CASO PRÁCTICAS

Navegación digital - Librería IPC2025

Interfaces Persona Computador

IPC – DSIC

UPV

Curso 2024-2025

Índice

Tablas

Figuras

1. Librería proporcionada para la persistencia de los datos

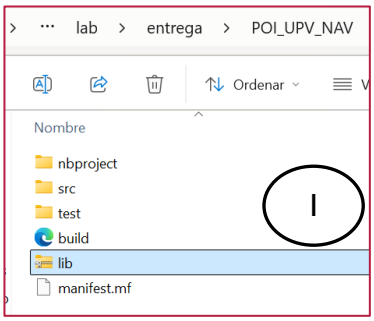
La persistencia de la información se va a realizar a través de una base de datos *SQLite*[1]. En la librería de acceso (*IPC2025.jar*) se define el modelo de datos a utilizar en la práctica. Esta librería os permitirá almacenar y recuperar toda la información que necesite persistencia. En concreto la aplicación mantendrá un conjunto de problemas de navegación con sus respuestas. También guardará la información de los usuarios, sus datos de registro y de cada sesión iniciada se guardará el número de problemas resueltos correctamente y el número de problemas no resueltos adecuadamente, así como la fecha de registro de la sesión.

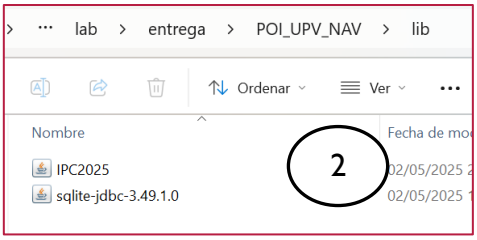
La librería gestiona todo el acceso a la base de datos, creándola de forma automática si es necesario dentro del directorio de vuestro proyecto, en un fichero denominado "data.db". Si el sistema crea la base de datos esta estará vacía por lo que no se dispondrán de problemas que mostrar al usuario. Os 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.

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].

1.1. Agregar la librería IPC2025.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: *IPC2025.jar y sqlite-jdbc-3.49.1.0.jar.* (2).

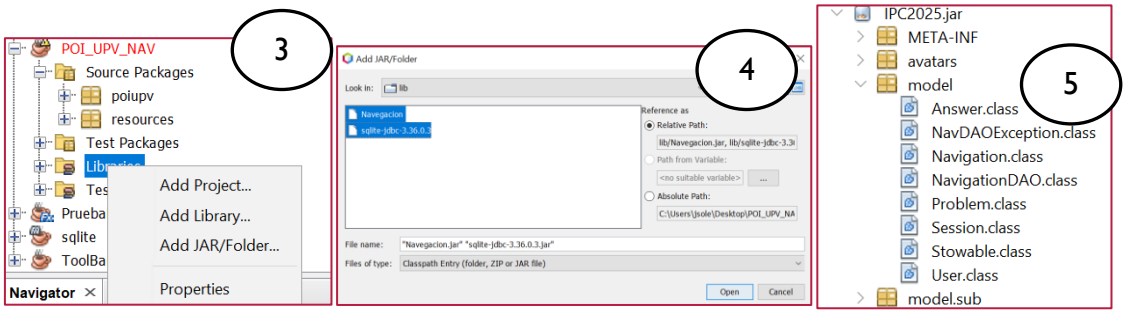




A continuación, añadiremos estas librerías a vuestro proyecto. Los proyectos creados por Netbeans para aplicaciones java tienen una carpeta `Libraries` donde añadir las librerías externas que se desee. Para ello, basta situarse sobre esa carpeta, y desde el

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

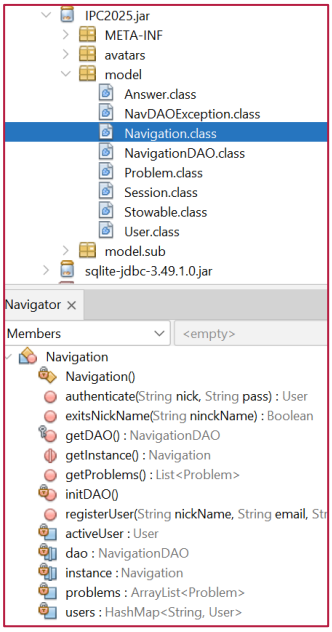
menú contextual (botón derecho del ratón), seleccionar la opción `Add JAR/Folder`, tal y como se muestra en (3).



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).

1.2. API proporcionado por *IPC2025.jar*

La librería proporciona diferentes clases, de las que solo necesitaréis usar aquellas que están dentro de la carpeta *model* (6). 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 (6). A continuación, describiremos los aspectos más importantes de las clases proporcionadas.



6

1.2.1. Clases del modelo

Navigation

La clase *Navigation* 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 **getInstance()**.

getInstance()

```
public static Navigation getInstance ()
```

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.

authenticate()

```
public User authenticate(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. El constructor de la clase User no es público por lo que esta es la única manera de crear objetos User nuevos.

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.

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, el campo nickName es el único que no se puede modificar una vez creado el usuario

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 1.

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(int hits, int faults)
```

Los atributos de la clase Session son:

- LocalDateTime **timeStamp**: día y hora en el que se 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:

- String **text**: 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:

- String **text**: el texto de la respuesta.
- Boolean **validity**: valor de veracidad de la respuesta (true es la correcta, false es incorrecta).

1.3. 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 *getInstance*(). 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);
Navigation navigation = Navigation.getInstance();
User result = navigation.registerUser(nickName, email, password,null , birthdate);
```

1.4. Base de datos de prueba

En la carpeta de recursos dejamos un fichero data.db que contiene varios problemas y un usuario con las credenciales: nickName= ***user1*** password= ***User123!***

Para acceder a estos datos debes de dejar el fichero en la carpeta donde está el proyecto de Netbeans.

2. Ayudas a la programación

2.1. 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, pero si lo utilizas, cambia su nombre).

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.

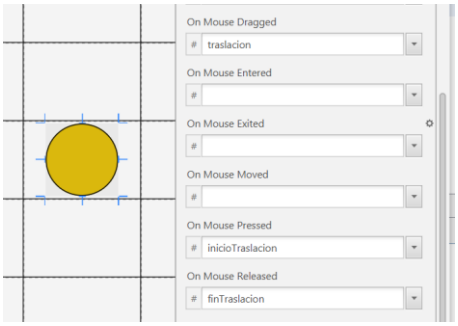
Para que esta estrategia funcione de manera adecuada es necesario utilizar objetos del tipo Group que no están disponibles en el SceneBuilder, el código necesario se incluye en el método initialize() de la clase controladora y esta explicado mediante comentarios.

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

Todos los objetos que se añadan al Group o ScrollPane se verán afectados por el escalado, si quieres añadir objetos sobre la imagen pero que no se vean afectados por el escalado puedes añadir sobre el Group un contenedor del tipo StackPane y añadir los objetos sobre este contenedor.

2.2. 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

Si el nodo a desplazar está en un contenedor afectado por un escalado, es decir, el ScrollPane sobre el que aplicamos el zoom en la aplicación Poi_UPV, debemos de tener en cuenta que el método *getSceneX()* y *getSceneY()* nos devuelve las coordenadas de la posición del ratón sin tener en cuenta este escalado, y por ello tendremos que aplicar una conversión para que el *setTranslateX()* y *setTranslateY()* funcione adecuadamente. Para resolver este problema y trasladar adecuadamente el nodo se pude utilizar la función ***sceneToLocal()*** que se encargar de la transformación entre los dos sistemas de coordenadas. Así pues, los dos pasos anteriores quedarían de la siguiente manera:

1- **MousePressed**

Iniciamos la acción y guardamos la posición del ratón en el sistema de coordenadas adecuado. La variable se define: Point2D localBase:

```
localBase = zoomGroup.sceneToLocal(ev.getSceneX(), ev.getSceneY());
baseX=transportador.getTranslateX();
baseY=transportador.getTranslateY();
ev.consume();
```

2- MouseDragged

Repintamos el nodo aplicando el desplazamiento del ratón en el espacio de coordenadas adecuado:

```
Point2D localPos = zoomGroup.sceneToLocal(ev.getSceneX(), ev.getSceneY());
transportador.setTranslateX(baseX+ localPos.getX()-localBase.getX());
transportador.setTranslateY(baseY+ localPos.getY()-localBase.getY());
ev.consume();
```

Recuerda que utilizar esta transformación solo es necesario si estamos aplicando un escalado al contenedor sobre el que desplazamos en este caso el objeto transportador.

2.3. 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

2.4. 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

2.5. 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

2.6. 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);
```

2.7. 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);
```

2.8. 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 1).

```
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);
                }
            };
        });
```

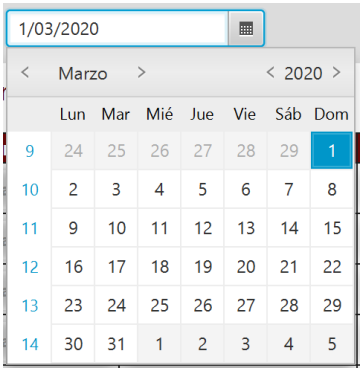


Figura 1. DataPicker configurado para inhabilitar días en el pasado

2.9. Carga de una imagen vectorial desde una hoja de estilos

Existe la posibilidad de mostrar una imagen vectorial, por ejemplo un transportador, sin utilizar un ImageView y haciendo uso de cualquier nodo que herede de la clase Region al que se le puede aplicar un estilo de CSS. En nuestro caso puede ser útil pues este tipo de imágenes son perfectamente transparentes si así lo consideramos y las podemos utilizar tanto para mostrar un transportador de ángulos, una regla o un compas.

Si consideramos el siguiente estilo:

```
.transportador{
    -fx-shape:"M 1900.0049,17090.005 V 9509.9951 1930 H 9480 17059.995 V 950
    -fx-pref-width: 350;
    -fx-pref-height: 350;
    -fx-scale-y: -1;                        /* invierte verticalmente */
    -fx-background-color: blue;
}
```

Podemos ver que la propiedad -fx-shape: tiene asignado un String que coincide con el path vectorial de la imagen que queremos mostrar (cuidado que en la imagen solo se ha capturado un trozo de este path). Ademas se asigna un tamaño al objeto y un color.

Por último, con la propiedad -fx-scale-y: -1 se consigue invertir la imagen para que el punto (0.0) de la imagen en JavaFX coincida con el (0,0) en el formato vectorial o svg.

Ahora solo nos falta crear un nodo, por ejemplo, del tipo Button, asignarle el estilo "transportador" y añadirlo al contenedor correspondiente para que se muestre:

```
Button transportador = new Button();
transportador.getStyleClass().add("transportador");
zoomGroup.getChildren().add(transportador);
```

El resultado es el siguiente: