

MANUAL TÉCNICO

TPV

Desarrollo de Aplicaciones Multiplataforma

Realizado por : Ricky Nastase Nitulescu

Fecha: 10/03/2024

Asignatura: **Desarrollo de Interfaces**

Tutor: Justo García de Paredes Haba

ÍNDICE

1. Introducción.	2
1.1 Objetivo del trabajo.	2
2. Planificación del proyecto.	2
2.1 Recursos y tecnologías usadas.	2
2.2 Planificación.	2
3. Diseño de la aplicación.	3
3.1 Diagrama de clases.	3
3.2 Explicación de estructura BBDD.	3
4. Desarrollo del proyecto.	4
4.1 Control de versiones.	4
4.2 Creación del proyecto.	5
4.3 Organización y explicación de ficheros.	6
5. Casos de uso.	7
5.1 Diagrama de casos de uso.	7
5.2 Explicación de casos de uso.	8
5.3 Explicación del código según el uso.	9
Visualizar sala.	9
Seleccionar mesa.	9
Añadir producto a mesa.	9
Eliminar producto de mesa.	10
Establecer cantidad de productos.	10
Cobrar mesa.	10
Generar factura.	10
6. Generación de JAR y EXE.	11
Compilado	11
Generar artefacto.	11
Compilar artefacto.	12
Configurar JRE.	12
Configurar el empaquetado del EXE.	13
7. Conclusiones.	13

1.Introducción.

1.1 Objetivo del trabajo.

El objetivo de este trabajo final para la asignatura de Desarrollo de Interfaces es el desarrollo de un sistema TPV (Terminal Punto de Venta) para la gestión y facturación de negocios hosteleros. En concreto, el negocio al que irá destinado actualmente se llama “*El Rinconcito de Ricky*”, sin embargo, este sistema gestor puede ser adaptado a otros negocios y modificado según las necesidades particulares de cada uno.

Se pretende que esta aplicación sirva principalmente para:

- ❖ Automatizar la gestión de los trámites hosteleros diarios.
- ❖ Evitar el uso de papel para anotaciones, registro de datos, etc.
- ❖ Con un uso adecuado de la misma, obtener información en las facturas correctamente detallada y exacta, sin errores ni problemas de integridad en los datos y métodos seguros de pago y transacción de los datos.
- ❖ Mayor rapidez y facilidad a la hora de atender clientes, recoger datos y visualizar la información.

2.Planificación del proyecto.

2.1 Recursos y tecnologías usadas.

Tras una explicación introductoria detallada de las necesidades a implementar en el trabajo por el profesorado y recoger los requisitos que este debe tener para su completa funcionalidad, se han establecido los siguientes recursos requeridos para llevar a cabo el desarrollo:

- ❖ Entorno de desarrollo principal: IntelliJ.
- ❖ Sistema gestor de base de datos: MySQL.
- ❖ Lenguaje de programación: JavaFX.
- ❖ Generación de reportes: JasperReports a través de JasperSoft.
- ❖ Estilos: hojas de estilo CSS.

2.2 Planificación.

Para la planificación del proyecto se han establecido estas 4 fases principales:

- ❖ Definición y planificación.
- ❖ Análisis y diseño.
- ❖ Desarrollo e implementación.
- ❖ Periodo de pruebas y correcciones.
- ❖ Documentación del proyecto.

3. Diseño de la aplicación.

3.1 Diagrama de clases.

A continuación, se proporciona el diagrama de clases realizado sobre la estructura de la base de datos para el almacenamiento y organización de la información del sistema gestor. Esta estructura está focalizada en ser lo más concisa y optimizada posible, con el número mínimo de tablas (como se ha recomendado por el profesorado) pero con la información necesaria para luego generar las facturas correctamente.

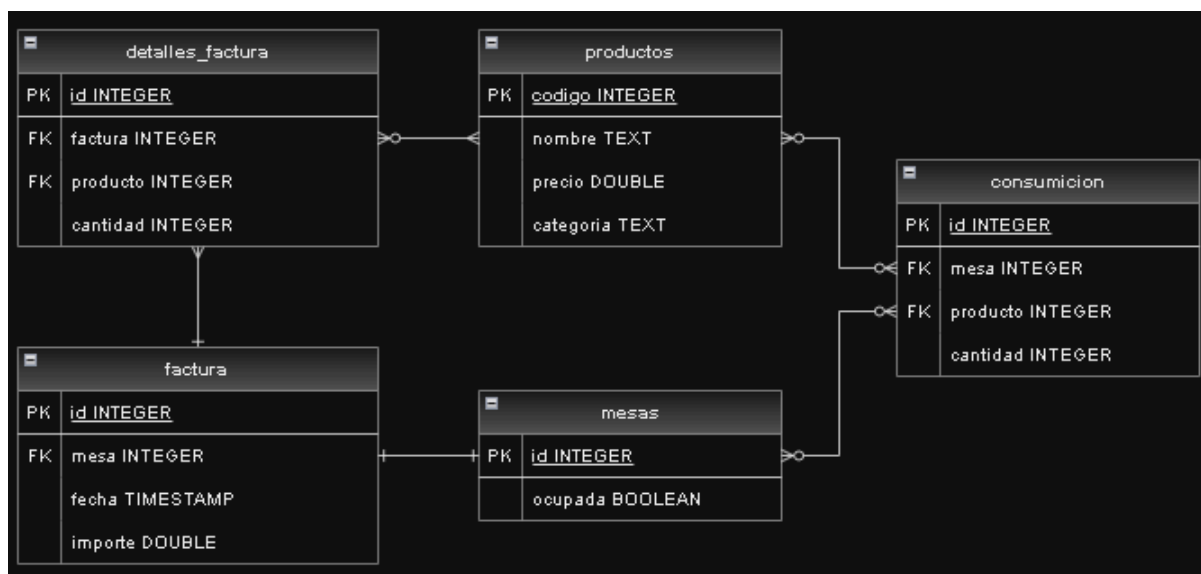


Diagrama de clases

3.2 Explicación de estructura BBDD.

En este apartado se procede a explicar la estructura de la base de datos y la relación entre las tablas fijándonos en el diagrama anterior:

- ❖ **productos**: la tabla de los productos contiene información propia de cada producto que se venderá en el establecimiento. Su relación con la tabla consumición es $M:N$, al igual que con la tabla de detalles de las facturas.
- ❖ **mesas**: esta tabla contiene el identificador propio de cada mesa del establecimiento así como una variable booleana para determinar si en el momento actual está siendo ocupada, es decir, posee productos que están siendo consumidos pendientes de pago.
- ❖ **consumicion**: esta tabla se usa como almacenamiento de información temporal, es decir, guardará las consumiciones de las mesas (los productos) en el momento actual. Una mesa puede consumir varios productos y los productos pueden ser consumidos por varias mesas.

En el momento en el que la mesa sea cobrada o sean eliminados los productos de la misma por algún motivo, esta tabla trasladará los datos a las tablas de factura y

detalle_factura y desaparecerán de la misma los datos correspondientes al id de la mesa que ha sido cobrada.

- ❖ **factura**: en esta tabla se almacenan los comprobantes que se han generado al haber cobrado las mesas. A través de los datos de la tabla consumicion, cuando se cobra la mesa se calcula el importe total (cantidad de producto * precio del producto, sobre todos los productos), y se almacena en esta tabla el id de la mesa cobrada, su importe total y la fecha y hora en la que se ha realizado esta facturación. Una factura en concreto solo puede tener una mesa y las mesas pueden generar una o varias facturas en distintos tiempos.
- ❖ **detalle_factura**: en esta tabla serán recogidos los detalles de cada factura que acabamos de explicar. Se almacena el id de la factura generada y, de la tabla consumicion, se recoge cada uno de los productos que han sido consumidos por la mesa en concreto de la factura. Esto se hace así para luego generar la factura diaria, la cual recogería todas las facturas creadas en la fecha de un día concreto y los productos consumidos por cada mesa, así como sus precios y el importe total. Los detalles de la factura solo pueden tener una factura y mesa a la vez pero varios productos asociados a estas. Las mesas pueden aparecer en varios detalles al igual que los productos, asociándose siempre al id de una factura en concreto. Y las facturas pueden tener muchos detalles de factura.

4.Desarrollo del proyecto.

4.1 Control de versiones.

Para el desarrollo seguro de la aplicación y un control de versiones que nos permitan respaldar el proyecto, asegurarnos de no perder ninguna implementación durante el proceso, gestionar los errores que puedan surgir para así poder volver y avanzar en la línea del tiempo de su creación y obtener un seguimiento exhaustivo de todo el proyecto, se ha optado por la utilización de la herramienta *GitHub*.

Para ello, se ha creado en una cuenta personal un repositorio con el nombre del proyecto y se ha añadido la carpeta donde se encuentra el mismo. Personalmente he optado por la utilización del intérprete de comandos de mi ordenador para la generación de “commits” y almacenar el código y los archivos modificados a lo largo del desarrollo en vez de la herramienta *Desktop* que proporciona *GitHub*.

La manera en la que se han ido añadiendo al historial de versiones los cambios realizados en el proyecto se ha basado en que, cada día que hubiese desarrollado algo significativo y notorio, se ha añadido al repositorio indicando en un comentario el cambio específico realizado. Sin embargo, los cambios insignificantes o prácticamente inacabados aún, se ha decidido no ser documentados en el repositorio.

4.2 Creación del proyecto.

Para la creación del proyecto con el entorno de desarrollo de *IntelliJ* se han escogido las siguientes características:

New Project

Empty Project

Generators

- Maven Archetype
- JavaFX
- Kotlin Multiplatform
- Compose for Desktop
- IDE Plugin
- Android

To create a general Maven project, go to the [New Project](#) page.

Name:

Location:

Project will be created in: E:\2DAM\DI\TPV

☐ Create Git repository

JDK:

Catalog: [Manage catalogs...](#)

Archetype:

Version:

Additional Properties

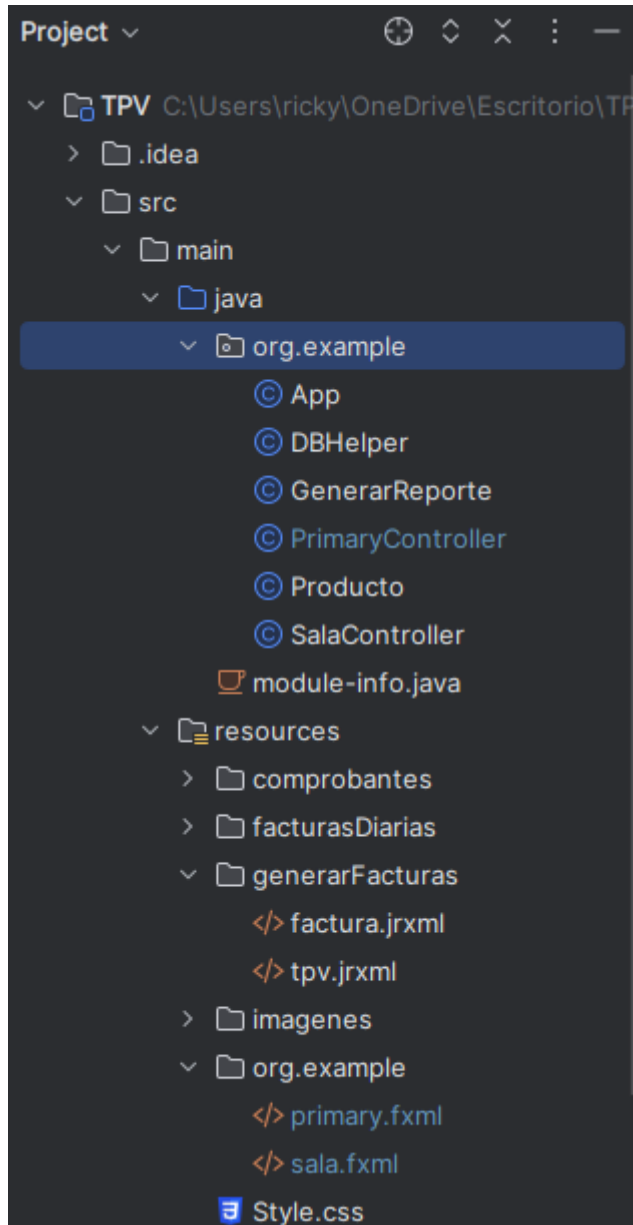
javafx-version	<input type="text" value="21"/>
javafx-maven-plugin-version	0.0.6
add-debug-configuration	N

> Advanced Settings

Creación del proyecto

4.3 Organización y explicación de ficheros.

Los ficheros del proyecto están organizados dentro del mismo de la siguiente manera:



En la carpeta **java/org.example** se encuentran las clases java principales donde se desarrolla todo el código fuente.

- **DBHelper**: clase que actuará como intermediaria para la gestión de la información de la base de datos con la interfaz del proyecto. Contendrá todos los métodos necesarios y requeridos para acceder a la misma y realizar operaciones CRUD.
- **GenerarReporte**: clase que maneja los JasperReports para generar comprobantes y facturas diarias.
- **PrimaryController**: clase principal donde se interactúa con el *primary.fxml* y lleva el manejo de las consumiciones de las mesas y la generación de reportes.
- **Producto**: clase representativa de cada producto que ofrece el establecimiento para consumición.
- **SalaController**: clase secundaria que interactúa con el *sala.fxml* y lleva el manejo de la elección de la mesa.

Carpeta **comprobantes**: directorio donde se almacenarán los comprobantes generados.

Carpeta **facturasDiarias**: directorio donde se almacenarán las facturas diarias generadas.

Carpeta **generarFacturas**: contiene los ficheros *jrxml* con los que se generarán los reportes.

Carpeta **imagenes**: contiene las imágenes que se muestran en las interfaces del programa.

En la carpeta **resources/org.example** se encuentran los ficheros *fxml* anteriormente mencionados los cuales contienen, respectivamente, la interfaz donde se gestionan los productos y la generación de reportes, y la interfaz que gestiona las mesas de la sala.

Por último, encontramos el fichero *Style.css* con el cual implementamos una mejora en el diseño de la aplicación.

5. Casos de uso.

5.1 Diagrama de casos de uso.

Se adjunta el diagrama de casos de uso creado:

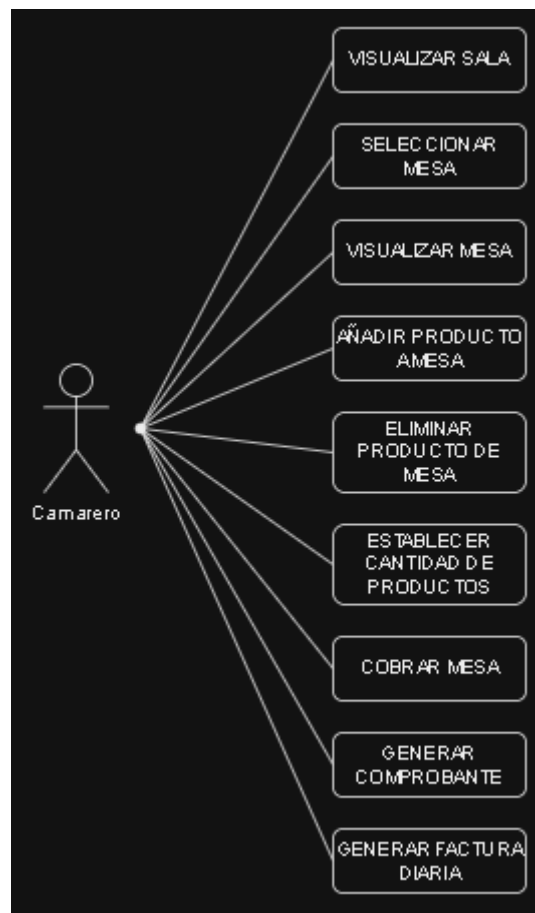


Diagrama de casos de uso

5.2 Explicación de casos de uso.

Descripción del caso de uso	Resultado esperado
Visualizar sala.	Se abre la interfaz de visualización de las mesas del bar, donde se muestra cada mesa y taburete con su número de identificación correspondiente y en caso de estar ocupados, el asiento o mesa se mostrarían de color rojo.
Seleccionar mesa.	Se vuelve a la interfaz principal de gestión general donde ahora se estarían realizando los cambios sobre la mesa anteriormente seleccionada.
Visualizar mesa.	Al seleccionar la mesa y cambiar a la interfaz general, se podría observar en la tabla los productos que la mesa en concreto estaría consumiendo. En caso de estar vacía, la mesa no estaría consumiendo productos y por ende estaría libre también.
Añadir producto a mesa.	Si no se ha seleccionado con anterioridad una mesa, al pinchar en cualquier producto no se ejecutaría ninguna función. En caso de haber seleccionado ya una mesa, al pinchar sobre un producto se añadiría a la tabla como consumición de la mesa y se establecería la misma como ocupada.
Eliminar producto de mesa.	Si seleccionamos un producto de la tabla y pinchamos sobre el botón "X" de la derecha, ese producto se eliminaría de la consumición de la mesa. En caso de pinchar el botón no teniendo un producto seleccionado, no se ejecutaría ninguna función.
Establecer cantidad de productos.	Si seleccionamos un producto, pulsamos los botones indicando la cantidad de productos que se han consumido en la mesa y pinchamos el botón "✓", se modificaría la cantidad consumida de ese producto en la mesa seleccionada.
Cobrar mesa.	Al pinchar en la imagen de "COBRAR", se limpiaría la tabla de la interfaz pues se eliminan las consumiciones de la mesa y se genera un comprobante, el cual se visualizará en la pantalla inmediatamente, indicando, entre otros datos, la mesa, los productos y el importe total del cobro.
Generar factura diaria.	Al pinchar la imagen de "FACTURAR" se generaría la factura diaria recogiendo todos los cobros de la fecha actual en la que se quiere generar y se podría visualizar en la pantalla de forma inmediata.

5.3 Explicación del código según el uso.

Visualizar sala.

Para establecer las mesas y taburetes como ocupados y no ocupados se ha creado el siguiente método:

```
1 usage  RickyNastase
private void setOcupada() {
    for (Button m : mesas) {
        String id = m.getId().split(regex: "a*")[1];
        if (db.getOcupada(Integer.parseInt(id))) {
            m.setStyle("-fx-background-color: #FF0000;");
        }
    }
}
```

Este método es llamado siempre que se accede a la interfaz de visualización de la sala y asegura, pasándole a la base de datos la lista de las mesas, que aquella que esté ocupada se muestre en rojo.

Seleccionar mesa.

```
@FXML
private void mesaSeleccionada(ActionEvent event) {
    Button mesa = (Button) event.getSource();
    App.idMesa = Integer.parseInt(mesa.getText());
    try {
        App.setRoot("primary");
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Este método es establecido en el *On Action* de cada botón de mesas en el cual se recoge el nombre del botón (el id de la misma), se pasa a la variable del programa principal y se cambia la interfaz al mismo, así se podrá gestionar sobre la seleccionada.

Añadir producto a mesa.

```
img.addEventHandler(MouseEvent.MOUSE_CLICKED, new EventHandler<MouseEvent>() {
    RickyNastase
    @Override
    public void handle(MouseEvent event) {
        String[] nombre = imagen.getName().split(regex: ".png");
        String p = nombre[0];
        addProducto(p);
    }
});

private void addProducto(String nombreProducto) {
    if (idMesa > 0) {
        db.addProductoMesa(nombreProducto, idMesa);
        cargarTabla();
    }
}
```

En cada imagen del grid que representa cada producto del establecimiento hay una función en la que se establece que al pinchar sobre esta se llame a la función que añadirá en la consumición de la mesa seleccionada el producto y se volverá a cargar la tabla con las consumiciones de la mesa.

Eliminar producto de mesa.

```
@FXML
private void eliminarProducto() {
    Producto p = tablaProductos.getSelectionModel().getSelectedItem();
    if (p != null) {
        db.eliminarConsumicion(idMesa, p.getCodigo());
        cargarTabla();
    }
}
```

Este método es establecido en el *On Action* del botón "X". Si hay un producto seleccionado, se elimina de la tabla de consumo de la mesa seleccionada.

Establecer cantidad de productos.

```
@FXML
private void aceptar() {
    if (!cantidad.getText().isEmpty()) {
        int cant = Integer.parseInt(cantidad.getText());
        Producto p = tablaProductos.getSelectionModel().getSelectedItem();
        if (p != null) {
            db.anadirCantidad(cant, idMesa, p.getCodigo());
            cantidad.setText("");
            cargarTabla();
        }
    }
}

@FXML
private void setCantidad(ActionEvent event) {
    Button boton = (Button) event.getSource();
    if (cantidad.getText().isEmpty() & boton.getText().equals("0")) {
    } else {
        cantidad.setText(cantidad.getText() + boton.getText());
    }
}
```

Este método es establecido en el *On Action* del botón "✓". Recoge el texto donde se añade la cantidad y en caso de no estar vacío y de que esté un producto seleccionado, se cambia la cantidad del producto de la mesa a la recogida nueva.

El método de abajo se setea a cada botón numérico para añadir la cantidad al texto de la interfaz. Se comprueba que la cantidad no empiece por cero.

Cobrar mesa.

```
cobrar.addEventHandler(MouseEvent.MOUSE_CLICKED, new EventHandler<MouseEvent>() {
    @Override
    public void handle(MouseEvent event) {
        if (tablaProductos.getItems().size() > 0) {
            tablaProductos.getItems().clear();
            db.generarFactura(idMesa);
            db.generarComprobante(idMesa);
            cargarTabla();
        }
    }
});
```

Método asociado a la imagen de cobrar. En caso de no estar vacía la tabla de productos, se llama al método que generará una factura y sus detalles en la base de datos, y también al método el cual generará el reporte, un comprobante con esos productos de la mesa y demás datos que se muestran inmediatamente en la pantalla y se guarda como pdf en la carpeta *comprobantes*.

Generar factura.

```
facturar.addEventHandler(MouseEvent.MOUSE_CLICKED, new EventHandler<MouseEvent>() {
    @Override
    public void handle(MouseEvent event) {
        db.generarFacturaDiaria(fechaActual);
        cargarTabla();
    }
});
```

Método asociado a la imagen de facturar. Llama al método que recoge los datos de las facturas y de sus detalles correspondientes para generar un reporte y almacenarlo en el directorio de *facturasDiarias*.

6. Generación de JAR y EXE.

A continuación se procede a explicar la generación del .jar para la aplicación desarrollada. En este caso no se ha conseguido hacer realmente por incompatibilidad de requerimientos. Es por ello que se hará una simulación habiendo guardado una copia de seguridad del proyecto para luego poderlo recuperar pues con las implementaciones que se necesitan para esta parte, el proyecto queda inutilizable.

Compilado

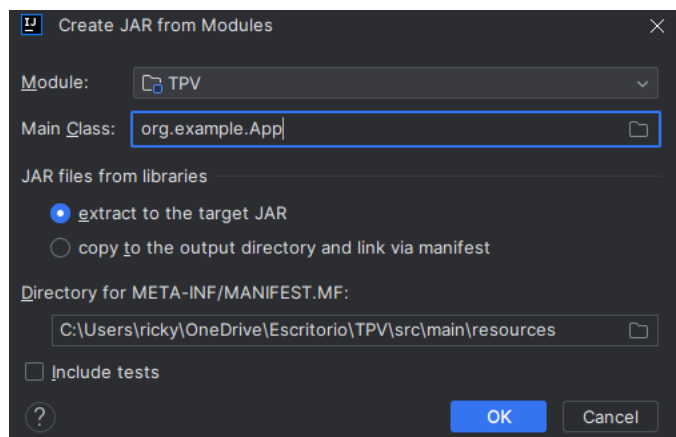
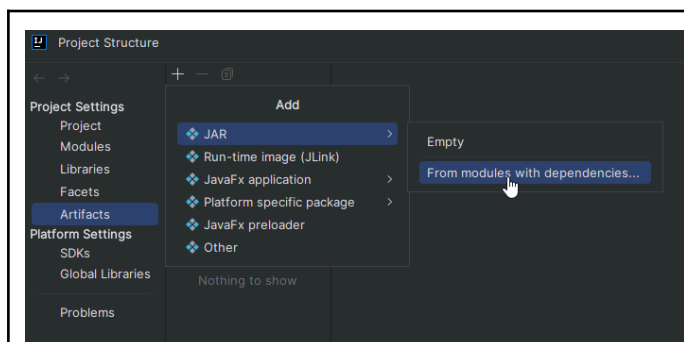
Se cambian las versiones de compilado a 1.8.

```
<properties>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<maven.compiler.source>1.8</maven.compiler.source>
<maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

```
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>3.8.0</version>
<configuration>
<release>1.8</release>
</configuration>
</plugin>
```

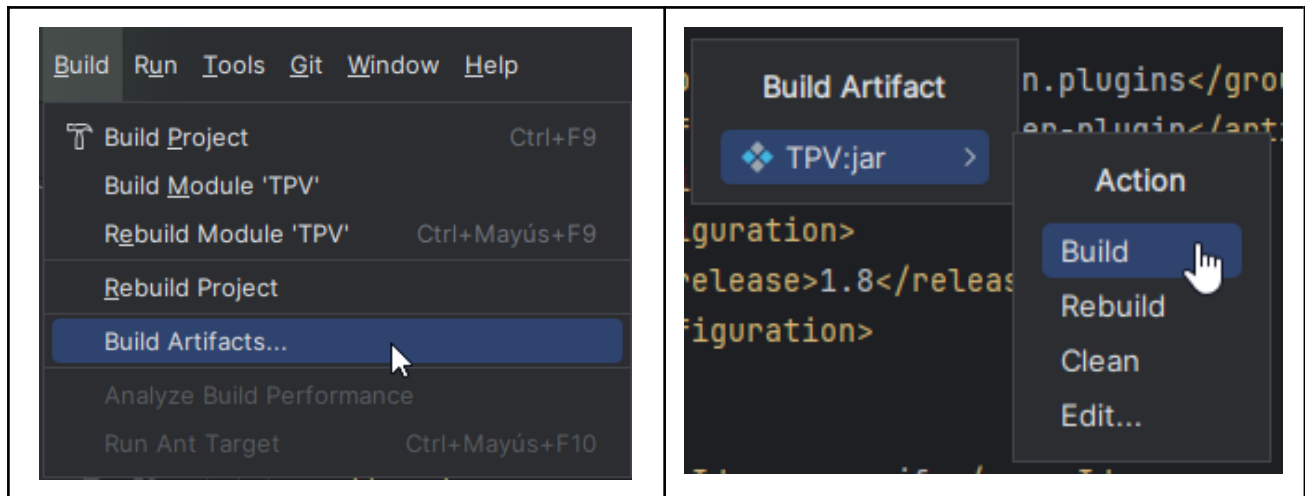
Generar artefacto.

Se genera un artefacto tras haber actualizado los cambios y borrado el archivo *"module-info.java"*.



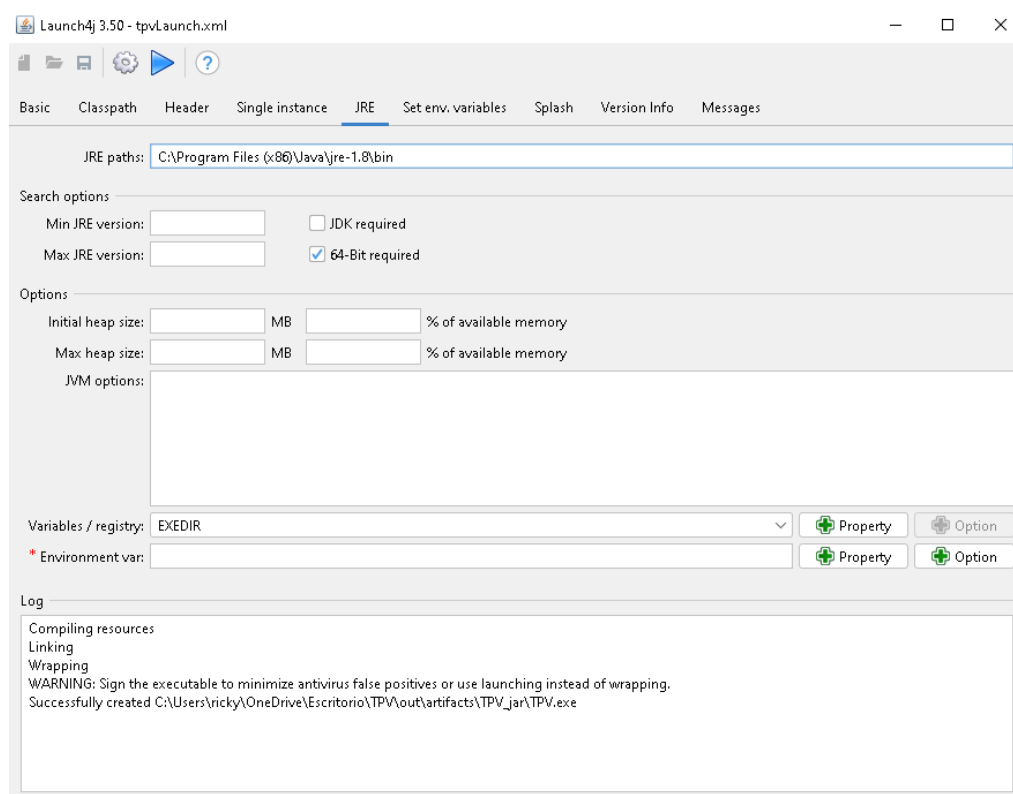
Compilar artefacto.

Se compila el artefacto creado.



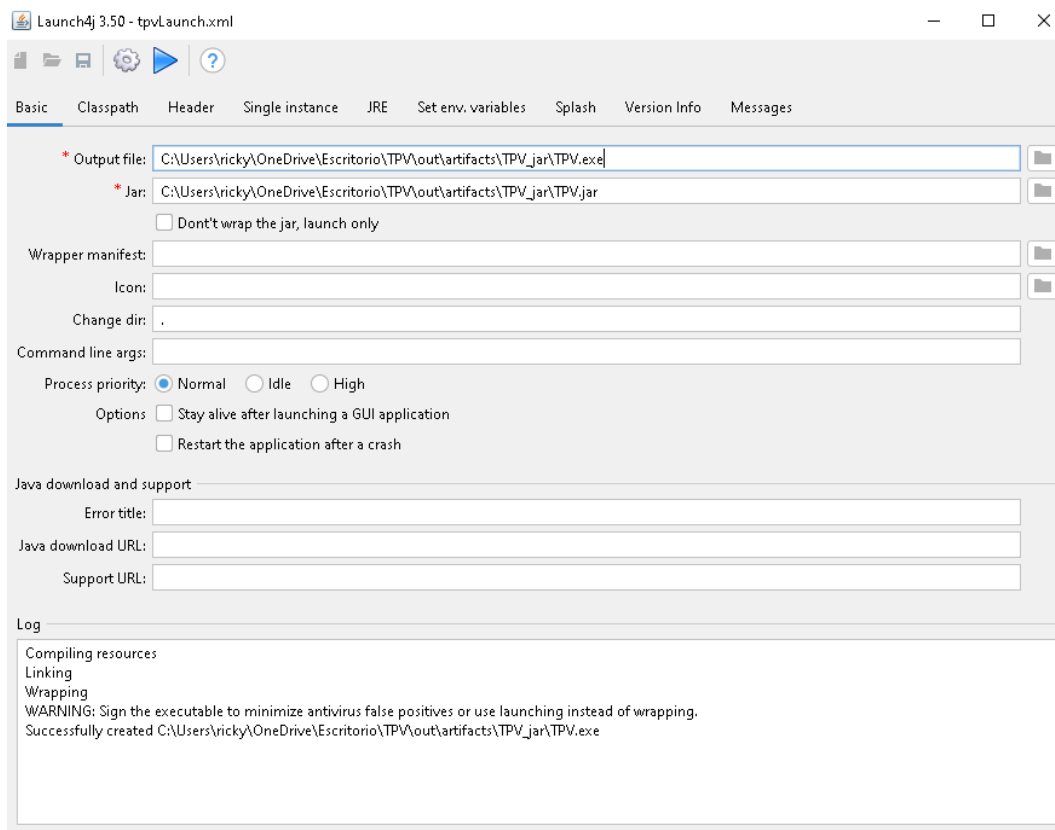
Configurar JRE.

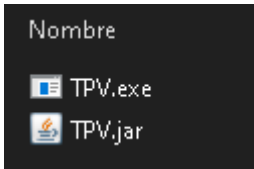
Con la aplicación *Launch4J* ya instalada, se procede a configurar el JRE que se va a usar.



Configurar el empaquetado del EXE.

El último paso para generar el .exe es establecer las rutas y darle a ejecutar.



	<p>De esta manera, ya tendríamos tanto el jar como el ejecutable de la aplicación que hemos creado.</p>
---	---

7. Conclusiones.

Finalizando este proyecto, como conclusión cabe decir que este trabajo ha sido uno entretenido a la par que desafiante en ciertos momentos. Se han implementado conocimientos que ya se tenían en la gran mayoría pero también se ha necesitado aprender ciertos métodos de integridad de datos, funcionalidades dentro del código, generación de documentos pdf, etc.

Este proyecto también ha requerido una fase de adquisición de información necesaria para el desarrollo de la lógica del código siendo el tema principal y único en el que se basa un negocio hostelero. Para cualquier proyecto que esté desarrollado para un sector en concreto o una temática específica, se necesita tener los conocimientos para poder implementar las funcionalidades específicas que se requieren. Es por ello que

personalmente ya sabía ciertas cosas sobre el funcionamiento de un TPV de hostelería y he podido ayudar a unos compañeros con ello, pero también he aprendido a razonar todo lo que el programa realiza por detrás y posibles mejoras futuras del mío para poderse parecer a los TPVs de uso comercial.

En definitiva, ha sido ardua tarea conseguir salir adelante con un proyecto que es muy común encontrar siendo usado por muchos negocios, pero muy gratificante haber conseguido acercarme un poco a aquellos sistemas de gestión hosteleros, pues me resultan una herramienta muy útil y fácil de usar, al igual que es muy requerida y bien monetizada.