

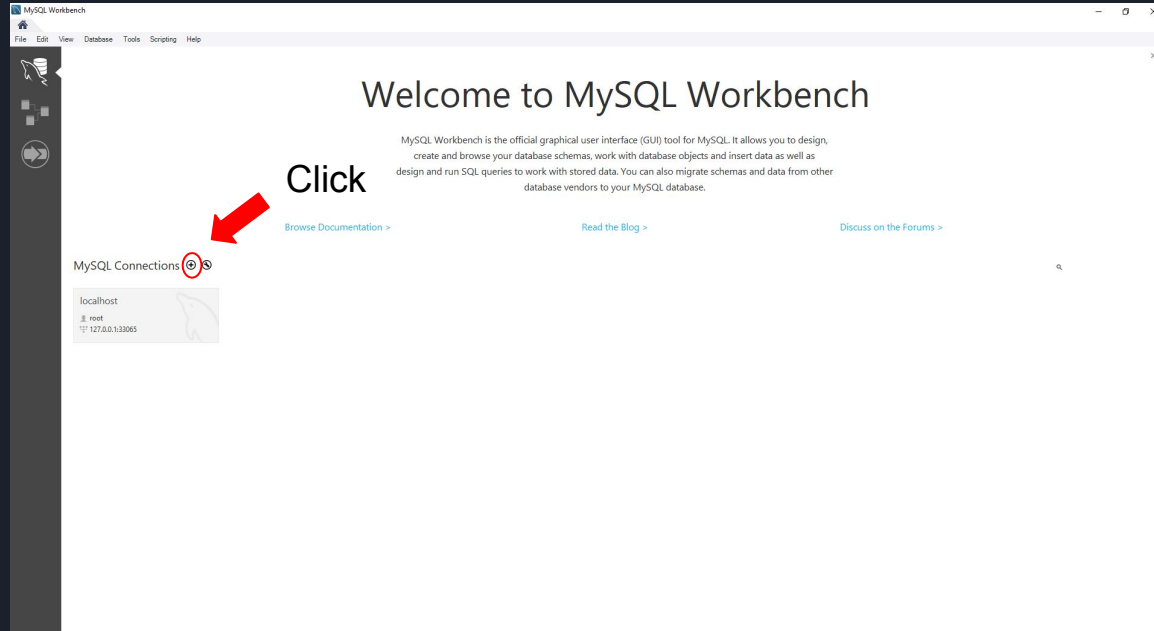


# MySQL Con Java

Alan Jesús Sánchez Catzín

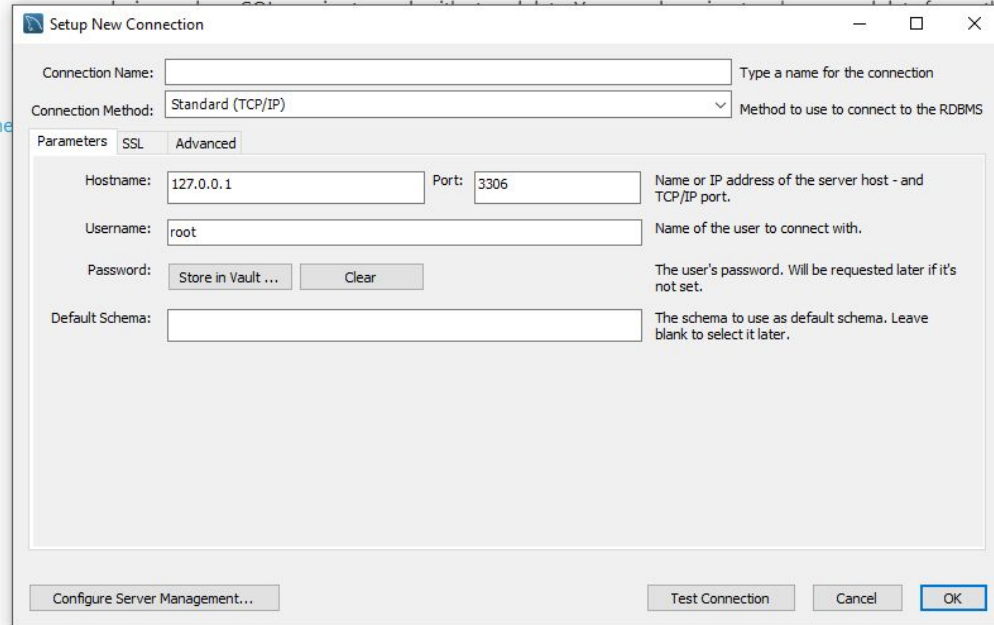
# Crear una base de datos

- Para crear la base de datos se necesita primero haber instalado el MySQL Workbench
- Ahora se necesita crear una conexión a MySQL esto se muestra en las siguientes imágenes



# Crear una base de datos

create and browse your database schemas, work with database objects and insert data as well as



**Setup New Connection**

Connection Name:  Type a name for the connection

Connection Method:  Method to use to connect to the RDBMS

Parameters SSL Advanced

Hostname:  Port:  Name or IP address of the server host - and TCP/IP port.

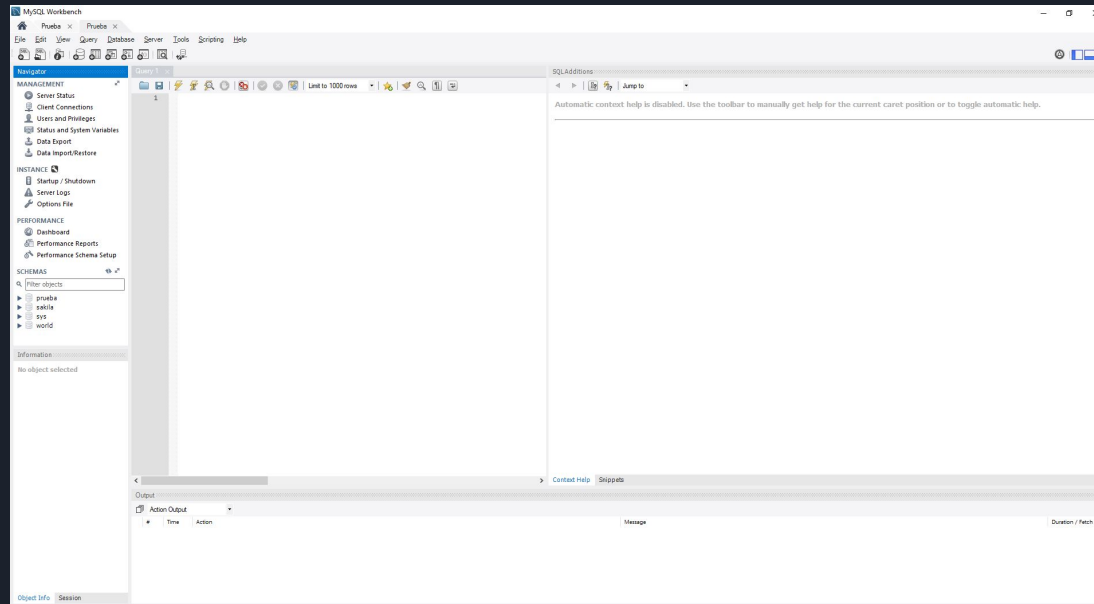
Username:  Name of the user to connect with.

Password:    The user's password. Will be requested later if it's not set.

Default Schema:  The schema to use as default schema. Leave blank to select it later.

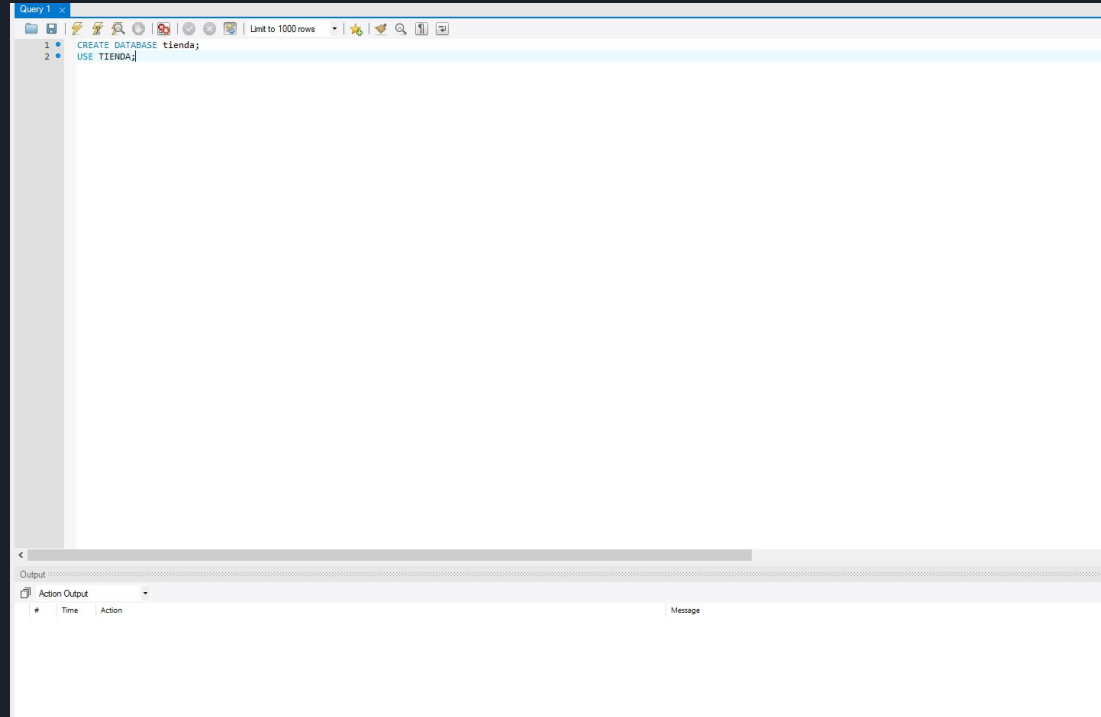
# Crear una base de datos

- Se creará el nombre de la conexión y se seleccionará un puerto, todo lo demás se dejará por default.
- Se dará doble click a la conexión que se nos creó y ahí se podrá acceder a todas los esquemas o bases de datos que tengamos inicializados en esa conexión



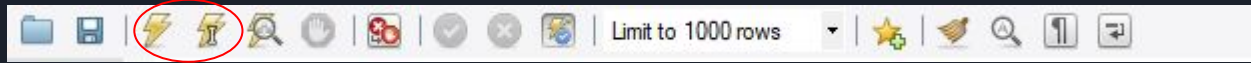
# Crear una base de datos

- Para crear una base de datos se hace uso de la sentencia “CREATE DATABASE nombre” y para poder usarla se necesita usar la sentencia “USE nombre”



# Crear una base de datos

- Para ejecutar cualquier código SQL que se escriba haremos click en los botones que se muestran en la siguiente imagen



- Esto nos mostrará la siguiente respuesta al ejecutarlo

Output				
#	Time	Action	Message	Duration / Fetch
✓ 1	16:06:32	CREATE DATABASE tienda	1 row(s) affected	0.016 sec
✓ 2	16:07:19	USE TIENDA	0 row(s) affected	0.000 sec

- Esto nos habrá creado la base de datos y ya se podrá crear tablas para almacenar datos



# Crear Tablas

- Una tabla es utilizada para organizar y presentar información. Las tablas se componen de filas y columnas de celdas que se pueden rellenar de distintos tipos de información como son las fechas, booleanos, textos, enteros, flotantes, etc.
- Para crear la tabla se usa la sentencia “CREATE TABLE nombre”
- Se escriben los nombres de los campos que va a tener la tabla luego de eso se escriben distintas características como por ejemplo si puede ser null o autoincrementable
- Toda tabla debería de tener una llave primaria
- Cualquier tabla puede tener de 0 a muchas llaves foráneas
- Las llaves primarias son campos con valores únicos que permiten identificar cada registro de dato de la tabla creada
- Las llaves foráneas sirven para obtener datos de otra tabla a través de un JOIN que funciona como unión

En la siguiente diapositiva un ejemplo de una tabla creada con lenguaje SQL

# Crear una tabla

En la siguiente imagen se muestra como se crea una tabla a través de código SQL

```
3 • CREATE TABLE category(  
4     idCat INT(10) NOT NULL AUTO_INCREMENT,  
5     nameCat VARCHAR(10) NOT NULL,  
6     PRIMARY KEY (idCat)  
7 );  
8  
9 • CREATE TABLE items (  
10     idItem INT(10) NOT NULL AUTO_INCREMENT,  
11     name VARCHAR(10) NOT NULL,  
12     price FLOAT NOT NULL,  
13     id_cat INT(10) NOT NULL,  
14     PRIMARY KEY (idItem),  
15     FOREIGN KEY (id_cat) REFERENCES category(idCat)  
16 );  
17  
18 • CREATE TABLE sales(  
19     idItem INT(10) NOT NULL,  
20     id_cat INT(10) NOT NULL,  
21     quantity INT NOT NULL,  
22     FOREIGN KEY (idItem) REFERENCES items(idItem),  
23     FOREIGN KEY (id_cat) REFERENCES category(idCat)  
24 );
```





# Insertar Datos

Para insertar datos usando lenguaje SQL se hace uso de la sentencia "INSERT INTO" y los valores que se desean agregar a continuación un ejemplo.

```
27 • INSERT INTO category (nameCat) VALUES
28     ("higiene"),
29     ("comida"),
30     ("ropa");
31
32 • INSERT INTO items (name, price, id_cat) VALUES
33     ("Papel Higienico", 30, 1),
34     ("Chetos", 15, 2),
35     ("Camiseta", 100, 3),
36     ("Pasta dental", 25, 1),
37     ("Charritos", 28, 2);
38 |
39
```



# Modificar datos

- Para modificar un registro de la base de datos se requiere de una condición, la manera más efectiva de la condición es con el id del registro.
- Para modificar un registro se hace uso de la sentencia SQL “UPDATE”. A continuación un ejemplo.

```
40 • UPDATE items SET name = "Papel de baño", price = 28 WHERE idItem = 1;|  
41
```

- En la sentencia “SET” se pueden poner todos los atributos que se deseen cambiar.
- La condicional sería la sentencia “WHERE”. Esta sentencia no solo puede aplicar con la clave primaria si no con cualquier otro campo.



# Eliminar Datos

- Eliminar registros de la base de datos funciona de una manera similar al actualizar.
- Se usa la sentencia SQL “DELETE” y al igual que el “UPDATE” se necesita una condicional. La manera más eficiente es eliminar el registro a través del id, pero igualmente puede ser a través de cualquiera de los campos. A continuación un ejemplo

```
45 • DELETE FROM items WHERE idItem = 6; |
```

# Obtener Datos

- Para obtener datos se necesita de la sentencia SQL “SELECT” ésta seleccionará los registros dependiendo de la condicional que la des. La condicional se basará en lo que se necesite obtener. A continuación un ejemplo

47 • `SELECT * FROM items;`

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: `IA`

idItem	name	price	id_cat
1	Papel de baño	28	1
2	Chetos	15	2
3	Camiseta	100	3
4	Pasta dental	25	1
5	Charritos	28	2
NULL	NULL	NULL	NULL

# Obtener datos

Otro ejemplo sería si necesita obtener todos los registros que su precio sea mayor que 25

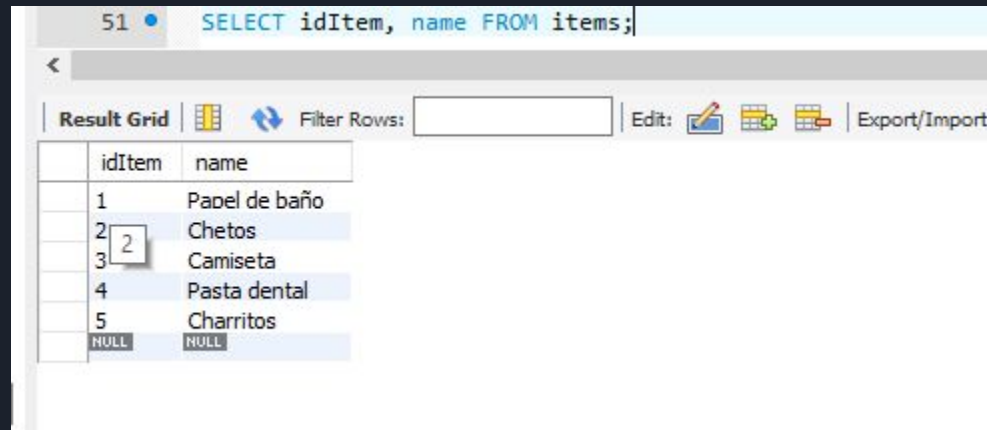
```
49 • SELECT * FROM items WHERE items.price > 25;
```

Result Grid | Filter Rows:  | Edit: | Export/Import: | Wrap Cell Content:

	idItem	name	price	id_cat
	1	Papel de baño	28	1
	3	Camiseta	100	3
	5	Charritos	28	2
	NULL	NULL	NULL	NULL

# Obtener Datos

- Si se necesita nada más una columna solo hace falta especificarla en vez del "\*". A continuación un ejemplo



The screenshot shows a database query tool interface. At the top, a SQL query is entered: `SELECT idItem, name FROM items;`. Below the query, there is a toolbar with icons for 'Result Grid', 'Filter Rows', 'Edit', and 'Export/Import'. The 'Result Grid' is active, displaying a table with two columns: 'idItem' and 'name'. The table contains five rows of data, with the second row (idItem 2, name Chetos) highlighted. The last row shows 'NULL' for both columns.

idItem	name
1	Papel de baño
2	Chetos
3	Camiseta
4	Pasta dental
5	Charritos
NULL	NULL

# Obtener datos de diferentes tablas

- Se necesita una consulta que me diga la categoría de todos los productos almacenados, para eso se necesita hacer una consulta a dos tablas diferentes, esto se resuelve a través de la sentencia SQL “JOIN” esta sentencia nos permite unir dos tablas que tengan datos en común, la manera más eficiente es hacerla comparando la clave foránea de una tabla con la clave primaria de la otra, a continuación un ejemplo.

```
53 • SELECT name, price, nameCat FROM
54 items JOIN category
55 ON items.id_cat = category.idCat;
```

<

Result Grid | | Filter Rows:  | Export: | Wrap Cell Content:

	name	price	nameCat
	Papel de baño	28	higiene
	Chetos	15	comida
	Camiseta	100	ropa
	Pasta dental	25	higiene
	Charritos	28	comida

- Como se puede observar también puedes seleccionar los campos que necesites de la otra tabla que estás uniendo

# Obtener datos de diferentes tablas

- Esta sentencia funciona igual que un “SELECT” normal ya que igual se le pueden agregar condicionales. A continuación un ejemplo

```
57 • SELECT name, price, nameCat FROM  
58 items JOIN category  
59 ON items.id_cat = category.idCat  
60 WHERE nameCat = "higiene";  
61  
62
```

<

Result Grid |   Filter Rows:  | Export:  | Wrap Cell Content: 

	name	price	nameCat
	Papel de baño	28	higiene
	Pasta dental	25	higiene



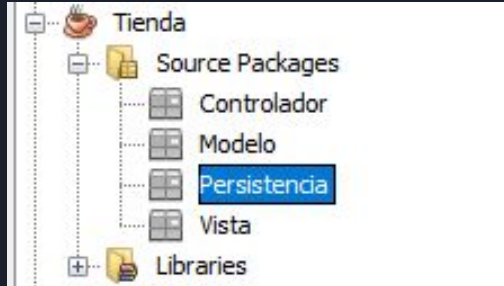


# Java Con MySQL

- Para crear una aplicación que se conecte a una base de datos se utilizará una arquitectura MVC (Modelo, Cliente, Servidor).
- Se necesita el .jar con las funciones para MySQL
- La conexión debe de estar activa
- La base de datos debe de estar creada
- Cuando se utilicen las tablas estas deben de estar creadas

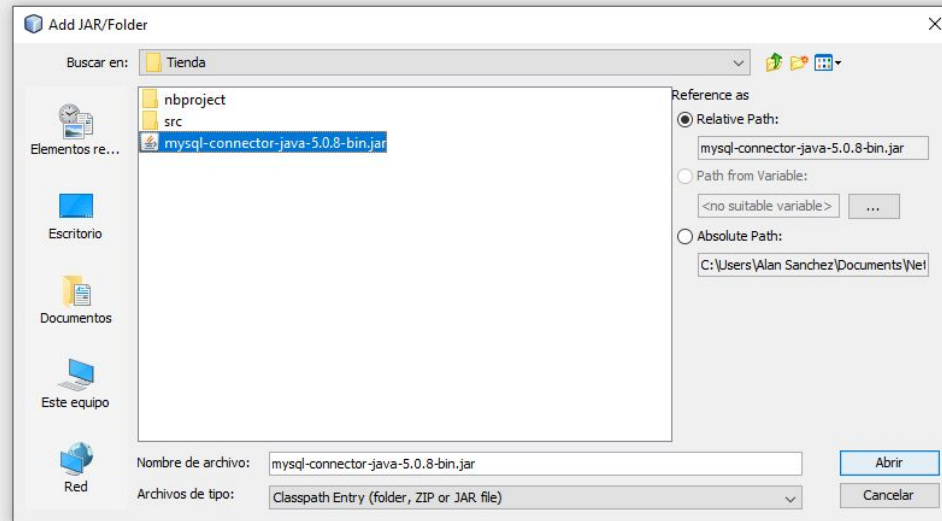
# Crear la arquitectura MVC

- Para crear la arquitectura del proyecto, se crearán tres paquetes que serían la vista, el modelo y los controladores a continuación se muestra la imagen de la estructura del proyecto.
- Se crea un paquete más, que sería el de la persistencia, ahí se crearán los DAO (Data Access Object), estas clases tendrán los métodos para realizar consultas a la base de datos



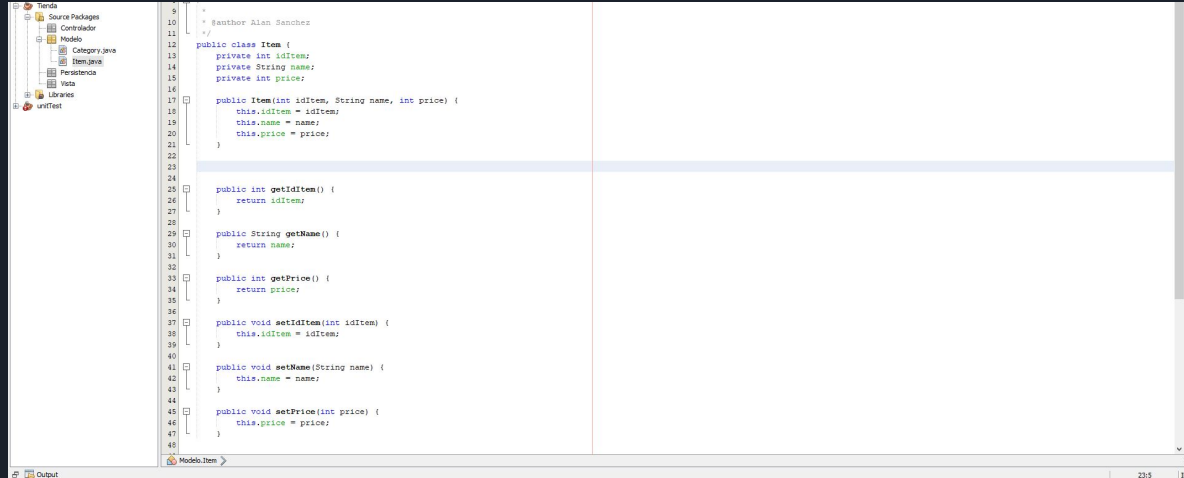
# Añadir el .jar de MySQL

- Para añadir el .jar al proyecto se da click derecho en donde dice “Libraries” y se selecciona la opción de “Add JAR/Folder” y se selecciona el .jar



# Crear el modelo

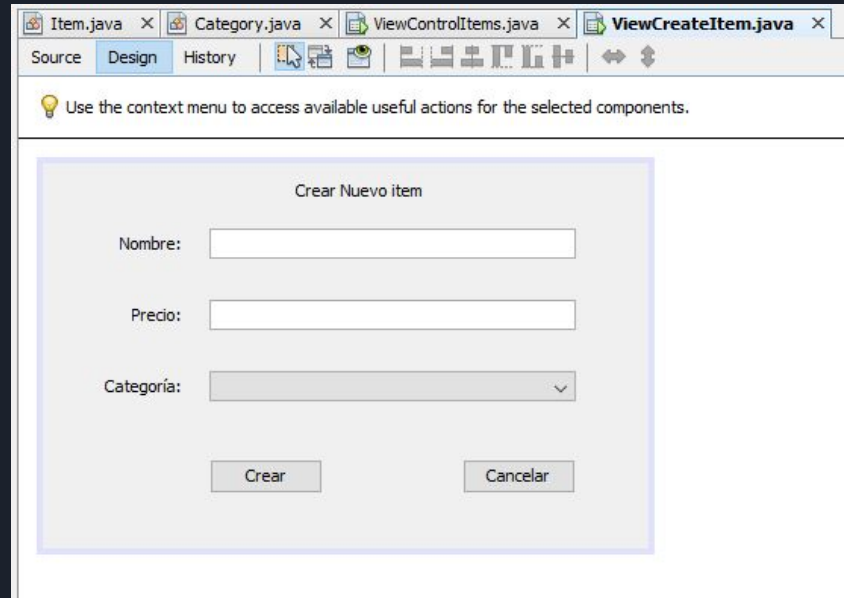
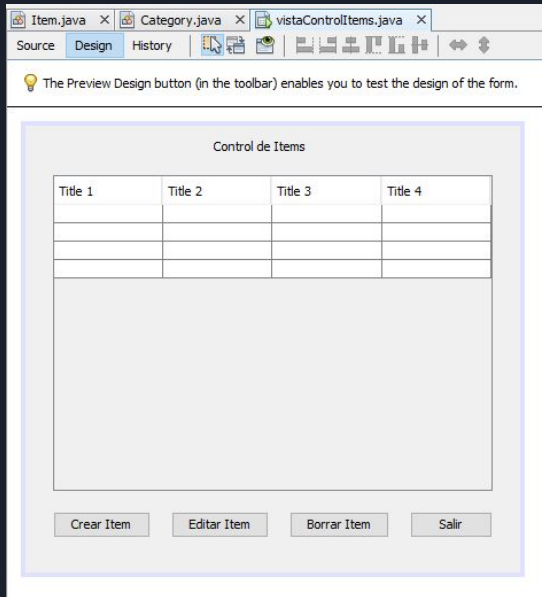
- Ahora se crean las clases del modelo, estas vendrían siendo las tablas que se crearon anteriormente.
- Estas clases llevarán los mismos atributos que en las tablas.
- Se crean los get y los Set de estos atributos, al igual que un constructor con todos los atributos
- A continuación se muestra la estructura del modelo



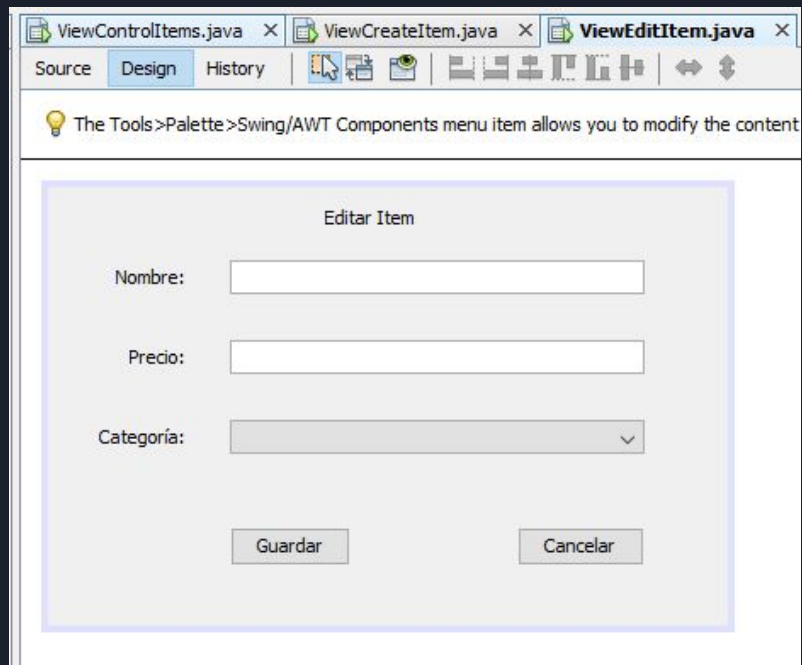
```
9  *  
10  * (author Alan Sanchez  
11  */  
12  public class Item {  
13      private int idItem;  
14      private String name;  
15      private int price;  
16  
17      public Item(int idItem, String name, int price) {  
18          this.idItem = idItem;  
19          this.name = name;  
20          this.price = price;  
21      }  
22  
23  
24  
25      public int getIdItem() {  
26          return idItem;  
27      }  
28  
29      public String getName() {  
30          return name;  
31      }  
32  
33      public int getPrice() {  
34          return price;  
35      }  
36  
37      public void setIdItem(int idItem) {  
38          this.idItem = idItem;  
39      }  
40  
41      public void setName(String name) {  
42          this.name = name;  
43      }  
44  
45      public void setPrice(int price) {  
46          this.price = price;  
47      }  
48  }
```

# Crear las vistas

- Para crear las vistas se usará el editor de NetBeans las vistas quedarán de la siguiente forma:



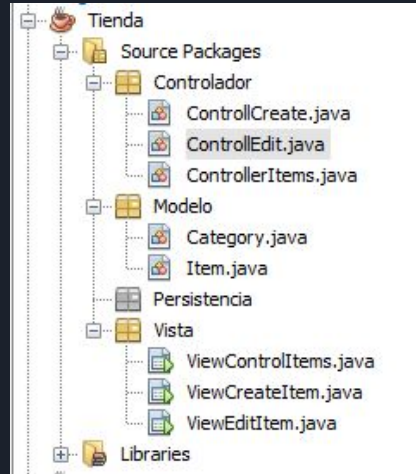
# Crear las vistas



- Luego de haber creado las vistas se cambiarán los nombre de los objetos de la vista para poder acceder a ellos luego, los métodos get se crean en la pestaña que dice “Source”

# Crear los controladores

- Los controladores se crearán de acuerdo a todas las vistas que se tengan, en este caso se crearán 3.
- Todos los controladores implementarán la clase “`ChangeListener`” que es lo que permitirá escuchar los eventos de los botones. A continuación se muestran las clases creadas.





# Controlador Principal

- El controlador principal tendrá un método que implementó de la clase ActionListener, este tendrá 4 if para verificar de donde proviene el evento que se dio en la ventana.
- Tendrá un constructor que recibirá la ventana principal que anteriormente se creó.
- Este constructor inicializa los listeners de los botones, llena la tabla con los datos de la base de datos y hace visible la ventan.
- Por el momento los if y el método estarán vacíos.
- El código se muestra en la siguiente diapositiva.



# Controlador Principal

```
8 import Vista.ViewControlItems;
9 import java.awt.event.ActionEvent;
10 import java.awt.event.ActionListener;
11 import javax.swing.JTable;
12
13 /**
14  *
15  * @author Alan Sanchez
16  */
17 public class ControllerItems implements ActionListener {
18
19     private ViewControlItems viewControlItems;
20
21     public ControllerItems(ViewControlItems viewControlItems) {
22         this.viewControlItems = viewControlItems;
23
24         this.viewControlItems.getJButtonCreate().addActionListener(this);
25         this.viewControlItems.getJButtonDelete().addActionListener(this);
26         this.viewControlItems.getJButtonEdit().addActionListener(this);
27         this.viewControlItems.getJButtonExit().addActionListener(this);
28
29         fillItemsTable(this.viewControlItems.getJTableItems());
30
31         this.viewControlItems.setVisible(true);
32     }
33
34
35     @Override
36     public void actionPerformed(ActionEvent e) {
37         if (e.getSource() == viewControlItems.getJButtonCreate()) {
38
39         } else if (e.getSource() == viewControlItems.getJButtonDelete()) {
40
41         } else if (e.getSource() == viewControlItems.getJButtonEdit()) {
42
43         } else if (e.getSource() == viewControlItems.getJButtonExit()) {
44             viewControlItems.dispose();
45             System.exit(0);
46         }
47     }
48
49     private void fillItemsTable(JTable jTableItems) {
50
51     }
52
53
54 }
```



# Controlador Crear item

- Este controlador al igual que el anterior también implementará el método heredado del ActionListener, este solo tendrá dos if ya que solo hay dos botones.
- Tendrá un constructor que reciba una ventana para crear un nuevo item que se realizó anteriormente.
- Este constructor inicializa los listeners de los dos botones.
- Tendrá una función para inicializar los items del combobox.
- tendrá dos if correspondientes a los eventos de los dos botones en la función ActionPerformed, el botón cancelar enviará de regreso a la vista principal
- una vez terminado este controlador se modifica el controlador principal para que al presionar el botón de crear nos dirija a la ventana para crear.
- El código se muestra a continuación.

# Controlador Crear item

```
8 import Vista.ViewControlItems;
9 import Vista.ViewCreateItem;
10 import java.awt.event.ActionEvent;
11 import java.awt.event.ActionListener;
12
13 /**
14  *
15  * @author Alan Sanchez
16  */
17 public class ControlCreate implements ActionListener{
18     private ViewCreateItem viewCreate;
19
20     public ControlCreate(ViewCreateItem viewCreate) {
21         this.viewCreate = viewCreate;
22
23         this.viewCreate.getJButtonCreate().addActionListener(this); //Inicializa el listener del evento del boton
24         this.viewCreate.getJButtonCancel().addActionListener(this); //Inicializa el listener del evento del boton
25
26         initializeCombobox();
27
28         this.viewCreate.setVisible(true);
29     }
30
31     @Override
32     public void actionPerformed(ActionEvent e) {
33         if(e.getSource() == viewCreate.getJButtonCreate()){
34
35         }else if(e.getSource() == viewCreate.getJButtonCancel()){
36             viewCreate.dispose();
37             ViewControlItems controlItems = new ViewControlItems();
38             ControllerItems control = new ControllerItems(controlItems);
39         }
40     }
41
42     private void initializeCombobox() {
43
44     }
45
46 }
47
48
49
```

```
@Override
public void actionPerformed(ActionEvent e) {
    if(e.getSource() == viewControlItems.getJButtonCreate()){
        ViewCreateItem createItem = new ViewCreateItem();
        ControlCreate control = new ControlCreate(createItem);
        viewControlItems.dispose();
    }
}
```



# Controlador Editar Item

- Este controlador funciona de la misma forma que el anterior controlador, lo único que cambiará es la acción que tendrá al presionar el botón de guardar.
- El constructor además de recibir la vista este recibirá un objeto Item.
- Se modificará igual el controlador principal para que al presionar el botón de editar este nos lleve a la vista para editar. El código se muestra a continuación

# Controlador Editar Item

```
6 package Controlador;
7
8 import Modelo.Item;
9 import Vista.ViewControlItems;
10 import Vista.ViewEditItem;
11 import java.awt.event.ActionEvent;
12 import java.awt.event.ActionListener;
13
14 /**
15  *
16  * @author Alan Sanchez
17  */
18 public class ControlEdit implements ActionListener {
19     private ViewEditItem viewEdit;
20     private Item item;
21
22     public ControlEdit(ViewEditItem viewEdit, Item item) {
23         this.viewEdit = viewEdit;
24         this.item = item;
25
26         this.viewEdit.getjButtonSave().addActionListener(this);
27         this.viewEdit.getjButtonCancel().addActionListener(this);
28
29         initializeCombobox();
30
31         this.viewEdit.setVisible(true);
32     }
33
34     @Override
35     public void actionPerformed(ActionEvent e) {
36         if (e.getSource() == viewEdit.getjButtonSave()) {
37
38         } else if (e.getSource() == viewEdit.getjButtonCancel()) {
39             ViewControlItems viewItems = new ViewControlItems();
40             ControllerItems controll = new ControllerItems(viewItems);
41             viewEdit.dispose();
42         }
43     }
44
45     private void initializeCombobox() {
46
47     }
48
49
50 }
```



# Persistencia

- Ahora se pasarán a crear las clases y los métodos que nos permitirán acceder a la base de datos
- Se creará un DAOGeneral (Data Access Object) este será una clase que los demás DAO heredarán
- En el DAOGeneral tendrá solo un atributo de tipo “Conexion” y dos métodos que son para abrir y cerrar la conexión a la base de datos.
- Por cada tabla que creamos se crearán los DAO respectivos, en este caso serán dos, el DAOItems y el DAOCategory
- En el DAOItems se generarán los métodos para seleccionar, eliminar, editar y crear
- A continuación se mostrarán los códigos para los DAO



# Persistencia: DAOGeneral

- Se crea el método MySqlConnection al cual se le pasará como parámetros el nombre de la base de datos, el usuario y la contraseña. Este utilizará el .jar que se puso en el proyecto para crear e inicializar la Conexión. El servidor será “localhost:puerto\_elegido”
- Se crea el método closeConnection que cerrará la conexión para que la base de datos no se sobresature de conexiones
- Se crea un constructor que ejecuta el método MySqlConnection y se le pasa como parámetros “tu\_usuario, tu\_contraseña, tienda”
- El código se muestra a continuación

# Persistencia: DAOGeneral

```
ControllerItems.java x ControllerCreate.java x ControllerEdit.java x DAOGeneral.java x DAOGeneral.java x
Source History
10 import java.sql.SQLException;
11 import java.util.logging.Level;
12 import java.util.logging.Logger;
13
14 /**
15  *
16  * @author Alan Sanchez
17  */
18 public class DAOGeneral {
19     private static Connection Conexion;
20
21     public DAOGeneral() {
22         MySQLConnection("root", "root", "tienda");
23     }
24
25
26
27
28     private void MySQLConnection(String user, String pass, String db_name) {
29         try {
30             Class.forName("com.mysql.jdbc.Driver");
31             Conexion = DriverManager.getConnection("jdbc:mysql://localhost:3306/" + db_name, user, pass);
32         } catch (ClassNotFoundException ex) {
33             Logger.getLogger(DAOGeneral.class.getName()).log(Level.SEVERE, null, ex);
34         } catch (SQLException ex) {
35             Logger.getLogger(DAOGeneral.class.getName()).log(Level.SEVERE, null, ex);
36         }
37     }
38
39     //Cierra la conexión con la base de datos
40     public void closeConnection() {
41         try {
42             Conexion.close();
43         } catch (SQLException ex) {
44             Logger.getLogger(DAOGeneral.class.getName()).log(Level.SEVERE, null, ex);
45         }
46     }
47
48     public static Connection getConexion() {
49         return Conexion;
50     }
51 }
52
```





# Persistencia: DAOItems

- Esta clase debe de heredar de la clase DAOGeneral.
- Se crea el método InsertItem que recibe como parámetro un objeto de tipo Item.
- Se crea el método selectAllItems que devuelve un ArrayList<Items> que contendrá todos los items que se hayan registrado en la base de datos.
- Se crea el método updateItem que recibirá como parámetro un objeto de tipo Item.
- Se crea el método deleteItem que recibirá un entero como parámetro, este entero .representa el id del item que se desea borrar.
- En cada método se creará una variable tipo String que se llamará query y se creará un statment para que este ejecute el query que se escribió, si el query es un select se creará un resultSet para obtener todos los datos.
- El código se muestra a continuación.

# Persistencia: DAOItems

```
20  * @author Alan Sanchez
21  */
22  public class DAOItems extends DAOGeneral {
23
24      public void insertItem(Item item) {
25          try {
26              String query = "INSERT INTO items (name, price, id_cat) VALUES ( "
27                  + "\"" + item.getName() + "\", "
28                  + "\"" + item.getPrice() + "\", "
29                  + "\"" + item.getIdCat() + "\"" );
30              Statement statement = getConexion().createStatement();
31              statement.executeUpdate(query);
32              JOptionPane.showMessageDialog(null, "Datos Almacenados correctamente");
33          } catch (SQLException ex) {
34              Logger.getLogger(DAOItems.class.getName()).log(Level.SEVERE, null, ex);
35          }
36      }
37
38      public ArrayList<Item> selectAllItems() {
39          try {
40              String query = "SELECT * FROM items";
41              Statement statement = getConexion().createStatement();
42              ResultSet resultSet = statement.executeQuery(query);
43              ArrayList<Item> listItems = new ArrayList();
44              while (resultSet.next()) {
45                  int idItem = Integer.parseInt(resultSet.getString("idItem"));
46                  String name = resultSet.getString("name");
47                  int price = Integer.parseInt(resultSet.getString("price"));
48                  int idCat = Integer.parseInt(resultSet.getString("id_cat"));
49                  Item item = new Item(idItem, name, price, idCat);
50                  listItems.add(item);
51              }
52              return listItems;
53          } catch (SQLException ex) {
54              Logger.getLogger(DAOItems.class.getName()).log(Level.SEVERE, null, ex);
55              return null;
56          }
57      }
58  }
```

```
59  public void updateItem(Item item) {
60      try {
61          String query = "UPDATE items SET name =\"" + item.getName() + "\", "
62              + "price =\"" + item.getPrice() + "\", "
63              + "id_cat =\"" + item.getIdCat() + "\" "
64              + "WHERE idItem =\"" + item.getIdItem() + "\"";
65          Statement statement = getConexion().createStatement();
66          statement.executeUpdate(query);
67          JOptionPane.showMessageDialog(null, "Se ha modificado el item con éxito");
68      } catch (SQLException ex) {
69          Logger.getLogger(DAOItems.class.getName()).log(Level.SEVERE, null, ex);
70      }
71  }
72
73  public void deleteItem(int idItem) {
74      int answer = JOptionPane.showConfirmDialog(null, "¿Está seguro de que desea eliminar el item seleccionado?",
75          "Advertencia", JOptionPane.YES_NO_OPTION, JOptionPane.ERROR_MESSAGE);
76
77      boolean confirmed = answer == 0;
78
79      if (confirmed) {
80          try {
81              String query = "DELETE FROM items WHERE idItem = \"" + idItem + "\"";
82              Statement statement = getConexion().createStatement();
83              statement.executeUpdate(query);
84
85              JOptionPane.showMessageDialog(null, "Se eliminó el item de manera exitosa");
86          } catch (SQLException ex) {
87              Logger.getLogger(DAOItems.class.getName()).log(Level.SEVERE, null, ex);
88          }
89      } else {
90          JOptionPane.showMessageDialog(null, "Operación cancelada");
91      }
92  }
93  }
```



# Persistencia: DAOCategory

- Esta clase debe de heredar de la clase DAOGeneral.
- Esta clase tendrá un método para obtener todos los datos que se tenga en la tabla, regresará un `ArrayList<Category>`.
- En cada método se creará una variable tipo `String` que se llamará `query` y se creará un `statment` para que este ejecute el `query` que se escribió, si el `query` es un `select` se creará un `resultSet` para obtener todos los datos.
- El código se muestra a continuación.

# Persistencia: DAOCategory

```
package Persistencia;

import Modelo.Category;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author Alan Sanchez
 */
public class DAOCategory extends DAOGeneral{

    public ArrayList<Category> selectCategory(){
        try {
            ArrayList<Category> listCategory = new ArrayList();
            String query = "SELECT * FROM category";
            Statement statement = getConexion().createStatement();
            ResultSet resultSet = statement.executeQuery(query);
            while(resultSet.next()){
                int idCat = Integer.parseInt(resultSet.getString("idCat"));
                String name = resultSet.getString("nameCat");
                Category cat = new Category(idCat, name);
                listCategory.add(cat);
            }
            return listCategory;
        } catch (SQLException ex) {
            Logger.getLogger(DAOCategory.class.getName()).log(Level.SEVERE, null, ex);
            return null;
        }
    }
}
```



# Controlador Principal

- Una vez creado los DAO ahora pasaremos a utilizarlos en la vista
- Se empezará en el ControllItems
- Primero Se llenará la tabla a través del método fillItemstable, se utilizará el DAOitems con el método selectAllItems.
- Luego en la acción que corresponde a borrar utilizaremos el DAOitems con el método para borrar el ítem, para eso el usuario seleccionará una fila y esa será la que se borrará
- Ahora en la acción que corresponde a editar ítem se obtendrá los valores de la fila que el usuario seleccionó y se creará un objeto de tipo ítem que se mandará al controlador para editar
- A continuación se muestra todo el código.

# Controlador Principal

```
private void fillItemsTable(JTable jTableItems) {
    DefaultTableModel model = new DefaultTableModel();

    jTableItems.setModel(model);

    model.addColumn("Id Item");
    model.addColumn("Nombre del item");
    model.addColumn("Precio");
    model.addColumn("Id Categoria");

    Object[] column = new Object[4];

    DAOItems dao = new DAOItems();
    ArrayList<Item> listItems = dao.selectAllItems();
    dao.closeConnection();
    for (Item item : listItems) {
        column[0] = item.getIdItem();
        column[1] = item.getName();
        column[2] = item.getPrice();
        column[3] = item.getIdCat();
        model.addRow(column);
    }
}
```

```
@Override
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == viewControlItems.getjButtonCreate()) {
        ViewCreateItem createItem = new ViewCreateItem();
        ControllCreate control = new ControllCreate(createItem);
        viewControlItems.dispose();
    } else if (e.getSource() == viewControlItems.getjButtonDelete()) {
        //Obtiene el numero de la fila seleccionada
        //Verifica si se selecciono una fila
        //Obtiene el valor del id de la fila seleccionada

        int rowSelected = viewControlItems.getjTableItems().getSelectedRow();
        if (rowSelected > 0) {
            int idSelected = (int) viewControlItems.getjTableItems().getValueAt(rowSelected, 0);
            DAOItems dao = new DAOItems();
            dao.deleteItem(idSelected);
            fillItemsTable(viewControlItems.getjTableItems());
        } else {
            JOptionPane.showMessageDialog(null, "Seleccione Una Fila");
        }
    } else if (e.getSource() == viewControlItems.getjButtonEdit()) {
        Item itemSelected = getSelectedItem();
        if (itemSelected != null) {
            ViewEditItem viewEdit = new ViewEditItem();
            ControllEdit controll = new ControllEdit(viewEdit, itemSelected);
            viewControlItems.dispose();
        }
    } else if (e.getSource() == viewControlItems.getjButtonExit()) {
        viewControlItems.dispose();
        System.exit(0);
    }
}
```

# Controlador Principal

```
private Item getSelectedItem() {  
    int rowSelected = viewControlItems.getjTableItems().getSelectedRow();  
    if (rowSelected > 0) {  
        int idItem = (int) viewControlItems.getjTableItems().getValueAt(rowSelected, 0);  
        String name = (String) viewControlItems.getjTableItems().getValueAt(rowSelected, 1);  
        int price = (int) viewControlItems.getjTableItems().getValueAt(rowSelected, 2);  
        int idCat = (int) viewControlItems.getjTableItems().getValueAt(rowSelected, 3);  
        Item item = new Item(idItem, name, price, idCat);  
        return item;  
    } else {  
        JOptionPane.showMessageDialog(null, "Seleccione Una Fila");  
        return null;  
    }  
}
```



# Controlador Crear item

- Primero se iniciará el combobox con la función “initializeCombobox”
- Luego en el evento del botón guardar se creará un objeto a partir de los valores ingresados en los textfields
- Se usará la función del DAOItem insertItem en el cual se le pasa un objeto de tipo item
- Luego de que se haya creado con éxito se regresará a la pantalla principal.
- A continuación se muestra el código



# Controlador Crear item

```
37 @Override
38 public void actionPerformed(ActionEvent e) {
39     if(e.getSource() == viewCreate.getJButtonCreate()){
40
41         String name = viewCreate.getJTextFieldName().getText();
42         int price = Integer.parseInt(viewCreate.getJTextFieldPrice().getText());
43         int idCat = (int) viewCreate.getJComboBoxCategory().getSelectedIndex()+1;
44         Item item = new Item(name, price, idCat);
45
46         DAOItems dao = new DAOItems();
47         dao.insertItem(item);
48
49         viewCreate.dispose();
50     }else if(e.getSource() == viewCreate.getJButtonCancel()){
51         viewCreate.dispose();
52         ViewControlItems controlItems = new ViewControlItems();
53         ControllerItems control = new ControllerItems(controlItems);
54     }
55 }
```

```
57 private void initializeCombobox() {
58     DAOCategory dao = new DAOCategory();
59     ArrayList<Category> listCategory = dao.selectCategory();
60
61     for(Category category : listCategory){
62         viewCreate.getJComboBoxCategory().addItem(category.getNameCat());
63     }
64 }
65
```



# Controlador Editar item

- Este controlador funciona de una manera similar al crear item, ya que se creará un objeto a partir de los fields.
- Este controlador varía en la función para llenar todos los field a partir del objeto item que se mandó como parámetro en el constructor al igual que utiliza la función updateItem del Objeto DAOItems.
- El código se muestra a continuación.

# Controlador Editar Item

```
39 @Override
40 public void actionPerformed(ActionEvent e) {
41     if(e.getSource() == viewEdit.getJButtonSave()) {
42         String name = viewEdit.getJTextFieldName().getText();
43         int price = Integer.parseInt(viewEdit.getJTextFieldPrice().getText());
44         int idCat = viewEdit.getJComboBoxCategory().getSelectedIndex() + 1;
45
46         item.setIdCat(idCat);
47         item.setName(name);
48         item.setPrice(price);
49
50         DAOItems dao = new DAOItems();
51         dao.updateItem(item);
52
53     } else if(e.getSource() == viewEdit.getJButtonCancel()) {
54         ViewControlItems viewItems = new ViewControlItems();
55         ControllerItems controll = new ControllerItems(viewItems);
56         viewEdit.dispose();
57     }
58 }
59
60 private void initializeCombobox() {
61     DAOCategory dao = new DAOCategory();
62     ArrayList<Category> listCategory = dao.selectCategory();
63
64     for(Category category : listCategory) {
65         viewEdit.getJComboBoxCategory().addItem(category.getNameCat());
66     }
67 }
68
69 private void fillAllFields(Item item) {
70     viewEdit.getJTextFieldName().setText(item.getName());
71     viewEdit.getJTextFieldPrice().setText(String.valueOf(item.getPrice()));
72     viewEdit.getJComboBoxCategory().setSelectedIndex(item.getIdCat()-1);
73 }
74
75
```