlo al manejador del arrastre del ratón:

```
double radio = Math.abs(event.getX()- inicioXArc);
circlePainting.setRadius(radio);
event.consume();
```

El resultado podría ser similar al mostrado en el siguiente video: arco

6.5. Añadir texto

Una estrategia para añadir texto puede ser la de añadir un TextField en el punto seleccionado con el ratón. El usuario introducirá el texto sobre el TextField y al perder este nodo el foco o al cuando el usuario pulse enter tendríamos que eliminar el textfield y crear en un lugar un nodo del tipo Text con el texto que el usuario ha introducido en el nodo anterior. Si queremos que el usuario pueda eliminar este nodo podemos añadir el manejador con el menú contextual que ya hemos visto antes.

Los pasos por seguir son pues, primero pulsamos con el ratón y sobre su posición añadimos un TextField:

1- MousePressed

```
TextField texto = new TextField();
```

Añadimos el texto al contenedor, lo posicionamos donde está el ratón y muy importante, pedimos el foco.

```
zoomGroup.getChildren().add(texto);
texto.setLayoutX(event.getX());
texto.setLayoutY(event.getY());
texto.requestFocus();
```

Añadimos el comportamiento al TextField

```
texto.setOnAction(e -> {
    Text textoT= new Text( texto.getText());
    textoT.setX(texto.getLayoutX());
    textoT.setY(texto.getLayoutY());
    textoT.setStyle("-fx-font-family: Gafata; -fx-font-size: 40;");
    zoomGroup.getChildren().add(textoT);
    zoomGroup.getChildren().remove(texto);
    e.consume();
});
```

Cuando terminamos con el textfield se crea un nodo del tipo Text que sustituirá al TextField. Si queremos añadir comportamiento al Text como por ejemplo que se muestre un menú contextual, este es el lugar adecuado

El resultado podría ser similar al mostrado en el siguiente video: texto

6.6. Carga de imágenes desde disco duro

Existen diversas formas de cargar una imagen desde el disco duro y mostrarla en una *ImageView*:

1. La imagen está en un subdirectorio del directorio src del proyecto, por ejemplo, llamado images:

```
String url = File.separator+"images"+File.separator+"woman.PNG";
Image avatar = new Image(new FileInputStream(url));
myImageView.imageProperty().setValue(avatar);
```

2. La imagen está en cualquier parte del disco duro y tenemos el *path* completo de la misma:

```
String url = "c:"+File.separator+"images"+File.separator+"woman.PNG";
Image avatar = new Image(new FileInputStream(url));
myImageView.imageProperty().setValue(avatar);
```

6.7. Manejo de fechas y del tiempo

En la aplicación se deben gestionar atributos de tipo *LocalDateTime, LocalDate* y de tipo *DateTime*. A continuación, os explicamos algunos métodos de utilidad.

Creación de una fecha o un campo de tiempo

Consulta el API de *LocalDate* y *LocalTime* para conocer todos los métodos que proporciona para crear un objeto de sus clases, pero algunos que te pueden resultar útiles son:

```
LocalDate date = LocalDate.now();
LocalDateTime datetime = LocalDateTime.now();

LocalDate sanJose = LocalDate.of(2019, 3,19);
LocalTime mascleta = LocalTime.of(14,0);
LocalDateTime sanJoseMascleta = LocalDateTime.of(2019,3,19,14,0);
LocalDate sanJose2 = sanJoseMascleta.toLocalDate();
LocalTime mascleta2 = sanJoseMascleta.toLocalTime();

LocalDateTime nextWeekDay= LocalDateTime.now().plusDays(7);
LocalDateTime nextMontDay = LocalDateTime.now().plusMonths(1);
LocalTime endedTime = LocalTime.now().plusMinutes(90);
```

6.8. Configurar DatePicker

Los componentes DatePicker permiten seleccionar al usuario una fecha, pudiendo acceder al valor seleccionado a través de su propiedad valueProperty(). Es posible configurar la forma en la que se visualiza por defecto cada día del calendario, siendo posible deshabilitar algunos días, cambiar el color de fondo, etc. La forma en la que se configura esta visualización es similar a la que empleamos para configurar una ListView o una TableView. La diferencia es que en este caso debemos extender la clase DateCell, y usar el método setDayCellFactory. Por ejemplo, el siguiente código configura un DatePicker para que se inhabiliten los días anteriores al 1 de Marzo de 2020 (ver Figura 5).

```
dpBookingDay.setDayCellFactory((DatePicker picker) -> {
            return new DateCell() {
                @Override
                public void updateItem(LocalDate date, boolean empty) {
                    super.updateItem(date, empty);
                    LocalDate today = LocalDate.now();
                    setDisable(empty || date.compareTo(today) < 0 );
                }
            };
        });
```

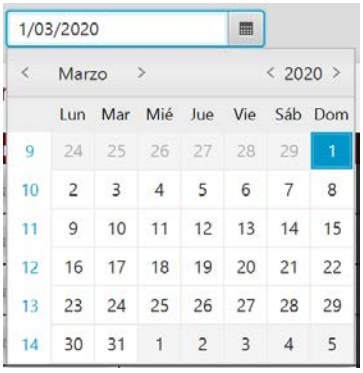


Figura 1. DataPicker configurado para inhabilitar días en el pasado

7. Instrucciones de Entrega

Todos los grupos tendrán la misma fecha entrega. Exportad el proyecto Netbeans a un fichero zip (opción `Fichero>Exportar Proyecto> A Zip`). O simplemente guardad vuestra carpeta donde está el proyecto en un fichero zip.

- Un único miembro del grupo sube el fichero zip a la tarea correspondiente, incluyendo en el campo de comentarios los nombres de los miembros del grupo.

8. Evaluación

- Aquellos proyectos que no compilen o que no se muestren la pantalla principal al arrancar se calificarán con un cero.
- Se deberán incluir los diálogos de confirmación, errores, etc. que se considere necesario.
- Hay que utilizar hojas de estilo para dar formato a la aplicación.
- Para evaluar el diseño de la interfaz de la aplicación se tendrán en consideración **los principios, guías de diseño y demás directrices estudiadas en clase de teoría**.
- Debe ser posible redimensionar la pantalla principal, los controles se ajustarán adecuadamente para usar el espacio disponible (usa los contenedores vistos en clase: *VBox*, *HBox*, *BorderPane*, *GridPane*, etc.).
- Se aplicará la Normativa de Integridad Académica de la UPV y la Normativa de Honestidad Académica de la ETSInf. En la asignatura existen herramientas anti plagio.