GESTIÓN DE INVENTARIADO

MEMORIA

AUTORES:

Carlos Cicuéndez

Luís Gonzaga Muñoz

Miguel Escanciano

Valentín Circo

Víctor Choza

La clase Inventario (controlador) es el corazón de la aplicación, es la clase encargada de ejecutar el Stage principal y sus correspondientes Scenes.

```
@Override
public void start(Stage escenarioPrincipal) {

    //Debo hacerlo para que luego me funcione en l carga de escenas
    this.escenarioPrincipal = escenarioPrincipal;

    //Establezco el título
    this.escenarioPrincipal.setTitle("Gestión de inventariado");

    //Inicializo el layout principal
    initLayoutPrincipal();

    //Muestro la vista persona
    muestraVistaTabs();

}
```

initLayoutPrincipal ejecuta la vista principal que contiene un BorderPane, es el contenedor del resto de las escenas. Dentro de este método aparte de cargar la vísta, tambien permito a VistaPrincipalController una referencia de la clase Inventario, y por último intento cargar el último archivo XML guardado para cargar los productos.

```
public void initLayoutPrincipal() {

    //Cargo el layout principal a partir de la vista VistaPrincipal.fxml
    FXMLLoader loader = new FXMLLoader();
    URL location = Inventario.class.getResource("../view/VistaPrincipal.fxml");
    loader.setLocation(location);
    try {
        layoutPrincipal = loader.load();
    } catch (IOException ex) {
        Logger.getLogger(Inventario.class.getName()).log(Level.SEVERE, null, ex);
    }

    //Cargo y muestro la escena que contiene ese layout principal
    Scene escena = new Scene(layoutPrincipal);
    escenarioPrincipal.setScene(escena);

    //Doy al controlador acceso a la aplicación principal
    VistaPrincipalController controller = loader.getController();
    controller.setInventarioReferencia(this);

    escenarioPrincipal.show();

    //Intento cargar el último archivo abierto
    File archivo = getRutaArchivoProductos();
    if (archivo != null) {
        cargaProductos(archivo);
    }

}
```

muestraVistaTabs carga la vista VistaTabs dentro del layoutPrincipal. Cabe destacar que VistaTabs se carga con <fx:include> para que los controladores de los tabs vayan por separado (VistaTabsProductos, VistaTabsInformacion y VistaTabsEstadisticas) para asi poder dividir los controladores y las vistas de los diferentes tabs y que no todo se encuentre en VistaTabs. Por último permito a VistaTabs una referencia de la clase Inventario para poder cargar la tabla correctamente.

```
public void muestraVistaTabs() {

    FXMLLoader loader = new FXMLLoader();
    URL location = Inventario.class.getResource("../view/VistaTabs.fxml");
    loader.setLocation(location);
    try {
        vistaTabs = loader.load();
    } catch (IOException ex) {
        Logger.getLogger(Inventario.class.getName()).log(Level.SEVERE, null, ex);
    }

    layoutPrincipal.setCenter(vistaTabs);

    // get del controlador de VistaTabs para despues mandarlo al controlador del tab productoss (nested controllers para cada Tab)
    VistaTabsController controller = loader.getController();
    controller.setInventarioVistaTabs(this);

}
```

Su constructor es el encargado de crear los productos con sus correspondientes características (código de producto, nombre, foto, descripción..). Dichos productos se almacenan en un ObservableList

Cabe destacar que para la creación de la imagen se ha usado un método que recibe de parámetro una parte del nombre del archivo que luego será concatenado a la ruta específica de cada producto.

```
private ImageView crearImagen(String nombre) {
    ImageView imagen = new ImageView(getClass().getResource("../img/products/" + nombre + ".jpg").toExternalForm());
    imagen.setFitHeight(50);
    imagen.setFitWidth(50);
    imagen.setPreserveRatio(true);

    return imagen;
}
```

El método **guardaProductos** es el encargado de convertir el objeto Observable list que contiene los productos a un archivo XML, una vez convertido, se utiliza una clase intermedia Empaquetador que hace de contenedor del List (contiene los productos para convertirlos a XML), y un método para devolver el List para popular la tabla

```
//Guardo personas en un fichero
public void guardaProductos(File archivo) {

    try {
        //Contexto
        JAXBContext context = JAXBContext.newInstance(Empaquetador.class);
        Marshaller m = context.createMarshaller();
        m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);

        //Empaqueto los datos de las personas
        Empaquetador empaquetador = new Empaquetador();
        empaquetador.setProductos(productoss);

        //Marshall y guardo XML a archivo
        m.marshal(empaquetador, archivo);

        //Guardo la ruta del archivo en el registro
        setRutaArchivoProductos(archivo);
        System.out.println("Guardado " + archivo);

    } catch (Exception e) { // catches ANY exception
        //Muestro alerta
        Alert alerta = new Alert(Alert.AlertType.ERROR);
        alerta.setTitle("Error");
        alerta.setHeaderText("No se puede guardar en el archivo " + archivo.getPath());
        alerta.setContentText(e.toString());

        //css dialog pane
        DialogPane dialogAlert = alerta.getDialogPane();
        dialogAlert.getStylesheets().add(getClass().getResource("../css/modena_dark.css").toExternalForm());
        alerta.showAndWait();
    }
}
```

el método **setRutaArchivoProductos** lo que hace es guardar en las Preferencias la ruta del archivo para cuando se abra la aplicación se recuerde la ruta de guardado y el archivo XML cargue.

```java
//Guardo la ruta del archivo en las preferencias de usuario en Java
public void setRutaArchivoProductos(File archivo) {

    Preferences prefs = Preferences.userNodeForPackage(Inventario.class);
    if (archivo != null) {
        //Añado la ruta a las preferencias
        prefs.put("rutaArchivo", archivo.getPath());
        //Actualizo el título del escenario a partir del archivo
        escenarioPrincipal.setTitle("Gestión de inventariado - (" + archivo.getAbsolutePath() + ")");
    } else {
        //Elimino la ruta de las preferencias
        prefs.remove("rutaArchivo");
        //Actualizo el título del escenario quitando el nombre del archivo
        escenarioPrincipal.setTitle("Gestión de inventariado - No guardado");
    }

}
```

y por el contrario, **getRutaArchivosProductos** lo que hace es coger la ruta actual de las preferencias.

El método **cargaProductos** es el encargado de coger el archivo XML y convertirlo a un objeto de tipo List y continuación aplicará dichos objetos a la tabla. Para la realización de dicha conversión se usará un Wraper (Empaquetador)

```java
//Leo personas de un fichero
public void cargaProductos(File archivo) {

    try {
        //Contexto
        System.out.println("A cargar: " + archivo);
        JAXBContext context = JAXBContext.newInstance(Empaquetador.class);
        Unmarshaller um = context.createUnmarshaller();

        //Leo XML del archivo y hago unmarshall
        Empaquetador empaquetador = (Empaquetador) um.unmarshal(archivo);

        //Borro los anteriores
        productoss.clear();
        for (int i = 0; i < empaquetador.getProductos().size(); i++) {

            // añado los productos desde el Empaquetador que a su vez lo extrae del xml
            productoss.add(new Producto(empaquetador.getProductos().get(i).getCodigo(),
                    empaquetador.getProductos().get(i).getNombre(),
                    empaquetador.getProductos().get(i).getCategoria(),
                    empaquetador.getProductos().get(i).getStock(),
                    empaquetador.getProductos().get(i).getPrecio(),
                    crearImagenConRutaDesdeXML(empaquetador.getProductos().get(i).getRutaFoto()),
                    empaquetador.getProductos().get(i).getDescripcion(),
                    empaquetador.getProductos().get(i).getRutaFoto()));

        }
        //Guardo la ruta del archivo al registro de preferencias
        setRutaArchivoProductos(archivo);

    } catch (Exception e) {
        //Muestro alerta
        Alert alerta = new Alert(Alert.AlertType.ERROR);

        alerta.setTitle("Error");
        alerta.setHeaderText("Imposible leer archivo: " + archivo.getPath());
        alerta.setContentText(e.toString());
        //css dialog pane
        DialogPane dialogAlert = alerta.getDialogPane();
        dialogAlert.getStylesheets().add(getClass().getResource("../css/modena_dark.css").toExternalForm());
        alerta.showAndWait();

    }

}
```

La clase empaquetador (modelo) contiene el List de productos con los getters y setters para convertir de objeto a XML y viceversa. También contiene las anotaciones *XmlRootElement* que hace referencia al nombre raíz del xml y *XmlElement* hace referencia a cada producto

<productos>

<producto> …. <producto>

<producto> …. <producto>

<productos>

```java
@XmlRootElement(name = "productos")
public class Empaquetador {

    private List<Producto> productos;

    @XmlElement(name = "producto") //Opcional para el elemento especificado
    public List<Producto> getProductos() {
        return this.productos;
    }

    public void setProductos(List<Producto> productos) {
        this.productos = productos;
    }

}
```

La clase Producto (modelo) es la clase encargada de gestiónar la creacion de los productos gracias a los getters y setters. En su constructor se establecen todos los atributos.

```java
public class Producto {

    private StringProperty codigo;
    private StringProperty nombre;
    private StringProperty categoria;
    private IntegerProperty stock;
    private DoubleProperty precios;

    private ObjectProperty foto;
    private StringProperty rutaFoto;
    private StringProperty descripcion;
    private StringProperty fechaAlta;
    private StringProperty fechaModificacion;

    public Producto() {
        //no alterar o el xml fallará al cargarse
        this(null, null, null, 0, 0.0, null, null, null);
    }

    public Producto(String codigo, String nombre, String categoria, Integer stock, Double precios,
            ImageView foto, String descripcion, String rutaFoto) {

        this.codigo = new SimpleStringProperty(codigo);
        this.nombre = new SimpleStringProperty(nombre);
        this.categoria = new SimpleStringProperty(categoria);
        this.stock = new SimpleIntegerProperty(stock);
        this.precios = new SimpleDoubleProperty(precios);

        this.foto = new SimpleObjectProperty(foto);
        this.rutaFoto = new SimpleStringProperty(rutaFoto);
        this.descripcion = new SimpleStringProperty(descripcion);
        this.fechaAlta = new SimpleStringProperty(new SimpleDateFormat("dd/MM/yyy HH:mm").format(Calendar.getInstance().getTime()));
        this.fechaModificacion = new SimpleStringProperty(new SimpleDateFormat("dd/MM/yyy HH:mm").format(Calendar.getInstance().getTime()));

    }
```

La clase PDFHelper (modelo)es la encargada de la creación del archivo pdf, recibe diferentes parámetros, tales como: la ruta de la imagen, la descripcion, el código de barras en formato imagen asociado a dicho producto, el precio…

Se aplica la imagen del producto al pdf con sus diferentes carácteristicas

```java
public class PDFHelper {


    public static void informacionProductoPDF(File carpetaImagenesCodigosBarras,
            File selectedDirectory, String codigo, String rutaAbsoluta, String nombre,
            String descripcion, String categoria, Double precio) throws FileNotFoundException, DocumentException, BadElementException, IOException {

        String outputFile = "CodigosBarras_" + codigo + "_" + new SimpleDateFormat("ddMMyyyHHmmssSS").format(Calendar.getInstance().getTime()) + ".pdf";
        List<String> files = new ArrayList<String>();
        for (final File fileEntry : carpetaImagenesCodigosBarras.listFiles()) {
            if (fileEntry.getName().contains("jpg")) {
                files.add(fileEntry.getName());
            }
        }

        Document document = new Document();
        PdfWriter.getInstance(document, new FileOutputStream(new File(selectedDirectory, outputFile)));
        document.open();

        document.newPage();
        Image imagenProducto = Image.getInstance(new File(rutaAbsoluta).getAbsolutePath());
        imagenProducto.setAbsolutePosition(75, 675);
        imagenProducto.setBorderWidth(0);
        imagenProducto.scaleAbsolute(100, 100);
        document.add(imagenProducto);

        Paragraph s;
        s = new Paragraph("Nombre: " + nombre);
        s.setSpacingBefore(150);
        s.setIndentationLeft(35);
        document.add(s);
        s = new Paragraph("Descripción: " + descripcion);
        s.setIndentationLeft(35);
        document.add(s);
        s = new Paragraph("Categoria: " + categoria);
        s.setIndentationLeft(35);
        document.add(s);
        s = new Paragraph("Precio: " + precio + "€");
        s.setIndentationLeft(35);
        document.add(s);
```

Los códigos de barras se crean en una carpeta para el producto en seleccionado, se crea el numero de codigos de barras solicitados por el usuario y se añaden a un array list para despues aplicarlos al pdf.

```java
document.newPage();
int x = 0;
int y = 650;
for (String f : files) {
    Image imagenCodigoBarras = Image.getInstance(new File(carpetaImagenesCodigosBarras, f).getAbsolutePath());
    imagenCodigoBarras.setAbsolutePosition(x, y);
    imagenCodigoBarras.setBorderWidth(0);
    imagenCodigoBarras.scaleAbsolute(200, 200);
    document.add(imagenCodigoBarras);
    if (x == 400) {
        if (y == 50) {
            document.newPage();
            x = 0;
            y = 650;
        } else {
            x = 0;
            y -= 150;
        }
    } else {
        x += 200;
    }
}
document.close();
```

La clase VistaTabsController es el controlador intermediario entre los diferentes controladores de los tabs (VistaProductosTabController, VistaInformacionController y VistaEstadisticasController). Primero se realiza una referencia hacia estos controladores (nested controllers):

```
// controlador del tab de productos
@FXML
VistaProductosTabController productosController;

// controlador del tab de informacion
@FXML
VistaInformacionTabController informacionController;

// controlador del tab de estadisticas
@FXML
VistaEstadisticasTabController estadisticasController;
```

En el **initilize** de esta clase, se permite a los tabs VistaProductosTabController, VistaInformacionController y VistaEstadisticasController una referencia de la clase **VistaTabsController.**

Y el tab Informacion viene desactivado hasta que se selecciona un elemento de la tabla para mostrar su contenido.

```
@Override
public void initialize(URL url, ResourceBundle rb) {
    informacionTab.setDisable(true);

    // envio este controlador a VistaProductosTabController y a VistaInformacionTabController
    productosController.comunicacionControlador(this);
    informacionController.comunicacionControlador(this);
    estadisticasController.comunicacionControlador(this);
}
```

En el método **setInventarioVistaTabs** se recibe la referencia de Inventario y se transmite a VistaTabsController para mostrar el contenido de la tabla.

Y se ejecuta un método (setToolTips) para los tool tips de los tabs.

```
// redirigido a VistaProductosTabController desde Inventario
public void setInventarioVistaTabs(Inventario inventario) {

    this.inventario = inventario;
    System.out.println("Inventario capturado por VistaTabsController");

    productosController.setInventarioTabProductos(this.inventario);
    System.out.println("Inventario enviado a VistaProductosTabController");

    setToolTips();

}
```

El método **setFilaInformacion** es el encargado de recibir los datos de la fila seleccionada de VistaTabsController y enviarlo a VistaInformacionController para poder mostrar la informacion del producto seleccionado de la tabla. También al seleccionar un elemento de la tabla, se cambia automaticamente al tab Información.

```
public void setFilaInformacion(Producto newValue) {
    // envio objecto seleccionado de la tabla a VistaInformacionTabController
    informacionController.setFilaInformacion(newValue);
    // cambio de tab a informcion
    Tabs.getSelectionModel().select(informacionTab);

}
```

eliminarProductos tabla elimina la fila seleccionada si se esta situado en en el tab productos, si se esta en el tab informacion, se borra el producto y se cambia al tab productos.

```
public void eliminarProductoTabla(Producto filaSeleccionadaProducto) {
    // llamo a eliminar el producto de la tabla
    productosController.eliminarProductoTabla(filaSeleccionadaProducto);
    // cambio de tab a productos
    Tabs.getSelectionModel().select(productosTab);
}
```

La clase VistaProductosTabController es el tab contenedor de la tabla con los productos y los botones de añadir, borrar, editar e ir a detalles.

Su método initilize en primer lugar deshabilita los botones borrar, editar e ir a detalles si no hay selección en la tabla, acto seguido, se rellena la tabla mediante setCellValueFactory para asignarle la columna especifica para cada atributo del producto.

Y starteo los listeners de los botones y la tabla.

```
@Override
public void initialize(URL location, ResourceBundle resources) {
    // desativo boton detalles si no hay seleccion en la tabla
    detalles.setDisable(true);
    editar.setDisable(true);
    borrar.setDisable(true);

    // relleno filas
    System.out.println("init: " + location);
    codigoColumn.setCellValueFactory(cellData -> cellData.getValue().codigoProperty());
    nombreColumn.setCellValueFactory(cellData -> cellData.getValue().nombreProperty());
    stockColumn.setCellValueFactory(cellData -> cellData.getValue().stockProperty());
    precioColumn.setCellValueFactory(cellData -> cellData.getValue().preciosProperty());
    fechaAlta.setCellValueFactory(cellData -> cellData.getValue().fechaAltaProperty());
    imagenProducto.setCellValueFactory(cellData -> cellData.getValue().fotoProperty());

    listeners();

}
```

El listener de la tabla realiza:

- Al seleccionar cualquier fila, se habilitan los botones añadir, borrar, editar y se guarda la fila seleccionada en una variable global para poder hacer uso de los atributos de Producto en la clase VistaInformacionTabController
- Al hacer doble click sobre una fila, se cambia de tab VistaInformacionTabController y se habilita el tab Informacion (previamente desactivado) y se muestran los datos del producto seleccionado. Y le envio a VistaInformacionTabController pasando por VistaTabsController la variable con los atributos del producto.

```
public void listeners() {
    tablaProductos.getSelectionModel().selectedItemProperty().addListener(
            (observable, oldValue, newValue)
            -> {
        // activo boton al hacer seleccion en tabla
        detalles.setDisable(false);
        editar.setDisable(false);
        borrar.setDisable(false);

        this.filaSeleccionada = (Producto) newValue;

    });

    // doble click en la fila cambia de tab
    tablaProductos.setOnMousePressed(new EventHandler<MouseEvent>() {
        @Override
        public void handle(MouseEvent mouseEvent) {
            if (mouseEvent.getButton().equals(MouseButton.PRIMARY)) {
                if (mouseEvent.getClickCount() == 2) {
                    // activo los tabs
                    tabsControler.activarTabs();
                    tabsControler.setFilaInformacion(filaSeleccionada);
                    tabsControler.editarProducto(false);
                    System.out.println("Double clicked");
                }
            }
        }
    });
```

Al hacer click en el boton detalles se realiza lo mismo que haciendo doble click.

El listener del boton borrar pide confirmacion de borrado. Si el producto esta abierto en el tab Informacion y este es borrado, el tab informacion se deshabilita hasta que se selecciona otro producto de la tabla.

```
borrar.setOnMouseClicked((MouseEvent e)
        -> {
    String borrarString = " > Código: " + filaSeleccionada.getCodigo() + "\n > Nombre: " + filaSeleccionada.getNombre() + "\n > Stock: " + filaSeleccionada.getStock() + " uds."
            + "\n > Precio: " + filaSeleccionada.getPrecio() + " €" + "\n > Fecha Alta: " + filaSeleccionada.getFechaAlta();
    System.out.println(tablaProductos.getItems().size());

    Alert alert;

    alert = new Alert(AlertType.WARNING, "Contenido de la fila a borrar:\n\n" + borrarString + "\n\nBorrar definitivamente?", ButtonType.YES, ButtonType.NO, ButtonType.CANCEL);
    alert.setHeaderText("Confirmación de borrado");

    //css dialog pane
    DialogPane dialogAlert = alert.getDialogPane();
    dialogAlert.getStylesheets().add(getClass().getResource("../css/modena_dark.css").toExternalForm());
    alert.showAndWait();

    if (alert.getResult() == ButtonType.YES) {
        // si se selecciona un producto que está abierto en la tab información, se deshabilitan las tabs
        if (filaSeleccionada.getCodigo().equals(tabsControler.getCodigoInformacionProducto())) {
            System.out.println("asd");
            tabsControler.desactivarTabs();
        }

        Producto selectedItem = tablaProductos.getSelectionModel().getSelectedItem();
        if (selectedItem != null) {
            inventario.getProductos().remove(selectedItem);
        }
    }

});
```

Al borrar un producto, se ejecuta un metodo que recoge el producto seleccionado y se lo envia a VistaTabsControler y a su vez a VistaEstadisticasController para que el producto sea borrado del grafico y este pueda actualizarse.

```
Producto selectedItem = tablaProductos.getSelectionModel().getSelectedItem();
// actualizo grafica con los productos despues del borrado
tabsControler.borrarProductoChart(selectedItem);
```

El boton de añadir envia la peticion de adición de producto a VistaTabsController que este a su vez lo restransmie a VistaInformacionTabs.

```
public void editAddProducto(boolean mode, String isAddOrEdit) {
    informacionController.isAddOrEdit = isAddOrEdit;
    informacionController.modoEditAdd(mode);
}
```

Dicho método identifica si se añade un producto o se edita uno existente (isAddOrEdit), si se añade, todos los campos del tab Informacion se resetean y se permite introducir los datos, si es edición, se cargan los datos del producto elegido

```
if (isAddOrEdit.equals("Add")) {
    imagenProducto.setImage(null);
    idProducto.setText("");
    nombreProducto.setText("");
    precioProducto.setText("");
    descripcionProducto.setText("");
    categoriaProducto.setText("");
    stockProducto.setText("");
    fechaAltaProducto.setText("");
    fechaModificacionProducto.setText("");
    comboBoxCodigosBarras.setVisible(false);
    crear.setVisible(false);
```

Al presionar el botón de guardado, se filtran los datos del nuevo producto introducido para saber si el producto es único y en formato correcto.

El método **setInventarioTabProductos** permite a VistaTabsController tener una referencia de la clase Inventario, rellena la tabla con todos los productos de Inventario, gestiona el combo box que permite visualizar por categorias los productos mediante un Sorted List. Cabe destacar que se crea un Filtered List apartir de los productos del Inventario, despues un Sorted List a partir del Filtered List y por último se actualiza la tabla con los productos del sorted list.

```
//Es llamado por la apliación principal para tener una referencia de vuelta de sí mismo
public void setInventarioTabProductos(Inventario inventario) {

    this.inventario = inventario;
    //Añado la lista obervable a la tabla
    tablaProductos.setItems(this.inventario.getProductos());

    filteredData = new FilteredList<>(inventario.getProductos());

    categoria.setOnAction((t) -> {

        //Se hace scroll hasta arriba para evitar errores
        tablaProductos.scrollTo(0);

        String categoriaElegida = categoria.getValue().toString();
        filteredData.setPredicate(product -> {
            if (categoriaElegida == null || categoriaElegida.isEmpty() || categoriaElegida.toLowerCase().equals("todas")) {
                return true;
            }

            if (product.getCategoria().toLowerCase().contains(categoriaElegida.toLowerCase())) {
                return true;
            }
            return false;
        });
    });
    SortedList<Producto> sortedData = new SortedList<>(filteredData);
    sortedData.comparatorProperty().bind(tablaProductos.comparatorProperty());
    tablaProductos.setItems(sortedData);

    rellenarComboBox();

}
```

rellenarComboBox rellena el combo box con las categorias disponibles (Natacion, Beisbol..), para ello, se recorren todas las categorias de los productos, y si esta duplicada no se tiene en cuenta. Se añaden las categorias solo una vez para posteriormente poder filtrar la lista por el criterio seleccionado en el combo box

```
private void rellenarComboBox() {
    ArrayList<Producto> productos = new ArrayList<>(inventario.getProductos());
    ArrayList<String> categorias = new ArrayList<>();

    categorias.add("Todas");
    for (int i = 0; i < productos.size(); i++) {
        boolean repetido = false;
        for (int j = 0; j < categorias.size(); j++) {
            if (categorias.get(j).equals(productos.get(i).getCategoria())) {
                repetido = true;
            }
        }
        if (!repetido) {
            categorias.add(productos.get(i).getCategoria());
        }
    }

    for (int i = 0; i < categorias.size(); i++) {
        categoria.getItems().add(categorias.get(i));
    }
}
```

El método **comunicacionControlador** permite a VistaProductosTabController tener una referencia de VistaTabsControlle.

La clase **VistaInformacionTabController** es el tab encargado de mostrar la informacion detallada de un producto y permite el guardado de un pdf con la información del producto y sus códigos de barras.

El método **comunicacionControlador** permite a esta clase tener una referencia de VistaTabsController

```
public void comunicacionControlador(VistaTabsController tabsController) {
    this.tabsController = tabsController;
}
```

El método **setFilaInformacion** recibe la fila seleccionada de VistaTabController que a su vez lo ha recibido de VistaProductosTabController. Y se rellenan los campos correspondientes al producto seleccionado.

```
// recibo la fila seleccionada de VistaTabController que a su vez lo ha recibido de VistaProductosTabController
public void setFilaInformacion(Producto newValue) {
    this.filaSeleccionadaProducto = newValue;
    System.out.println(filaSeleccionadaProducto.getCodigo());

    imagenProducto.setImage(new Image(getClass().getResource(filaSeleccionadaProducto.getRutaFoto()).toExternalForm()));
    nombreProducto.setText(filaSeleccionadaProducto.getNombre());
    precioProducto.setText(String.valueOf(filaSeleccionadaProducto.getPrecio()));
    descripcionProducto.setText(filaSeleccionadaProducto.getDescripcion());
    categoriaProducto.setText(filaSeleccionadaProducto.getCategoria());
    stockProducto.setText(String.valueOf(filaSeleccionadaProducto.getStock()));
    fechaAltaProducto.setText(String.valueOf(filaSeleccionadaProducto.getFechaAlta()));
    fechaModificacionProducto.setText(String.valueOf(filaSeleccionadaProducto.getFechaModificacion()));

    int numeroCodigosBarras = 100;
    if (filaSeleccionadaProducto.getStock() < numeroCodigosBarras) {
        numeroCodigosBarras = filaSeleccionadaProducto.getStock();
    }
    for (int i = 0; i < numeroCodigosBarras; i++) {
        comboBoxCodigosBarras.getItems().add(i + 1);
    }
}
```

El método **initalize** ejecuta los listeners:

- Borrar producto: borra el producto pidiendo confirmación

```
borrar.setOnMouseClicked(e
        -> {
    String borrarString = " > Código: " + filaSeleccionadaProducto.getCodigo() + "\n > Nombre: " + filaSeleccionadaProducto.getNombre()
            + "\n > Stock: " + filaSeleccionadaProducto.getStock() + " uds."
            + "\n > Precio: " + filaSeleccionadaProducto.getPrecio() + " €" + "\n > Fecha Alta: " + filaSeleccionadaProducto.getFechaAlta();

    Alert alert;

    alert = new Alert(Alert.AlertType.WARNING, "Contenido de la fila a borrar:\n\n" + borrarString + "\n\n¿Borrar definitivamente?",
            ButtonType.YES, ButtonType.NO, ButtonType.CANCEL);

    alert.setHeaderText("CONFIRMACIÓN DE BORRADO");

    //css dialog pane
    DialogPane dialogAlert = alert.getDialogPane();
    dialogAlert.getStylesheets().add(getClass().getResource("../css/modena_dark.css").toExternalForm());
    alert.showAndWait();

    if (alert.getResult() == ButtonType.YES) {
        tabsController.eliminarProductoTabla(filaSeleccionadaProducto);
    }

});
```

- Editar: habilita el modo de edicion para modificar un producto

- Cancelar: cancela el editado y establece los valores antes de empezar el editado

```
cancelar.setOnMouseClicked(e -> {
    imagenProducto.setImage(new Image(getClass().getResource(rutaOld).toExternalForm()));
    nombreProducto.setText(nombreOld);
    precioProducto.setText(precioOld);
    descripcionProducto.setText(descripcionOld);
    categoriaProducto.setText(categoriaOld);
    stockProducto.setText(stockOld);
    modoEditar(false);
});
```

- Guardar: guardar producto modificado. En primer lugar se tiene un filtro de errores para asegurar que en los campos de edición se inserte el formato adecuado, si los datos son correctos, se aplican los cambios

```
guardar.setOnMouseClicked((MouseEvent e) -> {
    System.out.println("Guardar");
    String erroresString = "";

    // errores nombre producto
    if (nombreProducto.getText().isEmpty()) {
        erroresString += " - El nombre no puede quedar vacío\n";
        nombreProducto.setUnFocusColor(Color.RED);
    } else {
        nombreProducto.setUnFocusColor(Color.rgb(42, 46, 55));
    }

    // errores precio producto
    if (!precioProducto.getText().isEmpty()) {
        try {
            Double valor = Double.valueOf(precioProducto.getText());
            precioProducto.setUnFocusColor(Color.rgb(42, 46, 55));
        } catch (NumberFormatException ex) {
            erroresString += " - El precio debe ser un número\n";
            precioProducto.setUnFocusColor(Color.RED);
        }
    } else {
        erroresString += " - El precio no puede quedar vacío\n";
        precioProducto.setUnFocusColor(Color.RED);
    }

    //errores descripcion producto
    if (descripcionProducto.getText().isEmpty()) {
        erroresString += " - La descripción no puede quedar vacía\n";
        descripcionProducto.setUnFocusColor(Color.RED);
    } else {
        descripcionProducto.setUnFocusColor(Color.rgb(42, 46, 55));
    }

    //errores categoria producto
    if (categoriaProducto.getText().isEmpty()) {
        erroresString += " - La categoría no puede quedar vacía\n";
        categoriaProducto.setUnFocusColor(Color.RED);
    } else {
        categoriaProducto.setUnFocusColor(Color.rgb(42, 46, 55));
    }
```

Cabe destacar que al presionar el boton de guardar, se recoge el campo actualizado del stock para mandarselo a VistaTabsController y este reenviarlo a VistaEstadisticasController para actualizar el grafico

```
// el stock actualizado se lo mando a VistaTabsController para que este a su vez actualize el grafico en EstadisticasController
tabsController.stockActualizado(Integer.valueOf(stockProducto.getText()));
```

```
//errores stock producto
if (!stockProducto.getText().isEmpty()) {
    try {
        int valor = Integer.valueOf(stockProducto.getText());
        stockProducto.setUnFocusColor(Color.rgb(42, 46, 55));
    } catch (NumberFormatException ex) {
        erroresString += " - El stock debe ser un número\n";
        stockProducto.setUnFocusColor(Color.RED);
    }
} else {
    erroresString += " - El stock no puede quedar vacio\n";
    stockProducto.setUnFocusColor(Color.RED);
}

if (!erroresString.isEmpty()) {
    Alert alert;
    alert = new Alert(Alert.AlertType.WARNING, "Se han encontrado los siguientes errores:\n\n" + erroresString + "\n\nResuelva los errores para poder continuar", ButtonType.OK);
    alert.setHeaderText("ERROR");
    DialogPane dialogAlert = alert.getDialogPane();
    dialogAlert.getStylesheets().add(VistaInformacionTabController.this.getClass().getResource("../css/modena_dark.css").toExternalForm());
    alert.showAndWait();
} else {
    System.out.println("Sin errores. Guardando...");
    filaSeleccionadaProducto.setNombre(nombreProducto.getText());
    filaSeleccionadaProducto.setPrecio(Double.valueOf(precioProducto.getText()));
    filaSeleccionadaProducto.setDescripcion(descripcionProducto.getText());
    filaSeleccionadaProducto.setCategoria(categoriaProducto.getText());
    filaSeleccionadaProducto.setStock(Integer.valueOf(stockProducto.getText()));
    filaSeleccionadaProducto.setFechaModificacion(new SimpleDateFormat("dd/MM/yyy HH:mm").format(Calendar.getInstance().getTime()));

    fechaModificacionProducto.setText(String.valueOf(filaSeleccionadaProducto.getFechaModificacion()));

    modoEditar(false);
    tabsController.actualizarTabla();
}
```

- Crear: Guarda pdf con inf. del producto y genera códigos de barras. En primer lugar se comprueba que el numero de códigos de barras no excedan el numero de stock y que debe haber un integer obligatorio para el pdf.

```
crear.setOnMouseClicked((MouseEvent e) -> {
    String erroresString = "";

    //errores cantidad código de barras
    if (comboBoxCodigosBarras.getValue() != null) {
        if (Integer.valueOf(comboBoxCodigosBarras.getValue().toString()) <= filaSeleccionadaProducto.getStock()) {
            try {
                int valor = Integer.valueOf(comboBoxCodigosBarras.getValue().toString());
                comboBoxCodigosBarras.setUnFocusColor(Color.rgb(42, 46, 55));
            } catch (NumberFormatException ex) {
                erroresString += " - La cantdad debe ser un número\n";
                comboBoxCodigosBarras.setUnFocusColor(Color.RED);
            }
        } else {
            erroresString += " - La cantidad no puede ser mayor que el stock\n";
            comboBoxCodigosBarras.setUnFocusColor(Color.RED);
        }
    } else {
        erroresString += " - La cantidad no puede quedar vacía\n";
        comboBoxCodigosBarras.setUnFocusColor(Color.RED);
    }

    if (!erroresString.isEmpty()) {
        Alert alert;
        alert = new Alert(Alert.AlertType.WARNING, "Se han encontrado los siguientes errores:\n\n" + erroresString + "\n\nResuelva los errores para poder continuar", ButtonType.OK);
        alert.setHeaderText("ERROR");
        DialogPane dialogAlert = alert.getDialogPane();
        dialogAlert.getStylesheets().add(VistaInformacionTabController.this.getClass().getResource("../css/modena_dark.css").toExternalForm());
        alert.showAndWait();
```

Acto seguido se permite al usuario elegir dónde guardar el pdf (el nombre se autogenera con el id del producto y el número de código de barras)

```
int numCodigos = Integer.parseInt((String) comboBoxCodigosBarras.getValue());

DirectoryChooser chooser = new DirectoryChooser();
chooser.setTitle("JavaFX Projects");
File defaultDirectory = new File("C:/");
chooser.setInitialDirectory(defaultDirectory);
File selectedDirectory = chooser.showDialog(null);

if (selectedDirectory != null && selectedDirectory.isDirectory()) {
    File carpetaImagenesCodigosBarras = new File("ImagenesCodigosBarras/" + numCodigos + "CodigosBarras_" + new SimpleDateFormat("ddMMyyyyHHmmssSS").format(Calendar.getInstance().getTime()));
    carpetaImagenesCodigosBarras.mkdirs();
    for (int i = 0; i < numCodigos; i++) {
        try {

            String numeroCodigo = (filaSeleccionadaProducto.getCodigo()).substring(2);
            for (int j = 0; j < 10; j++) {
                numeroCodigo += 0 + (int) (Math.random() * ((9 - 0) + 1));
            }

            JBarcodeBean barcode = new JBarcodeBean();
            barcode.setCodeType(new Interleaved25());
            barcode.setCode(numeroCodigo);
            barcode.setCheckDigit(true);

            BufferedImage bufferedImage = barcode.draw(new BufferedImage(400, 400, BufferedImage.TYPE_INT_RGB));
            File file = new File(carpetaImagenesCodigosBarras.getPath() + "/" + filaSeleccionadaProducto.getCodigo() + "_" + (i + 1) + ".jpg");
            ImageIO.write(bufferedImage, "jpg", file);

        } catch (IOException ex) {
        }

    }
    try {
        PDFHelper.informacionProductoPDF(carpetaImagenesCodigosBarras, selectedDirectory,
                filaSeleccionadaProducto.getCodigo(), getRutaAbsoluta(filaSeleccionadaProducto.getRutaFoto()),
                filaSeleccionadaProducto.getNombre(), filaSeleccionadaProducto.getDescripcion(),
                filaSeleccionadaProducto.getCategoria(), filaSeleccionadaProducto.getPrecio());
    } catch (DocumentException ex) {
    } catch (IOException ex) {
    }
}
```

El método **modoEditar** es el encargado de controlar si se ha entrado en el modo de editar despues de pulsar el boton editar producto. En caso de cancelar la edición, se reestablecen los datos anteriormente guardados antes de empezar la edicion. Y algunos detalles gráficos para saber si se esta en modo edicion o no, como el efecto underline al seleccionar una fila a editar.

```java
public void modoEditar(boolean mode) {
    if (mode) {
        // se guardan los datos anteriores a la edición
        rutaOld = filaSeleccionadaProducto.getRutaFoto();
        nombreOld = nombreProducto.getText();
        precioOld = precioProducto.getText();
        descripcionOld = descripcionProducto.getText();
        categoriaOld = categoriaProducto.getText();
        stockOld = stockProducto.getText();

        nombreProducto.setFocusColor(Color.rgb(230, 230, 0));
        nombreProducto.setUnFocusColor(Color.rgb(42, 46, 55));
        precioProducto.setFocusColor(Color.rgb(230, 230, 0));
        precioProducto.setUnFocusColor(Color.rgb(42, 46, 55));
        descripcionProducto.setFocusColor(Color.rgb(230, 230, 0));
        descripcionProducto.setUnFocusColor(Color.rgb(42, 46, 55));
        categoriaProducto.setFocusColor(Color.rgb(230, 230, 0));
        categoriaProducto.setUnFocusColor(Color.rgb(42, 46, 55));
        stockProducto.setFocusColor(Color.rgb(230, 230, 0));
        stockProducto.setUnFocusColor(Color.rgb(42, 46, 55));
    } else {
        nombreProducto.setFocusColor(Color.TRANSPARENT);
        nombreProducto.setUnFocusColor(Color.TRANSPARENT);
        precioProducto.setFocusColor(Color.TRANSPARENT);
        precioProducto.setUnFocusColor(Color.TRANSPARENT);
        descripcionProducto.setFocusColor(Color.TRANSPARENT);
        descripcionProducto.setUnFocusColor(Color.TRANSPARENT);
        categoriaProducto.setFocusColor(Color.TRANSPARENT);
        categoriaProducto.setUnFocusColor(Color.TRANSPARENT);
        stockProducto.setFocusColor(Color.TRANSPARENT);
        stockProducto.setUnFocusColor(Color.TRANSPARENT);
    }

    nombreProducto.setEditable(mode);
    precioProducto.setEditable(mode);
    descripcionProducto.setEditable(mode);
    categoriaProducto.setEditable(mode);
    stockProducto.setEditable(mode);

    anadir.setVisible(!mode);
    borrar.setVisible(!mode);
    editar.setVisible(!mode);
    cancelar.setVisible(mode);
    guardar.setVisible(mode);
}
```

La clase VistaPrincipalController es el controlador donde se gestionan las acciones del toolbar (ya que solo esta dicho elemento en la vista). Dichas acciones son principalmente para abir y guardar la lista de productos en un archivo XML, especificado por el usuario. El siguente item es para descargar la guia de usuario en pdf y los autores.

El metodo **nuevo** es utilizado para vaciar la lista de productos y setear las preferencias a null.

El método **abrir** abre un file chooser para escoger el fchero xml para leer los productos. En caso que sea un fichero válido se ejecutará cargaProductos que procerá a leer el archivo xml y convertirlo a objeto de tipo Producto. Por el contrario el método **guardar** guarda los productos en un archivo xml (ruta elegida por el usuario) asegurandose que la extensión es correcta.

```
private void nuevo() {
    inventario.getProductos().clear();
    inventario.setRutaArchivoProductos(null);
}

//Abro un File Chooser para que el usario seleccione una libreta
@FXML
private void abrir() {
    FileChooser fileChooser = new FileChooser();

    //Filtro para la extensión
    FileChooser.ExtensionFilter extFilter = new FileChooser.ExtensionFilter(
            "XML files (*.xml)", "*.xml");
    fileChooser.getExtensionFilters().add(extFilter);

    //Muestro el diálogo de guardar
    File archivo = fileChooser.showOpenDialog(inventario.getPrimaryStage());

    if (archivo != null) {
        inventario.cargaProductos(archivo);
    }
}

//Guardar
@FXML
private void guardar() {
    File archivo = inventario.getRutaArchivoProductos();
    if (archivo != null) {
        inventario.guardaProductos(archivo);
    } else {
        guardarComo();
    }
}
```

Cabe destacar que si al presionar guardar, no hay un archivo xml ya cargado, se ejecutará el método **guardarComo** que permite al usuario crear un nuevo archivo xml

```
//Abro un File Chooser para guardar como
@FXML
private void guardarComo() {

    FileChooser fileChooser = new FileChooser();

    //Filtro para la extensión
    FileChooser.ExtensionFilter extFilter = new FileChooser.ExtensionFilter(
            "XML files (*.xml)", "*.xml");
    fileChooser.getExtensionFilters().add(extFilter);

    //Muestro el diálogo de guardar
    File archivo = fileChooser.showSaveDialog(inventario.getPrimaryStage());

    if (archivo != null) {
        //Me aseguro de que tiene la extensión correcta
        if (!archivo.getPath().endsWith(".xml")) {
            archivo = new File(archivo.getPath() + ".xml");
        }
        inventario.guardaProductos(archivo);
    }
}
```

El método **descargarGuiaPDF** descarga la guia desde un enlace externo (google drive) en la ruta elegida por el usuario.

```
@FXML
private void descargarGuiaPDF() throws MalformedURLException, FileNotFoundException, IOException {

    DirectoryChooser chooser = new DirectoryChooser();
    chooser.setTitle("Elige destino de la guia en pdf");
    chooser.setInitialDirectory(new File("c:/"));
    File diectorio = chooser.showDialog(inventario.getPrimaryStage());


    try {
        System.out.println("Descargando guía PDF en: " + diectorio + "\\Guia_GestorInventariado.pdf");

        URL url = new URL("https://drive.google.com/uc?authuser=0&id=1wO7gEoBFnwU_lICo54L85bluTUh66JBS&export=download");
        InputStream in = url.openStream();
        OutputStream fos = new FileOutputStream(diectorio + "\\Guia_GestorInventariado.pdf");

        int length = -1;
        byte[] buffer = new byte[1024];

        while ((length = in.read(buffer)) != -1) {
            fos.write(buffer, 0, length);
        }
```

La clase **VistaEstadisticasController** es la clase encargada que mostrar el gráfico en la pestaña Estadísticas. Dicha pestaña se actualiza al borrar, editar o añadir producto.

El método **comunicacionControlador** sirve para tener una referencia de VistaTabs, y setInventarioTabProductos para tener una referencia de inventario para el array de productos.

```
public void comunicacionControlador(VistaTabsController tabsController){
    this.tabsController = tabsController;
}

public void setInventarioTabProductos(Inventario inventario) {

    this.inventario = inventario;
    addDatosGrafica();
```

El método **quitarProducto** es utilizado para resetear el grafico al borrar un producto, una vez reseteadas las variables y quitado el producto, se invoca addDatosGrafica que rellena el grafico con los datos actuales despues del borrado. A su vez, el método actualizarStock realiza algo parecido, al presionar el boton guardar al terminar la edicion de un producto, se recoge el stock y se envia a esta clase.

```
public void quitarProducto(Producto producto){

    serieDatos.getData().clear();
    this.productos_datos.remove(producto);
    this.nombresProducto.clear();
    System.out.println(producto.getNombre());
    datosEjeX.clear();
    grafica.getData().clear();

    System.out.println("Producto quitado");
    addDatosGrafica();
}

public void actualizarStock(int stock){
            serieDatos.getData().clear();
            grafica.getData().clear();
            addDatosGrafica();
```

El método **addDatosGrafica** se utiliza para popular los ejes X e Y con los datos del producto. Se guarda en arrayLists cada elemento del producto, en este caso, el stock y el nombre. Acto seguido se construyen las barras con los datos establecidos.

```
public void addDatosGrafica(){

    productos_datos = inventario.getProductos();
    nombresProducto = new ArrayList<>();

    System.out.println("Tamaño actual " + productos_datos.size());

    for (int i = 0; i < productos_datos.size(); i++) {
        Producto producto = productos_datos.get(i);
        nombresProducto.add(producto.getNombre());
    }

    // damos formato a los datos del eje X, convertimos de ArrayList a ObservableList,
    //tiene que ser así porque la gráfica los requiere en dicho formato
    datosEjeX = FXCollections.observableArrayList(nombresProducto);
    ejeX.setCategories(datosEjeX);

    // ahora vamos con el eje Y
    ejeY.setLabel("Stock");
    serieDatos = new XYChart.Series<>();
    serieDatos.setName("Stock por producto");
    for (int i = 0; i < productos_datos.size(); i++) {
        Producto producto = productos_datos.get(i);
        serieDatos.getData().add(new XYChart.Data<>(producto.getNombre(), producto.getStock()));
    }

    ejeX.setTickLabelRotation(270);
    ejeX.setTickLabelFill(Color.CHOCOLATE);
    ejeY.setTickLabelFill(Color.CHOCOLATE);

    // añadimos los datos a la gráfica
    grafica.getData().add(serieDatos);
}
```

El método **gráficoPDF** se encarga de hacer un print del grafico actual para posteriormente exportarlo en la ruta elegida por el usuario en formato png y pdf. Se abrirá dicho archivo al completarse la descarga.

```
void graficoPDF(MouseEvent event) {
    try {
        //Generamos una imagen a partir de la grafica
        File file;

        DirectoryChooser chooser = new DirectoryChooser();
        chooser.setTitle("Elige destino de la guia en pdf");
        chooser.setInitialDirectory(new File("c:/"));
        File diectorioImagenGrafica = chooser.showDialog(inventario.getPrimaryStage());

        String pathImagenGrafica = diectorioImagenGrafica + "\\grafica_productos_stock.png";
        String pathPDF = diectorioImagenGrafica + "\\grafica_productos_stock.pdf";

        // genera imagen a partir de la grafica
        WritableImage image = grafica.snapshot(new SnapshotParameters(), null);
        file = new File(pathImagenGrafica);
        ImageIO.write(SwingFXUtils.fromFXImage(image, null), "png", file);

        //Generamos un pdf a partir de la imagen creada anteriormente
        Document document = new Document();
        PdfWriter.getInstance(document, new FileOutputStream(new File(pathPDF)));
        document.open();

        Image imagenGrafico = Image.getInstance(new File(pathImagenGrafica).getAbsolutePath());
        imagenGrafico.setAbsolutePosition(10, 500);
        imagenGrafico.setBorderWidth(0);
        imagenGrafico.scaleAbsolute(500, 200);
        document.add(imagenGrafico);
        document.close();
```

Ideas de mejora

El proyecto podría realizarse utilizando bases de datos relacionales.

Sistema de autenticación por usuarios con panel de control.

Gestión de incidencias y resoluciones para los productos.

Sistema de registro para proveedores, los cuales puedan añadir productos.