



**CICLO: [DAM]**  
**MÓDULO DE [PROGRAMACIÓN]**

# **[Tarea N° 08]**

**Alumno:**  
**[Juan Carlos Filter Martín]**  
**[15456141A]**

## Contenido

<b>1. Documentos que se adjuntan a este informe.....</b>	<b>3</b>
<b>2. Descripción de la tarea.....</b>	<b>3</b>
<b>3. RA08_c) Se han instalado sistemas gestores de bases de datos orientados a objetos.....</b>	<b>4</b>
A) Proyecto Tarea8: Crear un proyecto llamado Tarea8.....	4
B) Preparar para trabajar con una BDOO DB4o.....	5
Descargar Bd4o.....	5
Añadir la biblioteca de Db4o al proyecto java de NetBeans.....	7
<b>4. RA08_e) Se han creado bases de datos y las estructuras necesarias para el almacenamiento de objetos.....</b>	<b>11</b>
A) Fichero Tarea8: Se va a trabajar con una BD que almacenaremos en un fichero “Tarea8”.....	11
<b>5. RA08_f) Se han programado aplicaciones que almacenen objetos en las bases de datos creadas.....</b>	<b>12</b>
A) Clase Artículo: para crear los objetos que almacenaremos en la BD.....	12
<b>6. RA08_g) Se han realizado programas para recuperar, actualizar y eliminar objetos de las bases de datos.....</b>	<b>14</b>
A) Clase GestorBD: Toda la gestión de la información se hará a través de una clase.....	14
B) Clase ControladorErrores: Se va a crear una clase para controlar los Errores.....	19
C) JFrame form: Parte gráfica para que sean llamado los distintos eventos de la interfaz.....	20
<b>7. RA08_h) Se han realizado programas para almacenar y gestionar tipos de datos estructurados, compuestos y relacionados.....</b>	<b>22</b>
Botón Insertar.....	22
Interfaz gráfica.....	22
Código.....	23
Botón Buscar.....	25
Interfaz gráfica.....	25
Código.....	26
Botón Borrar.....	28
Interfaz gráfica.....	28
Código.....	29
Botón Modificar.....	31
Interfaz gráfica.....	31
Código.....	32
Botón Mostrar.....	34
Interfaz gráfica.....	34
Código.....	35
Botón Salir.....	37
Interfaz gráfica.....	37

## 1. Documentos que se adjuntan a este informe.

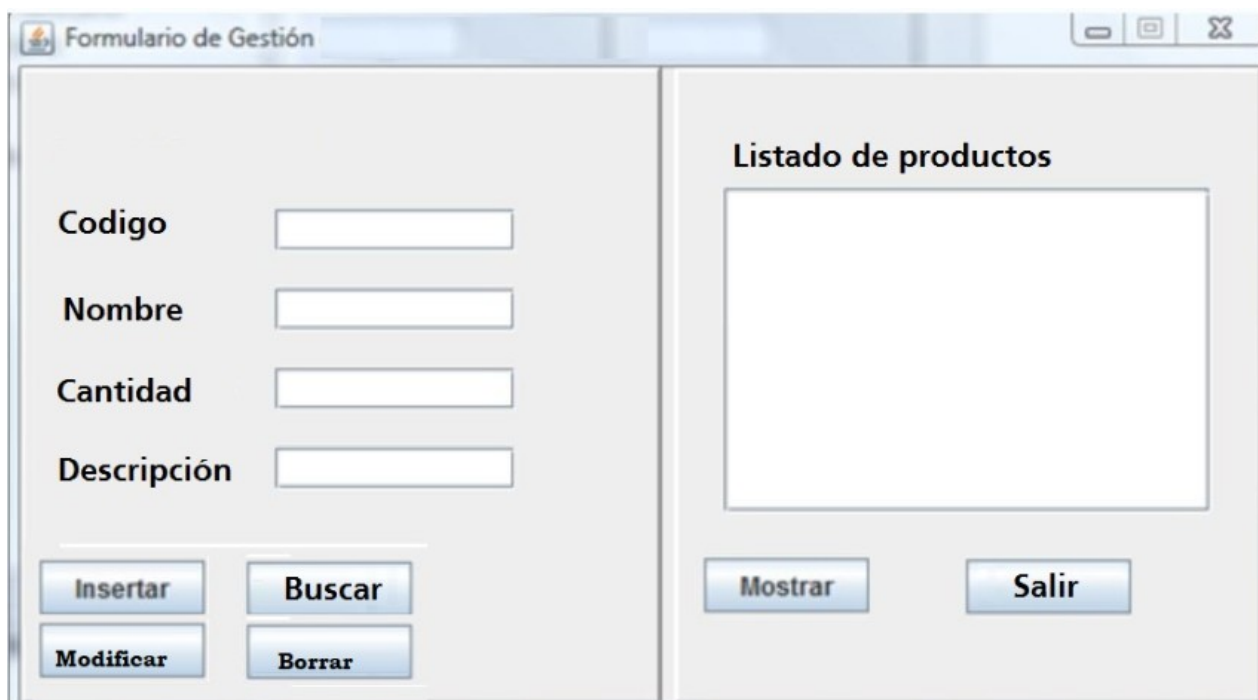
A continuación se detallan los documentos que componen la presente entrega de la tarea:

1. Informe de elaboración de la tarea.
2. Proyecto Java "Tarea8"

## 2. Descripción de la tarea

Realizar una aplicación Java que tendrá un entorno gráfico tal como el que se muestra en la siguiente figura:

*(Orientativo, se puede realizar el diseño que desee siempre que la aplicación no pierda funcionalidad)*



En cada producto se debe introducir (codigoProducto, Nombre, Cantidad, Descripción)

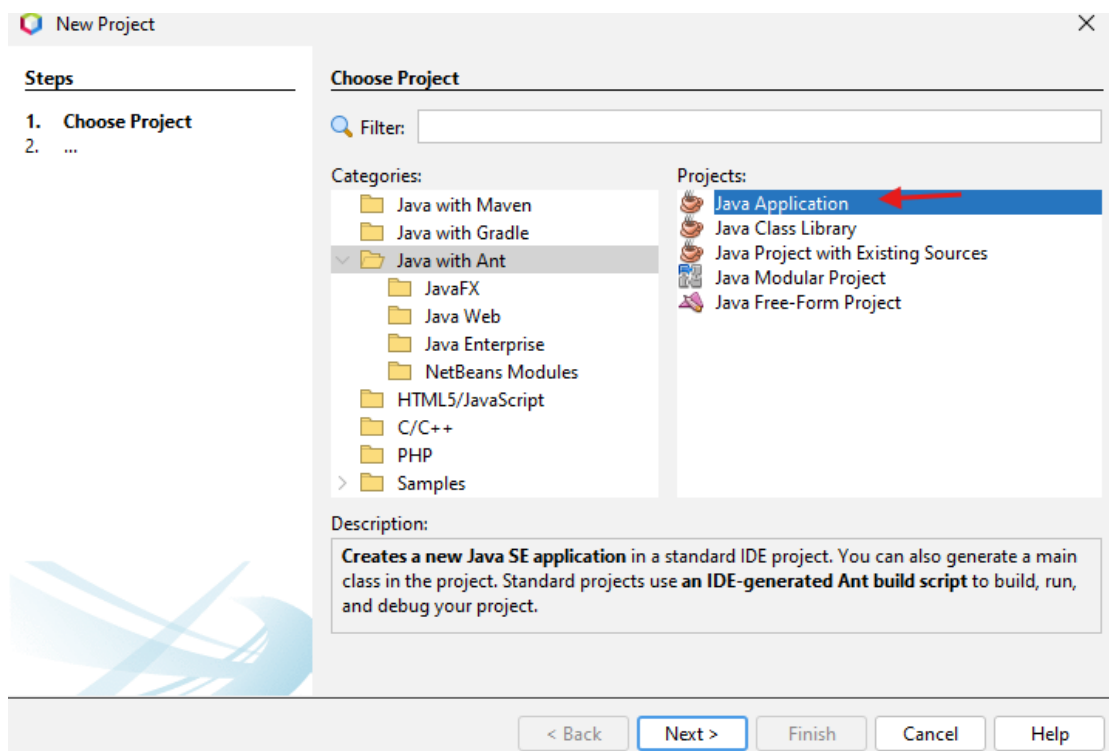
CodigoProducto	Nombre	Cantidad	Descripción
1	Tornillo	700	Tornillo pequeño
2	Tuerca	69	Tuerca para tornillo pequeño
3	Clavo	100	Clavo de cabeza plana
4	Alambre	300	Rollo de alambre

*Esto es un ejemplo, se pueden crear los datos que se deseen*

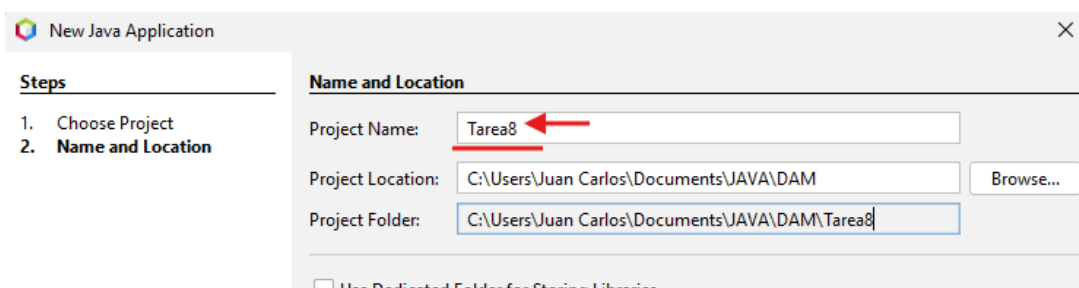
3. RA08\_c) Se han instalado sistemas gestores de bases de datos orientados a objetos.

***A) Proyecto Tarea8: Crear un proyecto llamado Tarea8.***

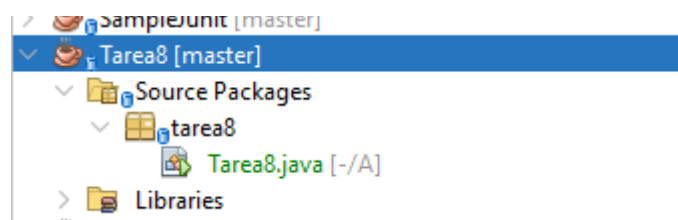
File > New Project > Java with Ant > **Java Application**



Ponemos de nombre al proyecto **Tarea8**



El proyecto ya estaría creado



## B) Preparar para trabajar con una BDOO DB4o.

### × Descargar Bd4o.

Bd4o es un motor de base de datos orientado a objetos. Este está disponible para entornos Java

Para hacer uso de esta base de datos tendremos que entrar en la web del fabricante y descargar la última versión



He tenido problemas para entra a la web. Esta captura está Realizada desde el PDF del tema 8 y se ha descargado Bd4o desde el campus

ABOUT	DEVELOPERS	CUSTOMERS
Industry Solutions	Product Information	Customers and Partners
News and Events	Company	

### db4o Base de Objetos de Código Abierto

- Nativa a Java y .NET
- 100% orientada a objetos, sin mapeo objeto-relacional
- Diseñada para uso embebido
- De código abierto y libre bajo la GPL

**La Ventaja que Necesita en sus Tiempos de Desarrollo.**

Incorpore a su producto db4o (el motor de base de objetos nativo para Java y .NET) y almacene incluso las estructuras de objetos más complejas con una sola línea de código!

db4o reduce el tiempo y costo de desarrollo, provee un desempeño superior, y no requiere de un DBA.

**Descargue y pruebe** la base de objetos más popular del mundo! Es libre y de código abierto bajo la GPL, no requiere de registración!

**Unase a la Comunidad de Bases de Objetos más Grande del Mundo Hoy!**



*"La base de datos de código abierto db4o se integra mejor en nuestro ambiente en comparación a la*

Descargar gratis y comenzar inmediatamente su evaluación:

- **db4o 8.0 for Java** (40.9 mb)

Base de datos para objetos con licencia [GPL](#).

**Enlaces Rápidos en Español**

- [Whitepaper Bases de Objetos](#)
- [Tutorial Java | C#](#)
- [Caso de Exito: INORA](#)
- [Foro de usuarios](#)
- [Blog de noticias](#)
- [Wiki en Español](#)

**Articulos en Español**

Alan Lavintman, German Viscuso  
["db4o: una alternativa a la persistencia"](#)  
.code, February 2007

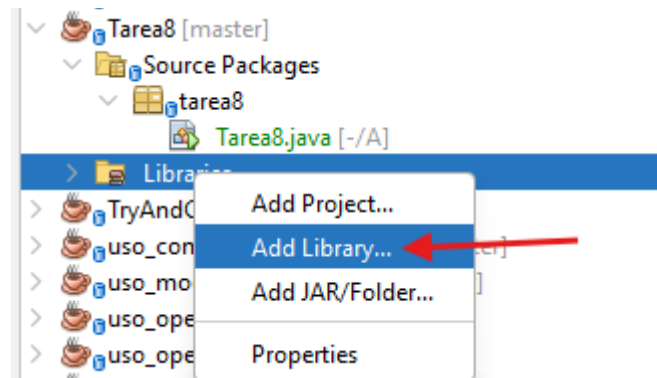
Juan Antonio Palos, Jim Paterson  
["Persistencia de Objetos Java utilizando db4o"](#)  
Programación, 15.4.2005

Una vez descargado **se descomprime** y **accedemos a la carpeta que se llama lib**. En esta se encuentran las **bibliotecas necesarias para poder utilizar este Sistema Gestor Base de Datos**

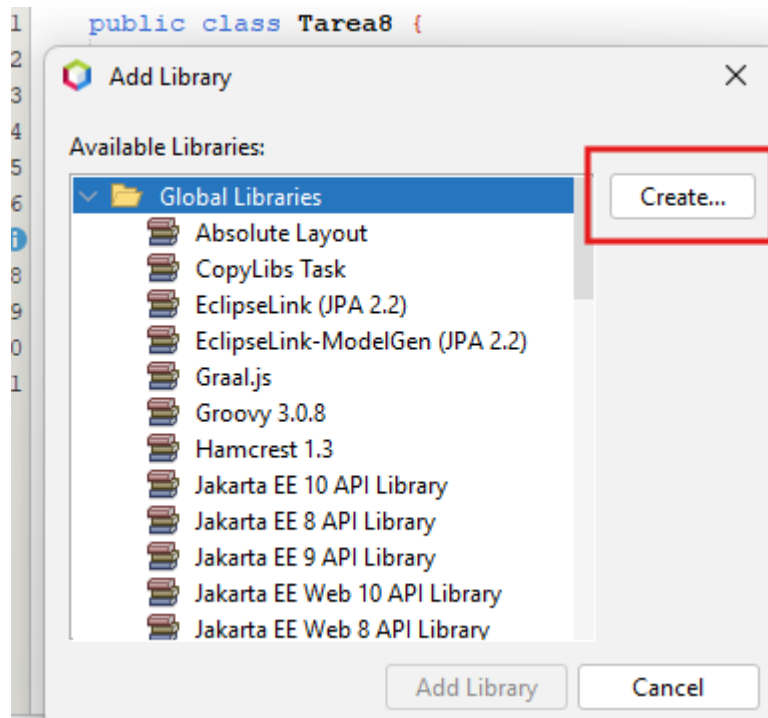
lib				
Documentos > JAVA > db4o-8.0 > lib				
Ordenar Ver				
Nombre	Fecha de modificación	Tipo	Tamaño	
3rdpartylibs.html	22/04/2024 16:58	Chrome HTML Do...	2 KB	
ant.jar	22/04/2024 16:58	Executable Jar File	1.260 KB	
ant.license.html	22/04/2024 16:58	Chrome HTML Do...	4 KB	
bloat.license.html	22/04/2024 16:58	Chrome HTML Do...	22 KB	
bloat-1.0.jar	22/04/2024 16:58	Executable Jar File	682 KB	
db4o-8.0.249.16098-all-java5.jar	22/04/2024 16:58	Executable Jar File	2.581 KB	
db4o-8.0.249.16098-bench.jar	22/04/2024 16:58	Executable Jar File	30 KB	
db4o-8.0.249.16098-core-java5.jar	22/04/2024 16:58	Executable Jar File	1.438 KB	
db4o-8.0.249.16098-cs.optional-java5.jar	22/04/2024 16:58	Executable Jar File	18 KB	
db4o-8.0.249.16098-cs-java5.jar	22/04/2024 16:58	Executable Jar File	203 KB	
db4o-8.0.249.16098-db4ounit-java5.jar	22/04/2024 16:58	Executable Jar File	242 KB	
db4o-8.0.249.16098-instrumentation-java...	22/04/2024 16:58	Executable Jar File	59 KB	
db4o-8.0.249.16098-nqopt-java5.jar	22/04/2024 16:58	Executable Jar File	70 KB	
db4o-8.0.249.16098-optional-java5.jar	22/04/2024 16:58	Executable Jar File	120 KB	
db4o-8.0.249.16098-osgi-java5.jar	22/04/2024 16:58	Executable Jar File	2.285 KB	
db4o-8.0.249.16098-osgi-test-java5.jar	22/04/2024 16:58	Executable Jar File	2.995 KB	
db4o-8.0.249.16098-taj-java5.jar	22/04/2024 16:58	Executable Jar File	23 KB	
db4o-8.0.249.16098-tools-java5.jar	22/04/2024 16:58	Executable Jar File	4 KB	
readme.html	22/04/2024 16:58	Chrome HTML Do...	6 KB	

✖ **Añadir la biblioteca de Db4o al proyecto java de NetBeans.**

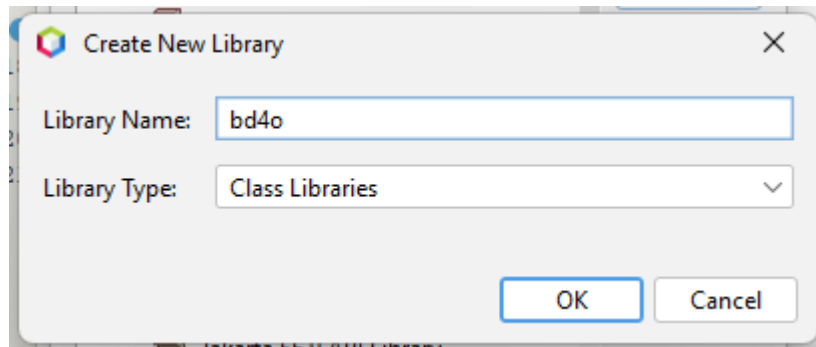
Botón derecho sobre la carpeta Libraries > **Add Library**



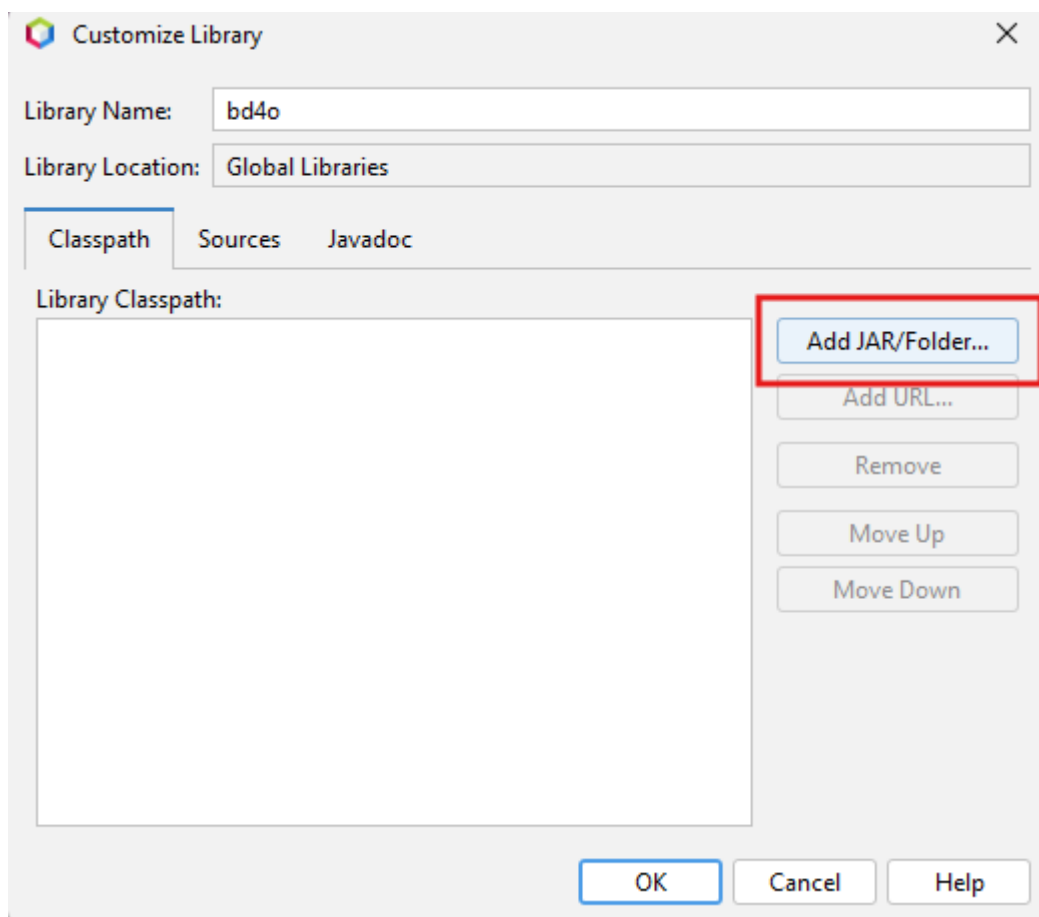
**Pulsamos en Create** para crear una nueva biblioteca



**Asignamos un nombre** a la librería

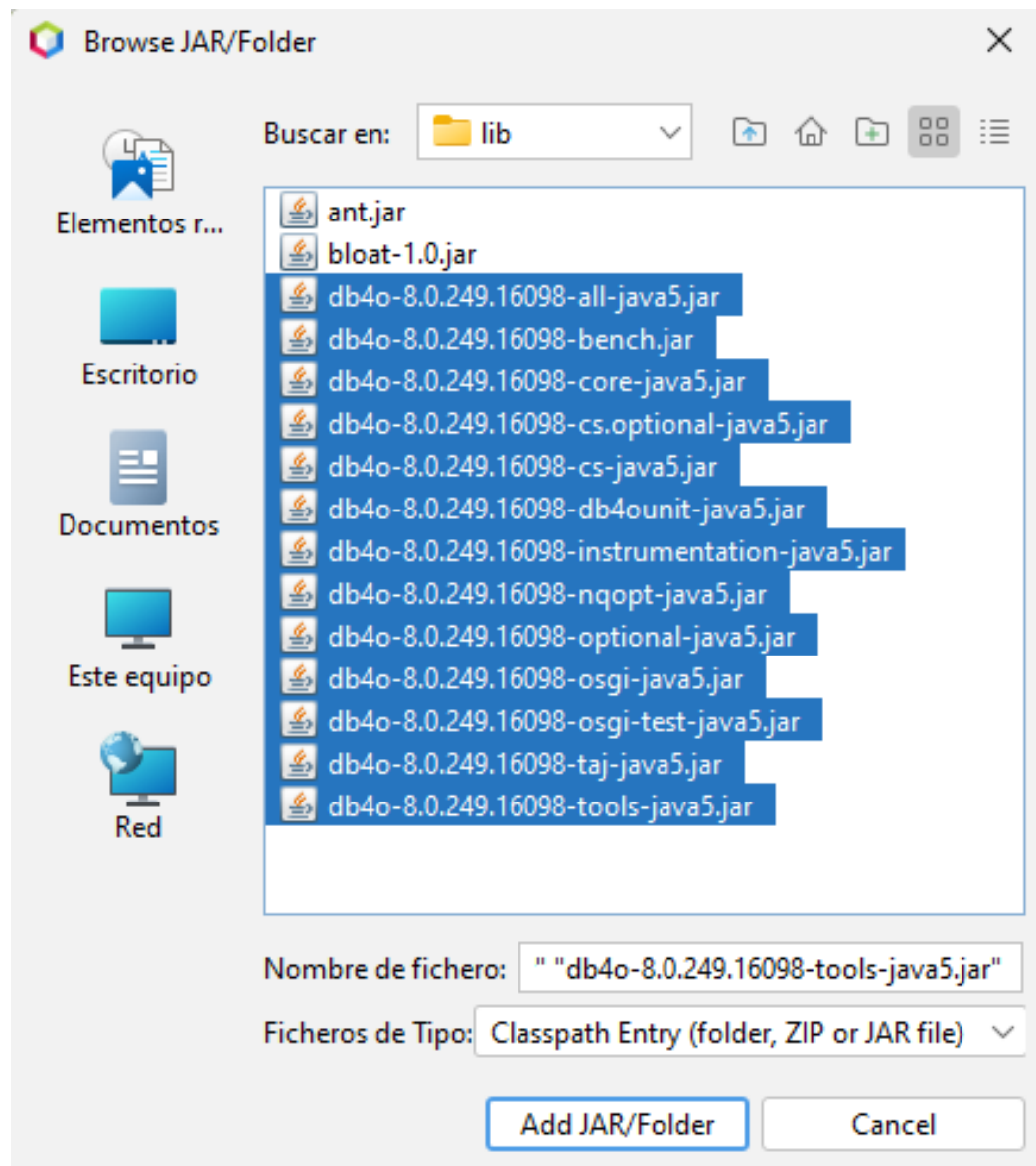


**Add/JAR/Folder** ( Añadir jar/carpeta) para añadir los ficheros a nuestra biblioteca.

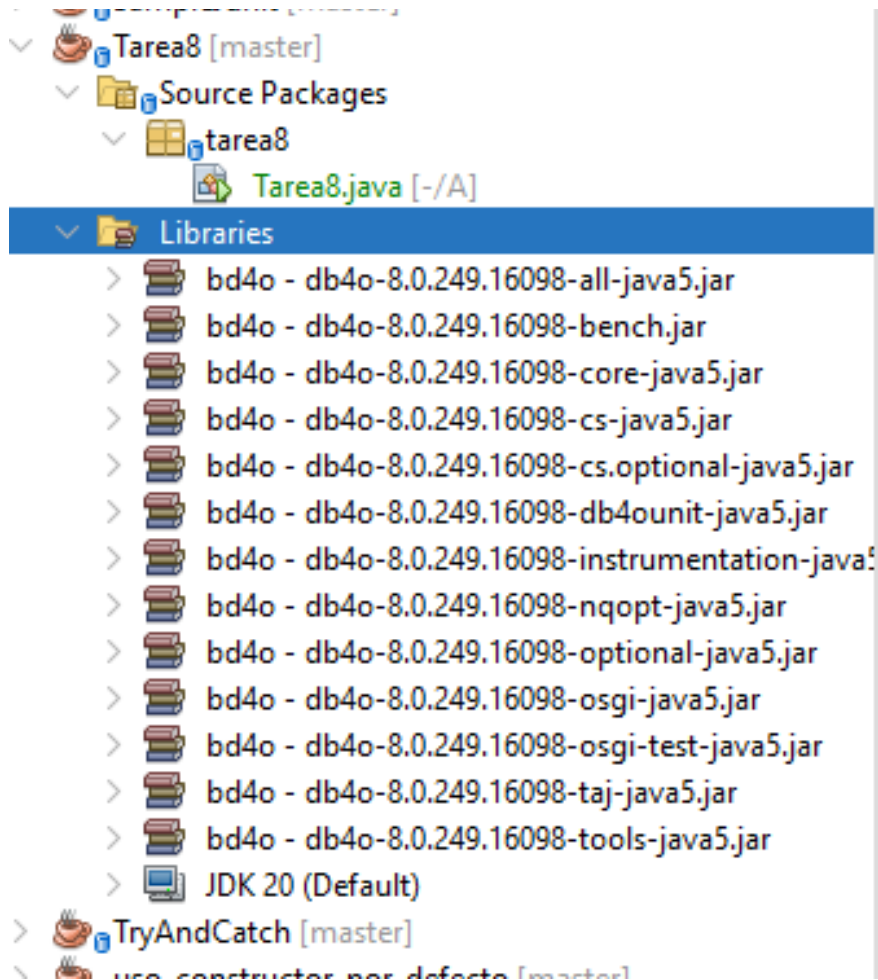




Añadimos los que se muestran a continuación



Finalmente en nuestro proyecto debe aparecer en librerías de la siguiente forma estando ya listo para poder trabajar con la Base de datos BD4o



#### 4. RA08\_e) Se han creado bases de datos y las estructuras necesarias para el almacenamiento de objetos.

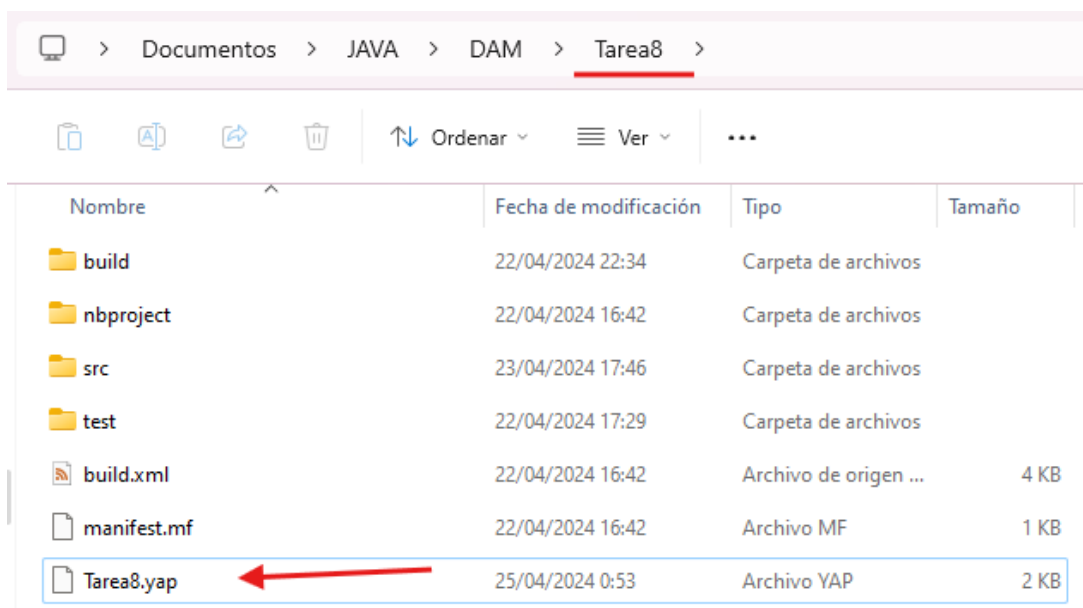
##### ***A) Fichero Tarea8: Se va a trabajar con una BD que almacenaremos en un fichero "Tarea8".***

En la clase que se va a encargar de gestionar la BD se va a crear una constante con la URL donde se va a encontrar el archivo llamado Tarea8.yap

Aparte tendremos un método para conectar a esa base de datos (Se abre el fichero con una configuración vacía y pasándole la ruta)

```
public class GestorBD {  
    private ObjectContainer bd;  
    final private String URL_BD = "Tarea8.yap";  
    /**  
     * Abrir conexion con la BD pasándole una configuración vacía y la ruta  
     */  
    public void conectarBD() {  
        try {  
            bd = Db4oEmbedded.openFile(  
                config: Db4oEmbedded.newConfiguration(), databaseFileName: URL_BD);  
        } catch (Exception e) {  
            ControladorErrores.ErrorshowDialog("Ha habido un error al conectar con la base  
        }  
    }  
}
```

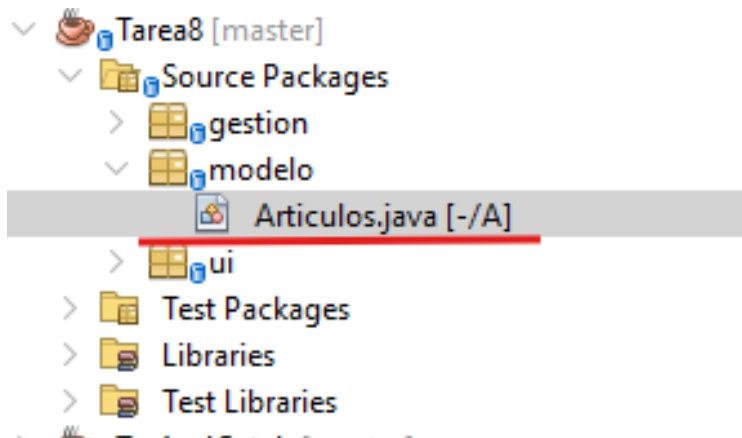
Este archivo se encuentra en la ruta del proyecto ya que es una ruta relativa.



Documentos > JAVA > DAM > Tarea8 >				
Ordenar Ver ...				
Nombre	Fecha de modificación	Tipo	Tamaño	
build	22/04/2024 22:34	Carpeta de archivos		
nbproject	22/04/2024 16:42	Carpeta de archivos		
src	23/04/2024 17:46	Carpeta de archivos		
test	22/04/2024 17:29	Carpeta de archivos		
build.xml	22/04/2024 16:42	Archivo de origen ...	4 KB	
manifest.mf	22/04/2024 16:42	Archivo MF	1 KB	
Tarea8.yap	25/04/2024 0:53	Archivo YAP	2 KB	

5. RA08\_f) Se han programado aplicaciones que almacenen objetos en las bases de datos creadas.

**A) Clase Artículo:** para crear los objetos que almacenaremos en la BD.



**Atributos de la clase:** Código, Nombre, Cantidad, Descripción

```
public class Articulos {  
  
    //ATRIBUTOS DE CLASE  
    private int codigo;  
    private String nombre;  
    private int cantidad;  
    private String descripcion;  
}
```

**Constructores:** Con parámetros y sin parámetros.

```
//CONSTRUCTOR CON PARAMETROS  
public Articulos(int codigo, String nombre, int cantidad, String descripcion) {  
    this.codigo = codigo;  
    this.nombre = nombre;  
    this.cantidad = cantidad;  
    this.descripcion = descripcion;  
}  
  
//CONSTRUCTOR VACÍO  
public Articulos() {  
  
}
```

**Métodos:** *Getters y Setters*

```
//METODO GETTERS Y SETTERS
public int getCodigo() {
    return codigo;
}

public void setCodigo(int codigo) {
    this.codigo = codigo;
}

public String getNombre() {
    return nombre;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

public int getCantidad() {
    return cantidad;
}

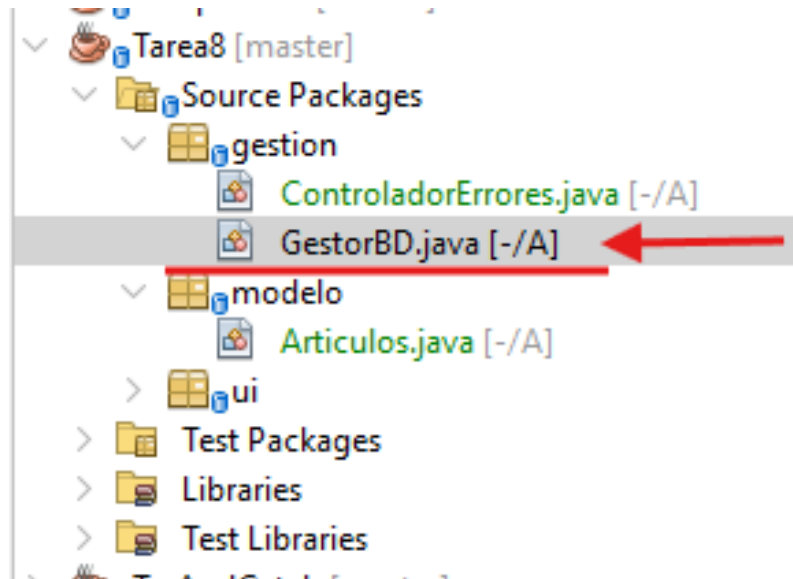
public void setCantidad(int cantidad) {
    this.cantidad = cantidad;
}

public String getDescripcion() {
    return descripcion;
}

public void setDescripcion(String descripcion) {
    this.descripcion = descripcion;
}
```

6. RA08\_g) Se han realizado programas para recuperar, actualizar y eliminar objetos de las bases de datos.

***A) Clase GestorBD: Toda la gestión de la información se hará a través de una clase.***



En esta clase llamada GestorBD se encuentran los distintos métodos para la conexión con la Base de dato (insertar, consultar, actualizar, borrar).

Estos métodos serán llamados desde los distintos eventos de la interfaz.

**Conectar a la base de datos**

```
public void conectarBD() {  
    try {  
        bd = Db4oEmbedded.openFile(  
            config: Db4oEmbedded.newConfiguration(), databaseFileName: URL_BD);  
    } catch (Exception e) {  
        ControladorErrores.ErrorshowDialog("Ha habido un error al conectar con la base de datos\n" + e.getMessage());  
    }  
}
```

## Insertar artículo en la BD

```
public void insertar(Articulos articulo) {  
  
    try {  
        //si existe el codigo de articulo (es true)... indicamos que ya existe  
        if (ComprobarSiExisteArticulo(codigo: articulo.getCodigo())) {  
            ControladorErrores.ErrorshowDialog(msg: " Este codigo de articulo ya existe");  
            return;  
        }  
  
        //Conectamos con la BD  
        conectarBD();  
        //sino existe(es false)... añadimos el articulo  
        bd.store(s: articulo);  
        ControladorErrores.InfoshowDialog(msg: "El articulo se ha insertado correctamente");  
    } catch (Exception e) {  
        ControladorErrores.ErrorshowDialog("Ha habido un error al insertar el artículo en la BD\n" + e.getMessage());  
    } finally {  
        bd.close(); //cerramos conexión con la BD  
    }  
}
```

## Buscar artículo en la BD

```
/**  
 * Busca el artículo en la BD por su código.  
 *  
 * @param codigo: código del artículo.  
 * @return Articulos: objeto Articulos que contiene la descripción o  
 * null si no se encuentra.  
 */  
public Articulos buscar(int codigo) {  
    //Establecemos conexión a la BD  
    conectarBD();  
  
    try {  
        //llamamos al ObjectSet(colección 'parecido a un ArrayList')  
        //con nombre resultBuscar que recoge el artículo con el código.  
        ObjectSet<Articulos> resultBuscar = bd.queryByExample(new Articulos(codigo, nombre: null, cantidad: 0, descripcion: null));  
  
        //Si no hay artículo con el código retorna null.  
        if (resultBuscar.isEmpty()) {  
            return null;  
        }  
  
        //Si hay artículo con el código hay artículo con el código entonces retorna el contenido.  
    } else {  
        return resultBuscar.next();  
    }  
}  
  
} catch (Exception e) {  
    ControladorErrores.ErrorshowDialog("Ha habido un error al buscar el artículo en la BD\n" + e.getMessage());  
    return null;  
} finally {  
    bd.close();  
}
```

## Borrar artículo en la BD

```
/**
 * Borra un
 *
 * @param c
 */
public void borrar(int codigo) {

    //Establecemos conexión a la BD
    conectarBD();
    try {
        //llamamos al ObjectSet (coleccion 'parecido a un ArrayList')
        //con nombre resultBuscar que recoge el articulo con el codigo.
        ObjectSet<Articulos> resultBuscar = bd.queryByExample(new Articulos(codigo, nombre: null, cantidad: 0, descripcion: null));

        //Si resultBuscar es vacio... cerramos la BD y indicamos que no hay articulos
        if (resultBuscar.isEmpty()) {
            bd.close();
            ControladorErrores.ErrorshowDialog(msg: "No se ha encontrado el articulo referente a ese codigo");
        }

        int pregunta = ControladorErrores.PreguntashowDialog(msg: "¿Estás seguro de que deseas borrar este artículo?", option1: "SI", option2: "NO");

        //Si la pregunta es 0 (significa "SI")... Borramos el articulo de la bd
        if (pregunta == 0) {
            bd.delete(o: resultBuscar.next());
            ControladorErrores.InfoshowDialog(msg: "Se ha borrado correctamente");

            //Si la pregunta no es 0 (significa "NO")... No eliminamos el articulo
        } else {
            ControladorErrores.InfoshowDialog(msg: "El articulo no se ha eliminado");
        }

    } catch (Exception e) {
        ControladorErrores.ErrorshowDialog("Ha habido un error al borrar el articulo en la BD\n" + e.getMessage());
    } finally {
        bd.close(); //cerramos conexión con la BD
    }
}
```



```

/**
 * Actualiza el articulo en la bd
 *
 * @param codigo Código referente para actualizar el articulo
 * @param
 * @param
 * @param
 */

```

**Actualizar el articulo en la BD**

```

public void actualizar(int codigo, String nombre, int cantidad, String descripcion) {
    //Establecemos conexión a la BD
    conectarBD();
    try {
        //llamamos al ObjectSet (coleccion 'parecido a un ArrayList')
        //con nombre resultBuscar que recoge el articulo con el codigo.
        ObjectSet<Articulos> resultBuscar = bd.queryByExample(new Articulos(codigo, nombre: null, cantidad: 0, descripcion: null));

        //Si resultbuscar es vacio... cerramos la bd y indicamos que no hay articulos
        if (resultBuscar.isEmpty()) {
            bd.close();
            ControladorErrores.ErrorshowDialog(msg: "No se ha encontrado el articulo para modificar");
        }

        //En cambio si tuviera articulos con el codigo indicado... creamos un objeto articulo del resultBuscar
        Articulos articulo = resultBuscar.next();
        //si nombre no es nulo... añadimos el nombre a objeto
        if (nombre != null) {
            articulo.setNombre(nombre);
        }

        //si precio es 0... añadimos el precio a objeto
        if (cantidad != 0) {
            articulo.setCantidad(cantidad);
        }

        //si descripcion es nulo... añadimos la descripcion a objeto
        if (descripcion != null) {
            articulo.setDescripcion(descripcion);
        }

        //Actualizamos el articulo, mostramos mensaje de exito y cerramos la BD
        bd.store(o: articulo);
        ControladorErrores.InfoshowDialog(msg: "El articulo se ha modificado correctamente");

    } catch (Exception e) {
        ControladorErrores.ErrorshowDialog("Ha habido un error al actualizar el articulo en la BD\n" + e.getMessage());
    } finally {
        bd.close();//cerramos conexión con la BD
    }
}
}

```

## Mostrar todos los artículos en la BD

```
public String mostrar() {
    //Establecemos conexión a la BD
    conectarBD();
    try {
        String resultado;

        //llamamos al ObjectSet (coleccion 'parecido a un ArrayList') con nombre resultBuscar que recoge el articulo
        ObjectSet<Articulos> resultBuscar = bd.queryByExample(new Articulos());

        //Si resultBuscar es vacio (no hay articulos)
        if (resultBuscar.isEmpty()) {
            //Almacenamos en resultado que no se encontraron articulos
            resultado = "No se han encontrado articulos en la BD";
        } else {
            //si resultBuscar no está vacio... entra al while para recoger cada articulo
            //almacenandolo en la variable resultado y repitiendo el while hasta no obtener más articulos.
            resultado = "Números de articulos: " + resultBuscar.size() + "\n";
            while (resultBuscar.hasNext()) {
                Articulos art = (Articulos) resultBuscar.next();
                resultado = resultado + "\nCódigo: " + art.getCodigo()
                    + "\nNombre: " + art.getNombre()
                    + "\nCantidad: " + art.getCantidad()
                    + "\nDescripción: " + art.getDescripcion() + "\n";
            }
        }
        //retornamos el resultado.
        return resultado;
    } catch (Exception e) {
        ControladorErrores.ErrorshowDialog("Ha habido un error al mostrar los artículos en la BD\n" + e.getMessage());
        return null;
    } finally {
        bd.close(); //cerramos conexión con la BD
    }
}
```

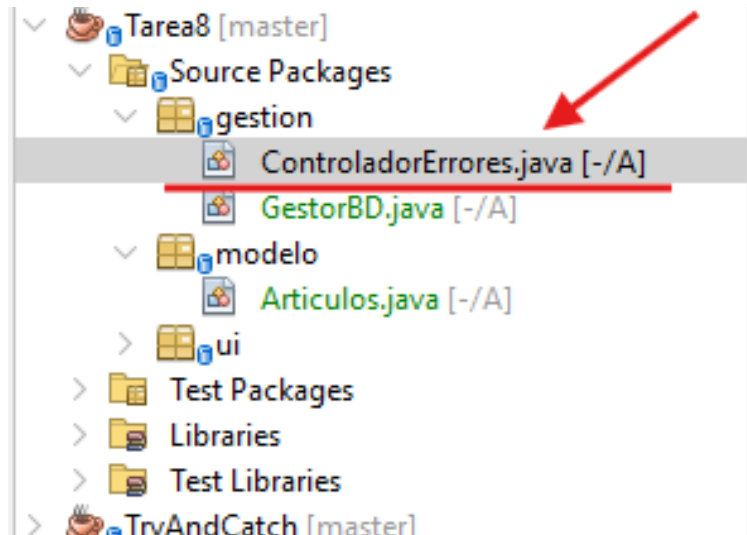
## Método para comprobar si existe el artículo, obteniendo true o false

```
/**
 * Compr
 *
 * @para
 * @retu
 */
private boolean ComprobarSiExisteArticulo(int codigo) {
    conectarBD();
    try {
        //llamamos al ObjectSet (coleccion 'parecido a un ArrayList') con nombre resultBuscar que recoge el codigo del articulo.
        ObjectSet<Articulos> resultBuscar = bd.queryByExample(new Articulos(codigo, nombre: null, cantidad: 0, descripcion: null));

        return !resultBuscar.isEmpty();
    } finally {
        bd.close(); //cerramos conexión con la BD
    }
}
```

## **B) Clase ControladorErrores: Se va a crear una clase para controlar los Errores.**

Esta clase se va a llamar **ControladorErrores**



Va a contener métodos para evitar dejar campos en blancos y posibles errores del programa.

También se van a almacenar los diferentes métodos para mostrar JOptionPane

```
public class ControladorErrores {  
  
    /**  
     * Controla que ningun campo se encuentre vacío  
     */  
    Método para evitar dejar campos Vacíos  
    /**  
     * Control de posibles errores, si cualquier textField está vacío... entra a su if, guarda la cadena de texto en la variable msgError  
     */  
    public static String controlarCamposVacios(String codigo, String nombre, String cantidad, String descripcion) {  
        String msg = null;  
        if (codigo.equals(anObject: "") & nombre.equals(anObject: "") & cantidad.equals(anObject: "") & descripcion.equals(anObject: "")) {  
            msg = "Te has dejado todos los campos en blanco";  
        } else if (codigo.equals(anObject: "")) {  
            msg = "Te has dejado el CODIGO en blanco";  
        } else if (nombre.equals(anObject: "") || cantidad.equals(anObject: "") || descripcion.equals(anObject: "")) {  
            msg = "Te has dejado nombre, cantidad o descripcion en blanco";  
        }  
        return msg;  
    }  
  
    /**  
     * Control de posibles errores, si cualquier textField está vacío... entra a su if, guarda la cadena de texto en la variable msgError  
     */  
    Método para evitar dejar el campo código vacío  
    /**  
     * Control de posibles errores, si cualquier textField está vacío... entra a su if, guarda la cadena de texto en la variable msgError  
     */  
    public static String controlarCampoCodigo(String codigo) {  
        String msg = null;  
        if (codigo.equals(anObject: "")) {  
            msg = "Te has dejado el CODIGO en blanco";  
        }  
        return msg;  
    }  
}
```

### Métodos para enviar ventanas emergentes de error, pregunta o información

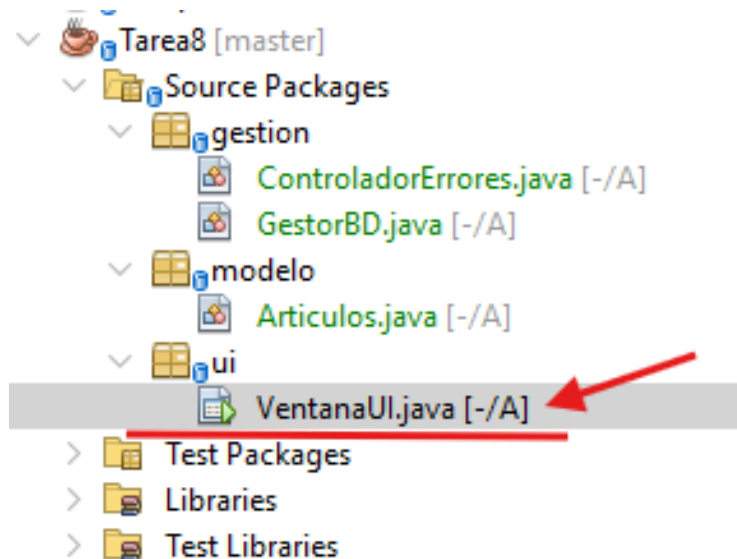
```
/**
 * JOptionPane de error
 * @param msg mensaje a mostrar
 */
public static void ErrorshowDialog(String msg) {
    JOptionPane.showMessageDialog(parentComponent:null, message:msg,
        title: "Error", messageType:JOptionPane.ERROR_MESSAGE);
}

/**
 * JOptionPane para preguntar
 * @param msg mensaje a mostrar
 * @param option1 Si
 * @param option2 NO
 * @return
 */
public static int PreguntashowDialog(String msg, String option1, String option2) {
    int pregunta = JOptionPane.showOptionDialog(parentComponent:null, message:msg,
        title: "Ventana de confirmación", optionType: JOptionPane.YES_NO_CANCEL_OPTION, messageType:JOptionPane.QUESTION_MESSAGE,
        icon: null, new Object[]{option1, option2}, initialValue: "SI");
    return pregunta;
}

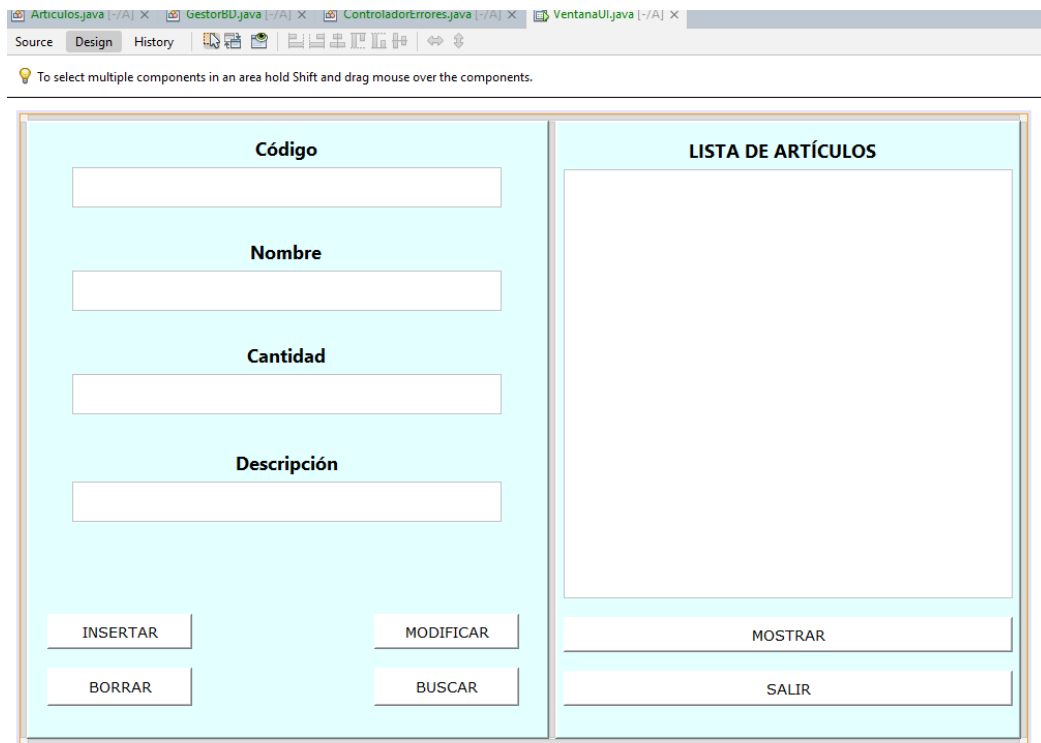
/**
 * JOptionPane de informacion
 * @param msg mensaje a mostrar
 */
public static void InfoshowDialog(String msg) {
    JOptionPane.showMessageDialog(parentComponent:null, message:msg,
        title: "ÉXITO", messageType:JOptionPane.INFORMATION_MESSAGE);
}
```

### C) JFrame form: Parte gráfica para que sean llamado los distintos eventos de la interfaz

JFrame form será nombrado como **VentanaUI**.



Contiene una parte gráfica



Como también la parte de código en la cual se llamará a los métodos de la clase GestorBD para que al hacer interacción con los botones realicen su función.

### Ejemplo de una parte del código

```
private void insertarBtActionPerformed(java.awt.event.ActionEvent evt) {  
    GestorBD bd = new GestorBD();  
  
    String msg = ControladorErrores.controlarCamposVacios(codigo: this.codigoTf.getText(), nombre: this.nombreTf.getText(), cantidad:  
  
    if (msg == null) {  
        //Objeto Articulos que recoge por parámetros los textfield de la ventana  
        Articulos art = new Articulos(codigo: Integer.parseInt(s: this.codigoTf.getText()), nombre: this.nombreTf.getText(), cantidad:  
  
        //Se llama al método insert para insertar el articulo y el mensaje que  
        //retorna se almacena en la variable mensaje  
        bd.insertar(articulo: art);  
  
        //mostramos el mensaje en el label y reiniciamos los campos TextField  
        limpiarCampos();  
    } else {  
        ControladorErrores.ErrorshowDialog(msg);  
    }  
}  
  
private void buscarBtActionPerformed(java.awt.event.ActionEvent evt) {  
    GestorBD bd = new GestorBD();  
    String msg = ControladorErrores.controlarCampoCodigo(codigo: this.codigoTf.getText());  
  
    if (msg == null) {  
        //Se llama al metodo buscar que devuelve un objeto  
        Articulos articuloEncontrado = bd.buscar(codigo: Integer.parseInt(s: this.codigoTf.getText()));  
  
        //Si (articuloEncontrado no es null) entonces rellena los campos restante TextFiel del articulo  
        if (articuloEncontrado != null) {  
            this.mensajeLb.setText(text: "Se ha encontrado el articulo");  
  
            this.nombreTf.setText(s: articuloEncontrado.getNombre());  
            this.cantidadTf.setText(s: Integer.toString(i: articuloEncontrado.getCantidad()));  
            this.descripcionTf.setText(s: articuloEncontrado.getDescripcion());  
        }  
    }  
}
```

7. RA08\_h) Se han realizado programas para almacenar y gestionar tipos de datos estructurados, compuestos y relacionados.

### ***Botón Insertar***

*El objetivo es captar la información tecleada por el usuario de la aplicación y guardarla la base de datos.*

### **Interfaz gráfica**

Al rellenar los TextField se añadirá el artículo en la BD, evitando duplicados y controlando que no se dejen campos vacíos.

La interfaz gráfica está dividida en dos paneles principales:

- Panel izquierdo (Formulario de entrada):**
  - Código:
  - Nombre:
  - Cantidad:
  - Descripción:
  - Botones de acción: INSERTAR, MODIFICAR, BORRAR, BUSCAR.
- Panel derecho (Lista de artículos):**
  - LISTA DE ARTÍCULOS:
  - Botones de acción: MOSTRAR, SALIR.

Una flecha roja apunta al botón INSERTAR.

## Código

### Clase GestorBD

- Si el método ComprobarSiExisteArtículo devuelve true entonces indica que ya existe el artículo.
- Si no existe entonces se conecta a la BD, añade el artículo, lo indica mediante una ventana emergente y por último en el finally cierra la conexión con la BD

```
/**
 * Inserta un artículo en la BD
 *
 * @param articulo
 */
public void insertar(Articulos articulo) {
    try {
        //si existe el código de artículo (es true)... indicamos que ya existe
        if (ComprobarSiExisteArtículo(código: articulo.getCódigo())) {
            ControladorErrores.ErrorshowDialog(msg: "Este código de artículo ya existe");
            return;
        }

        //Conectamos con la BD
        conectarBD();
        //sino existe (es false)... añadimos el artículo
        bd.store(o: articulo);
        ControladorErrores.InfoshowDialog(msg: "El artículo se ha insertado correctamente");
    } catch (Exception e) {
        ControladorErrores.ErrorshowDialog("Ha habido un error al insertar el artículo en la BD\n" + e.getMessage());
    } finally {
        bd.close(); //cerramos conexión con la BD
    }
}
```

## Jform VentanaUI

- Creamos un objeto de la clase GestorBD para llamar a los métodos
- Almacenamos en un String msg el resultado que obtenemos con el método ControlarCamposVacios de la clase controlarErrores (al ser static se llama directamente a la clase sin tener que crear un objeto de esta)
- Si msg es null significa que no hay campos vacíos entonces creamos un objeto de la clase articulo (indicándole por parámetros los TextField)
- Posteriormente insertamos el articulo en la base de datos con el método insertar de la clase GestorBD y limpiamos los campos TextField
- En caso de que obtuviéramos algún mensaje mediante el método controlarCamposVacios este enviará ese mensaje mediante una ventana emergente y no almacenaría el articulo en el fichero

```
private void insertarBtActionPerformed(java.awt.event.ActionEvent evt) {  
    GestorBD bd = new GestorBD();  
  
    String msg = ControladorErrores.controlarCamposVacios(codigo: this.codigoTf.getText(), nombre: this.nombreTf.getText(), c.  
  
    if (msg == null) {  
        //Objeto Articulos que recoge por parámetros los textField de la ventana  
        Articulos art = new Articulos(codigo: Integer.parseInt(s: this.codigoTf.getText()), nombre: this.nombreTf.getText(), c.  
  
        //Se llama al método insert para insertar el articulo y el mensaje que  
        //retorna se almacena en la variable mensaje  
        bd.insertar(articulo: art);  
  
        //mostramos el mensaje en el label y reiniciamos los campos TextField  
        limpiarCampos();  
    } else {  
        ControladorErrores.ErrorshowDialog(msg);  
    }  
}
```



## Botón Buscar

*El usuario escribirá un código en el campo correspondiente, al hacer click en buscar, el programa buscará en la BD el producto con ese código.*


*Si no existe dará un mensaje de error indicándolo.*

*Si existe, rellenará el resto de campos con los datos que corresponden al producto con el código tecleado.*

## Interfaz gráfica

Al rellenar el TextFiel código este buscará el articulo en la BD, evitando que no se deje el campo código en blanco.

Código	LISTA DE ARTÍCULOS
<input type="text"/>	
Nombre	
<input type="text"/>	
Cantidad	
<input type="text"/>	
Descripción	
<input type="text"/>	
<input type="button" value="INSERTAR"/>	<input type="button" value="MOSTRAR"/>
<input type="button" value="BORRAR"/>	<input type="button" value="SALIR"/>
<input type="button" value="MODIFICAR"/>	
<input type="button" value="BUSCAR"/>	



## Código

### Clase GestorBD

- Establecemos conexión con la BD
- Creamos un objeto de la colección ObjectSet que va a recoger el artículo con el código indicado.
- Este comprobará si existe o no
- Si no existe entonces se va a retornar null y si existe se va a retornar el artículo.
- Por último en el finally cerramos la bd .

```
/**
 * Busca un artículo a partir del código introducido
 *
 * @param codigo
 * @return el artículo rellenando los campos nombre, cantidad, descripción o
 * null si no se encuentra
 */
public Articulos buscar(int codigo) {
    //Establecemos conexión a la BD
    conectarBD();

    try {
        //llamamos al ObjectSet(colección 'parecido a un ArrayList')
        //con nombre resultBuscar que recoge el artículo con el código.
        ObjectSet<Articulos> resultBuscar = bd.queryByExample(new Articulos(codigo, nombre: null, cantidad: 0, descripcion: null));

        //Si no hay artículo con el código retorna null.
        if (resultBuscar.isEmpty()) {
            return null;

            //Si hay artículo con el código hay artículo con el código entonces retorna el contenido.
        } else {
            return resultBuscar.next();
        }
    } catch (Exception e) {
        ControladorErrores.ErrorshowDialog("Ha habido un error al buscar el artículo en la BD\n" + e.getMessage());
        return null;
    } finally {

        bd.close();
    }
}
```

## Jform VentanaUI

- Creamos un objeto de la clase GestorBD para llamar a los métodos
- Almacenamos en un String msg el resultado que obtenemos con el método ControlarCampoCodigo de la clase controlarErrores (al ser static se llama directamente a la clase sin tener que crear un objeto de esta)
- Si msg es null significa que el campo código no está vacío entonces buscamos el artículo (indicándole por parámetros el código introducido en el TextField)
- Si el resultado de la búsqueda que obtengamos es diferente a null significa que si ha encontrado un artículo entonces rellenamos los campos restantes (nombre, cantidad, descripción) en los TextField.
- Si se diera el caso de obtener como resultado de la búsqueda null (no hay artículo) enviaríamos un mensaje indicándolo
- 
- En caso de que obtuviéramos un mensaje mediante el método controlarCampoCodigo este enviará ese mensaje en una ventana emergente indicando que está vacío el TextField código.

```
private void buscarBtActionPerformed(java.awt.event.ActionEvent evt) {  
  
    GestorBD bd = new GestorBD();  
    String msg = ControladorErrores.controlarCampoCodigo(codigo: this.codigoTf.getText());  
  
    if (msg == null) {  
        //Se llama al metodo buscar que devuelve un objeto  
        Articulos articuloEncontrado = bd.buscar(codigo: Integer.parseInt(s: this.codigoTf.getText()));  
  
        //Si (articuloEncontrado no es null) entonces rellena los campos restante TextFiel del articulo  
        if (articuloEncontrado != null) {  
            this.mensajeLb.setText(text: "Se ha encontrado el artículo");  
  
            this.nombreTf.setText(t: articuloEncontrado.getNombre());  
            this.cantidadTf.setText(s: Integer.toString(i: articuloEncontrado.getCantidad()));  
            this.descripcionTf.setText(t: articuloEncontrado.getDescripcion());  
  
            //Si (articuloEncontrado es null) entonces reinicia los campos TextField y envia un mensaje de error  
        } else {  
            limpiarCampos();  
            ControladorErrores.ErrorshowDialog(msg: "Ese código de artículo no está resgistrado");  
        }  
    } else {  
        ControladorErrores.ErrorshowDialog(msg);  
        limpiarCampos();  
    }  
}
```

## Botón Borrar

*El usuario escribirá un código en el campo correspondiente, al hacer click si existe ese producto, el programa preguntará si está seguro de eliminar ese producto si responde que sí se eliminará definitivamente..*

## Interfaz gráfica

Al rellenar el TextField código este borrará el artículo en la BD, evitando que no se deje el campo código en blanco y preguntando antes de eliminarlo si realmente desea borrar el artículo.

Código	LISTA DE ARTÍCULOS
<input type="text"/>	
Nombre	
<input type="text"/>	
Cantidad	
<input type="text"/>	
Descripción	
<input type="text"/>	
<input type="button" value="INSERTAR"/>	<input type="button" value="MOSTRAR"/>
<input type="button" value="BORRAR"/>	<input type="button" value="SALIR"/>
<input type="button" value="MODIFICAR"/>	
<input type="button" value="BUSCAR"/>	



## Código

### Clase GestorBD

- Establecemos conexión con la BD
- Creamos un objeto de la colección ObjectSet que va a recoger el artículo con el código indicado.
- Este comprobará si existe o no
- Si no existe entonces mandará una ventana emergente indicándolo.
- En cambio si existe lanzará una pregunta si realmente desea eliminarlo.
- Si la respuesta a la pregunta es SI entonces eliminará el artículo e indicándolo que se ha borrado.
- Si la respuesta a la pregunta es NO entonces no eliminará el artículo y se indicará mediante un mensaje.
- Por último en el finally cerramos la bd .

```
/**
 * Borra un artículo a partir del código introducido
 *
 * @param codigo
 */
public void borrar(int codigo) {

    //Establecemos conexión a la BD
    conectarBD();
    try {
        //llamamos al ObjectSet (colección 'parecido a un ArrayList')
        //con nombre resultBuscar que recoge el artículo con el código.
        ObjectSet<Articulos> resultBuscar = bd.queryByExample(new Articulos(codigo, nombre: null, cantidad: 0, descripcion: null));

        //Si resultBuscar es vacío... cerramos la BD y indicamos que no hay artículos
        if (resultBuscar.isEmpty()) {
            ControladorErrores.ErrorshowDialog(msg: "No se ha encontrado el artículo referente a ese código");
        } else {

            int pregunta = ControladorErrores.PreguntashowDialog(msg: "¿Estás seguro de que deseas borrar este artículo?", optional: "SI", option2: "NO");

            //Si la pregunta es 0 (significa "SI")... Borramos el artículo de la bd
            if (pregunta == 0) {
                bd.delete(o: resultBuscar.next());
                ControladorErrores.InfoshowDialog(msg: "Se ha borrado correctamente");

                //Si la pregunta no es 0 (significa "NO")... No eliminamos el artículo
            } else {

                ControladorErrores.InfoshowDialog(msg: "El artículo no se ha eliminado");
            }
        }
    }
} catch (Exception e) {
    ControladorErrores.ErrorshowDialog("Ha habido un error al borrar el artículo en la BD\n" + e.getMessage());
} finally {
    bd.close(); //cerramos conexión con la BD
}
}
```

## Jform VentanaUI

- Creamos un objeto de la clase GestorBD para llamar a los métodos
- Almacenamos en un String msg el resultado que obtenemos con el método ControlarCampoCodigo de la clase controlarErrores (al ser static se llama directamente a la clase sin tener que crear un objeto de esta)
- Si msg es null significa que el campo código no está vacío entonces llamamos al método borrar indicándole por parámetros el código introducido en el TextField.
- Ese método hace la función de la búsqueda del articulo mediante el código, si se encuentra pregunta si desea borrarlo y realiza su eliminación.
- Posteriormente limpiamos los campos TextField y borramos también el articulo en la lista del TextArea mediante el método mostrar(realmente lo que hace es sobrescribir esta lista con los nuevos cambios)
- En caso de que obtuviéramos un mensaje mediante el método controlarCampoCodigo este enviará ese mensaje en una ventana emergente indicando que está vacío el TextField código.

```
private void borrarBtActionPerformed(java.awt.event.ActionEvent evt) {  
  
    GestorBD bd = new GestorBD();  
  
    String msg = ControladorErrores.controlarCampoCodigo(codigo: this.codigoTf.getText());  
  
    if (msg == null) {  
        //llamada al metodo borrar pasandole por parámetros el codigo del TextField  
        bd.borrar(codigo: Integer.parseInt(=: this.codigoTf.getText()));  
  
        //reiniciamos los campos TextField  
        limpiarCampos();  
  
        //Borra tambien el articulo en la lista del textArea generada con mostrar  
        this.mensajeTa.setText(=: bd.mostrar());  
    } else {  
        ControladorErrores.ErrorshowDialog(msg);  
        limpiarCampos();  
    }  
  
}
```


## Botón Modificar

*El usuario escribirá un código en el campo correspondiente, a modificar (código), en el resto de campos se pondrán los nuevos valores. Al pulsar sobre modificar si existe ese producto se modificarán sus datos.*

## Interfaz gráfica

Al rellenar el TextField código este modificará el artículo en la BD, evitando que no se deje el campo código en blanco.

Código	LISTA DE ARTÍCULOS
<input type="text"/>	
Nombre	
<input type="text"/>	
Cantidad	
<input type="text"/>	
Descripción	
<input type="text"/>	
<input type="button" value="INSERTAR"/>	<input type="button" value="MOSTRAR"/>
<input type="button" value="BORRAR"/>	<input type="button" value="SALIR"/>
<input type="button" value="MODIFICAR"/>	
<input type="button" value="BUSCAR"/>	



## Código

### Clase GestorBD

- Establecemos conexión con la BD
- Creamos un objeto de la colección ObjectSet que va a recoger el artículo con el código indicado.
- Este comprobará si existe o no
- Si no existe entonces mandará una ventana emergente indicándolo.
- En cambio si existe se crea un objeto artículo del resultado de la búsqueda y realizará la siguiente operación:
- Si el nombre, la cantidad y la descripción pasada por parámetros no es null/0 le asignamos ese nuevo cambio mediante los métodos setter de la clase artículo y añadimos el artículo a la BD con los cambios realizados
- Por último en el finally cerramos la bd .

```
* Actualiza el artículo en la bd
*
* @param codigo Código referente para actualizar el artículo
* @param nombre Nombre a modificar
* @param cantidad cantidad a modificar
* @param descripcion descripcion a modificar
*/
public void actualizar(int codigo, String nombre, int cantidad, String descripcion) {
    //Establecemos conexión a la BD
    conectarBD();
    try {
        //llamamos al ObjectSet (colección 'parecido a un ArrayList')
        //con nombre resultBuscar que recoge el artículo con el código.
        ObjectSet<Articulos> resultBuscar = bd.queryByExample(new Articulos(codigo, nombre: null, cantidad: 0, descripcion: null));

        //Si resultbuscar es vacío... cerramos la bd y indicamos que no hay artículos
        if (resultBuscar.isEmpty()) {
            ControladorErrores.ErrorshowDialog(msg: "No se ha encontrado el artículo para modificar");
        }

        //En cambio si tuviera artículos con el código indicado... creamos un objeto artículo del resultBuscar
        Articulos articulo = resultBuscar.next();
        //si nombre no es nulo... añadimos el nombre a objeto
        if (nombre != null) {
            articulo.setNombre(nombre);
        }

        //si precio es 0... añadimos el precio a objeto
        if (cantidad != 0) {
            articulo.setCantidad(cantidad);
        }

        //si descripcion es nulo... añadimos la descripcion a objeto
        if (descripcion != null) {
            articulo.setDescripcion(descripcion);
        }

        //Actualizamos el artículo, mostramos mensaje de éxito y cerramos la BD
        bd.store(o: articulo);
        ControladorErrores.Info showDialog(msg: "El artículo se ha modificado correctamente");
    } catch (Exception e) {
        ControladorErrores.ErrorshowDialog("Ha habido un error al actualizar el artículo en la BD\n" + e.getMessage());
    } finally {
        bd.close(); //cerramos conexión con la BD
    }
}
```



## Jform VentanaUI

- Creamos un objeto de la clase GestorBD para llamar a los métodos
- Almacenamos en un String msg el resultado que obtenemos con el método ControlarCamposVacios de la clase controlarErrores (al ser static se llama directamente a la clase sin tener que crear un objeto de esta)
- Si msg es null significa que no hay campos vacíos entonces creamos un objeto de la clase articulo (indicándole por parámetros los TextField)
- Llamamos al método actualizar indicándole por parámetros los TextField, para que busque el articulo y si es encontrado mediante el código entonces realizar una actualización de este.
- Se ha implementado también la actualización de la lista que se va a mostrar en el TextArea mediante el método mostrar de la clase GestorBD (sobrescribe la lista actualizando la modificación del artículo)
- En caso de que obtuviéramos algún mensaje mediante el método controlarCamposVacios este enviará ese mensaje mediante una ventana emergente y no podrá actualizar el articulo indicado.

```
private void modificarBtActionPerformed(java.awt.event.ActionEvent evt) {  
    GestorBD bd = new GestorBD();  
  
    String msg = ControladorErrores.controlarCamposVacios(codigo: this.codigoTf.getText(), nombre: this.nombreTf.getText());  
  
    if (msg == null) {  
        int codigo = Integer.parseInt(this.codigoTf.getText());  
  
        //llamamos al metodo actualizar pasandole por parámetros los TextField  
        bd.actualizar(codigo, nombre: this.nombreTf.getText(), cantidad: Integer.parseInt(this.cantidadTf.getText()));  
  
        //Modifica tambien el articulo en la lista del textArea generada con mostrar  
        //String mensajeMostrar = bd.mostrar();  
        this.mensajeTa.setText(bd.mostrar());  
    } else {  
        ControladorErrores.ErrorshowDialog(msg);  
    }  
}
```

## Botón Mostrar

*Muestra dentro del cuadro de la derecha una lista con todos los datos registros en el fichero.*

## Interfaz gráfica

Al pulsar sobre este botón la función que va a desempeñar es mostrar en la lista de artículos (TextArea) todos los artículos que contiene el fichero.

**Código**

**Nombre**

**Cantidad**

**Descripción**

INSERTAR MODIFICAR

BORRAR BUSCAR

**LISTA DE ARTÍCULOS**

Números de artículos: 4

Código: 1  
Nombre: Tornillo  
Cantidad: 700  
Descripción: Tornillo pequeño

Código: 2  
Nombre: Tuerca  
Cantidad: 69  
Descripción: Tuerca para tornillo pequeño

Código: 3  
Nombre: Clavo  
Cantidad: 100  
Descripción: Clavo de cabeza plana

Código: 4  
Nombre: Alambre  
Cantidad: 300  
Descripción: Rollo de alambre

MOSTRAR

SALIR

## Código

### Clase GestorBD

- Establecemos conexión con la BD
- Creamos un objeto de la colección ObjectSet que va a recoger los artículos.
- Este comprobará si existen artículos o no
- Si no existe almacenará en la variable resultado que no se han encontrado artículos
- En cambio si existe almacenará en la variable resultado que se han encontrado (números de artículos) y entrará en el bucle while recorriendo cada artículo y recogiendo su código, nombre, cantidad y descripción.
- Todo esto será almacenado en resultado mediante una concatenación y es lo que va a devolver este método al ser llamado
- Por último en el finally cerramos la bd .

```
/**
 * Muestra todos los artículos de la BD
 *
 * @return una cadena de texto con toda la información
 */
public String mostrar() {
    //Establecemos conexión a la BD
    conectarBD();
    try {
        String resultado;

        //llamamos al ObjectSet (colección 'parecido a un ArrayList') con nombre resultBuscar que recoge el artículo
        ObjectSet<Articulos> resultBuscar = bd.queryByExample(new Articulos());

        //Si resultBuscar es vacío (no hay artículos)
        if (resultBuscar.isEmpty()) {
            //Almacenamos en resultado que no se encontraron artículos
            resultado = "No se han encontrado artículos en la BD";
        } else {
            //si resultBuscar no está vacío... entra al while para recoger cada artículo
            //almacenándolo en la variable resultado y repitiendo el while hasta no obtener más artículos.
            resultado = "Números de artículos: " + resultBuscar.size() + "\n";
            while (resultBuscar.hasNext()) {
                Articulos art = (Articulos) resultBuscar.next();
                resultado = resultado + "\nCódigo: " + art.getCodigo()
                    + "\nNombre: " + art.getNombre()
                    + "\nCantidad: " + art.getCantidad()
                    + "\nDescripción: " + art.getDescripcion() + "\n";
            }
        }
        //retornamos el resultado.
        return resultado;
    } catch (Exception e) {
        ControladorErrores.ErrorshowDialog("Ha habido un error al mostrar los artículos en la BD\n" + e.getMessage());
        return null;
    } finally {
        bd.close(); //cerramos conexión con la BD
    }
}
```

### Jform VentanaUI

- Creamos un objeto de la clase GestorBD para llamar al método.
- Llamamos al método mostrar.
- Este método va enviar un String que va a ser almacenado en la variable mensaje
- Por último la variable mensaje se va a mostrar en el TextArea mediante el método setText (para sobrescribir el contenido)

```
}
```

```
private void mostrarBtActionPerformed(java.awt.event.ActionEvent evt) {  
    GestorBD db = new GestorBD();  
    //llamamos al metodo mostrar y se almacena en mensaje  
    String mensaje = db.mostrar();  
  
    //método .append --> añade al textarea  
    //MÉTODO .setText --> sobrescribe  
    this.mensajeTa.setText(mensaje);  
}
```

## Botón Salir

*Cierra la aplicación.*

## Interfaz gráfica

Botón para cerrar la aplicación

Diagrama de la interfaz gráfica de usuario (GUI) de una aplicación. La interfaz está dividida en dos paneles. El panel izquierdo contiene cuatro campos de entrada etiquetados como 'Código', 'Nombre', 'Cantidad' y 'Descripción'. Debajo de estos campos hay cuatro botones: 'INSERTAR', 'MODIFICAR', 'BORRAR' y 'BUSCAR'. El panel derecho está titulado 'LISTA DE ARTÍCULOS' y contiene un área vacía para mostrar la lista. En la parte inferior del panel derecho hay dos botones: 'MOSTRAR' y 'SALIR'. Una flecha roja apunta al botón 'SALIR'.

## Jform VentanaUI

- Simplemente indicamos mediante `System.exit(0)` para salir.

```
}
```

```
private void salirBtActionPerformed(java.awt.event.ActionEvent evt) {  
    System.exit(status: 0);  
}
```