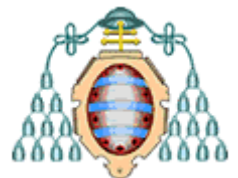


Arquitectura

Ingeniería del Software

José Ramón de Diego





Contenido

- Diseño del software
- Tipos de diseño
- Arquitectura



Diseño del Software

Describe:

- La arquitectura del software (organización en componentes)
- Los interfaces entre componentes.
- El detalle de cada componente para poder implementarlo

Tipos de diseño

■ Arquitectura

- ☐ Describe la estructura y organización a alto nivel. (subsistemas, componentes y sus relaciones)
- ☐ Es el primer paso en el diseño y su resultado es la arquitectura del sistema

■ Detallado

- ☐ Describe cada componente y su comportamiento, permite su construcción

Arquitectura

- La arquitectura del sistema es su estructura interna
- Junto con la arquitectura será necesario definir:
 - Implementación de los datos (clases, BDs)
 - Implementación del comportamiento (métodos)
 - Implementación del interfaz (casos reales)

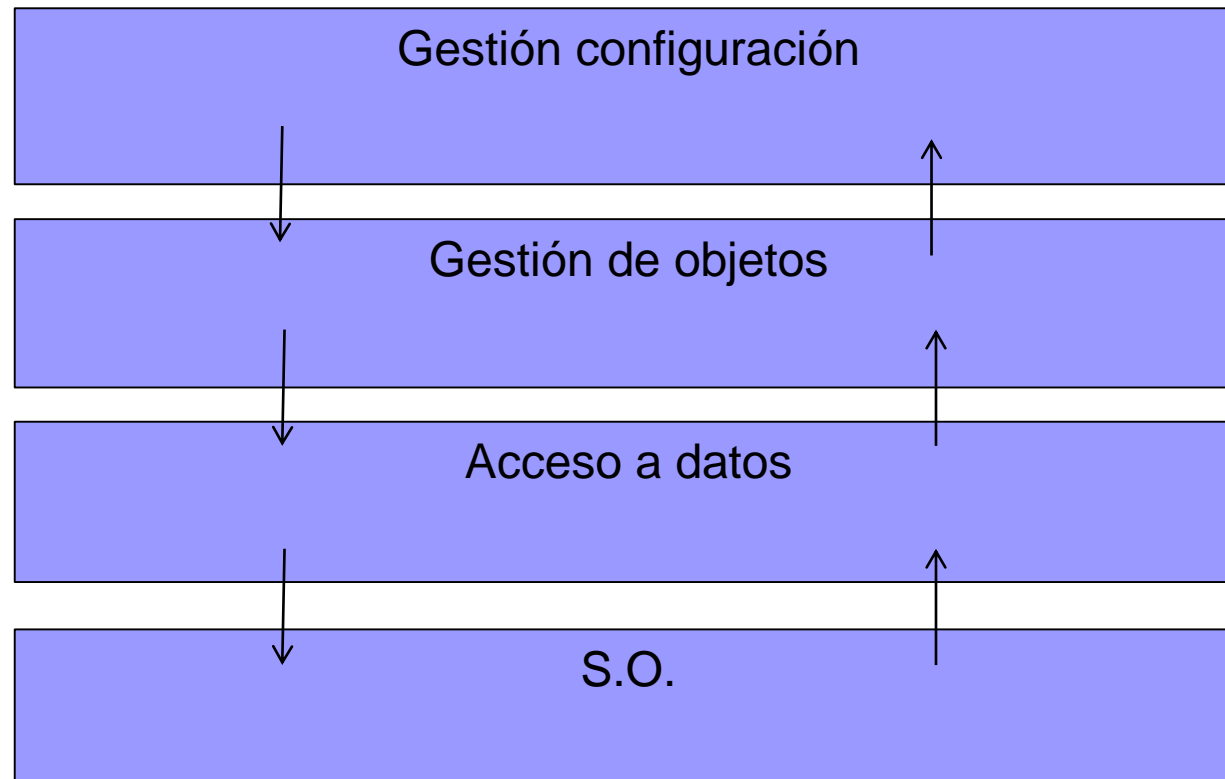
Arquitectura

- A tener en cuenta:
 - ¿Existe alguna arquitectura generica que podamos usar?
 - ¿Cuál es la distribución del sistema?
 - ¿Qué módulos tendrá?
 - ¿Cuál será la estrategia de control a seguir?
 - ¿Cómo lo vamos a documentar?

Tipos de arquitecturas

- General:
 - Capas, repositorio compartido
- Distribuidos
 - Cliente-servidor
- Interactivos
 - Modelo-Vista-Controlador
- Adaptable
 - Micro nucleo, reflexión
- Otros

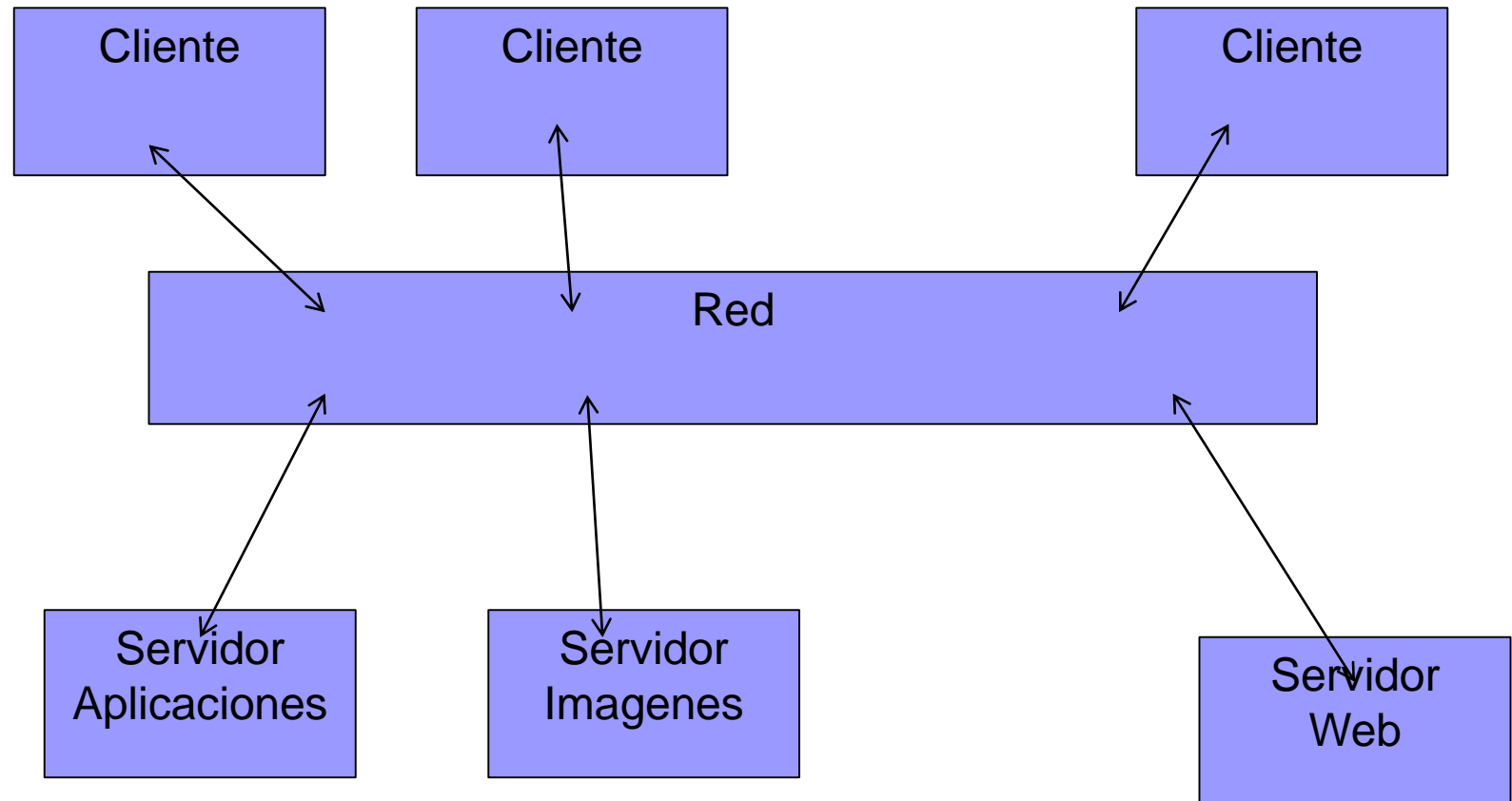
Arquitectura de capas



Repositorio compartido

- Los datos se guardan en un BD compartida:
 - Distintos subsistemas acceder a la misma BD
 - Permite compartir grandes cantidades de datos
 - Cada subsistema no necesita preocuparse de los backups
 - Es necesario tener el mismo modelo de datos para todos -> dificulta la evolución
 - La distribución eficiente de los datos es difícil.

Cliente - servidor

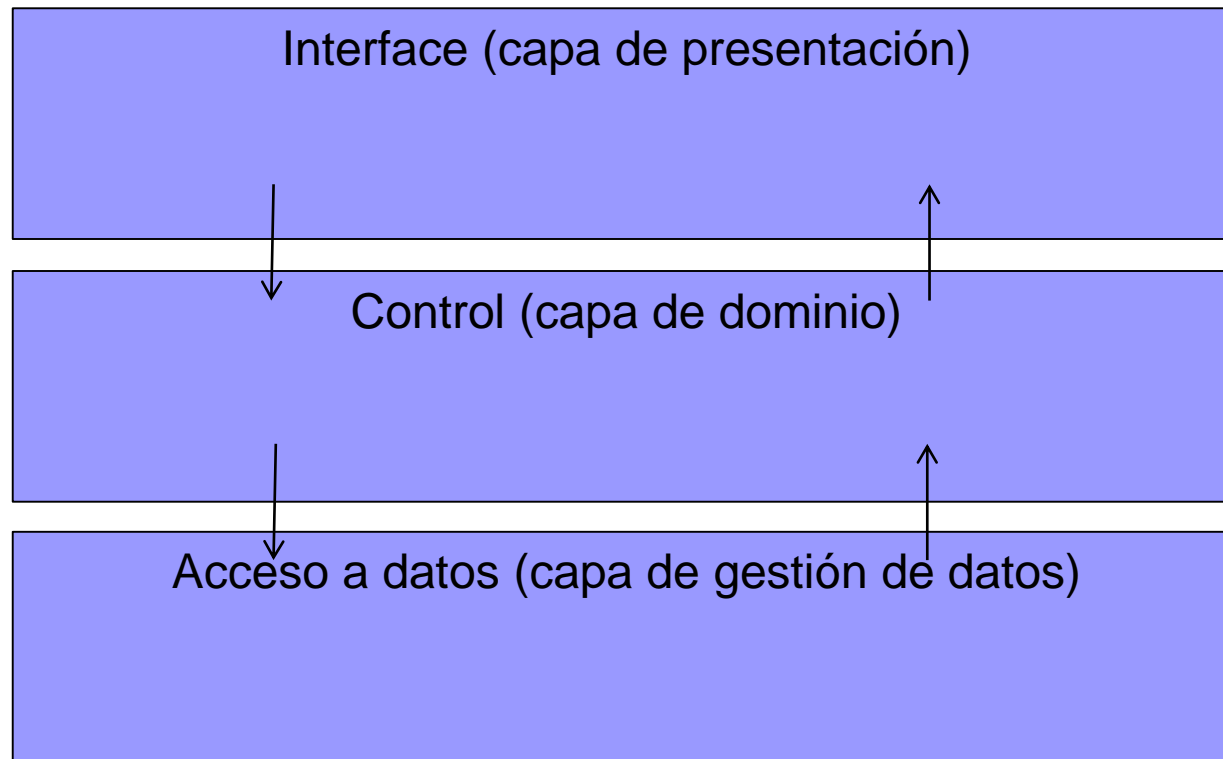


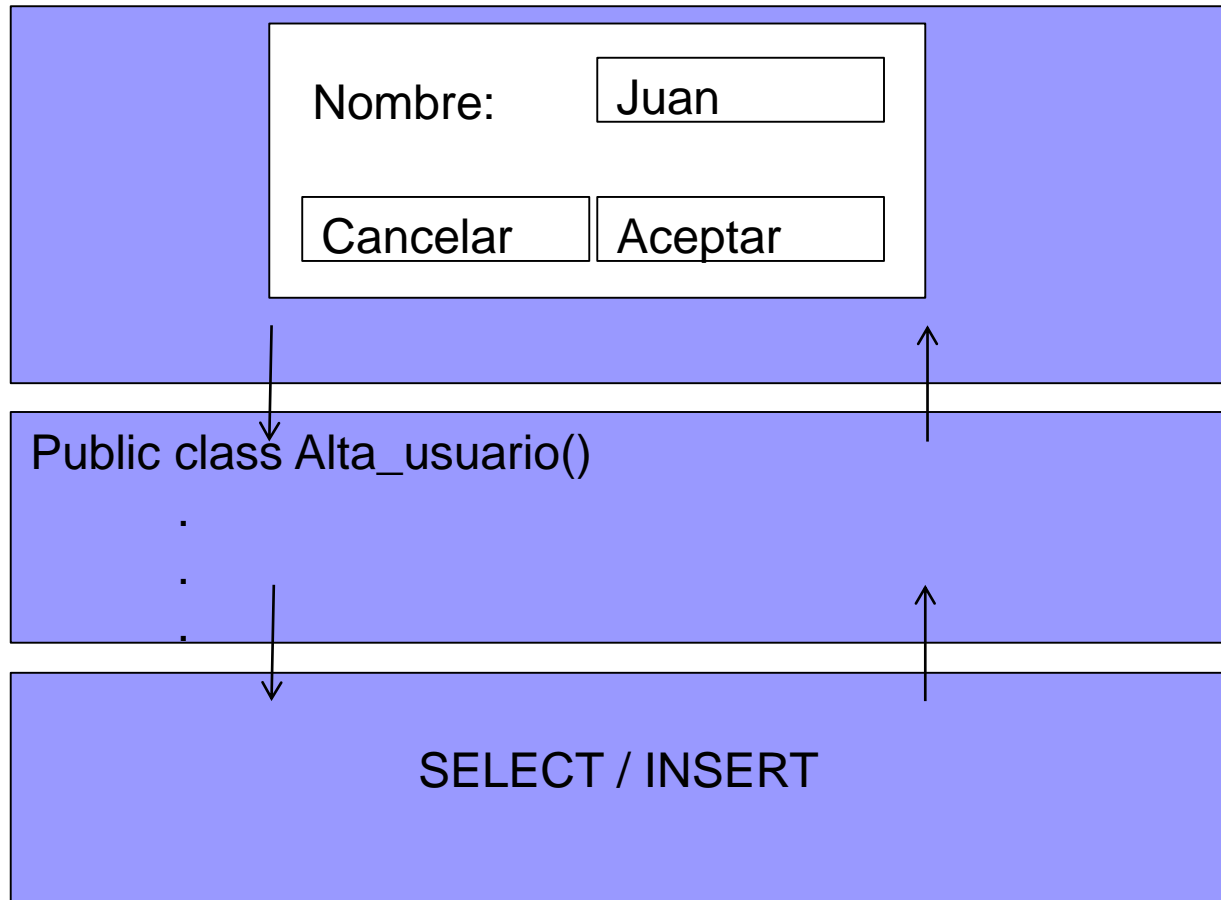
Cliente - servidor

- Distribución de datos real
- Uso de sistemas en red correcta
- Fácil ampliación

- Cada subsistema puede usar modelos de datos diferentes -> ineficiente
- Gestor en cada servidor -> redundancia
- Difícil controlar que servicios están disponibles

Arquitectura de tres capas (habitual)







Principal ventaja.

- Cambios fáciles en cada nivel con pocos efectos sobre el resto.
- Aísla la lógica de la aplicación convirtiéndola en una capa intermedia
- Permite distribuir las capas por nodos físicos.

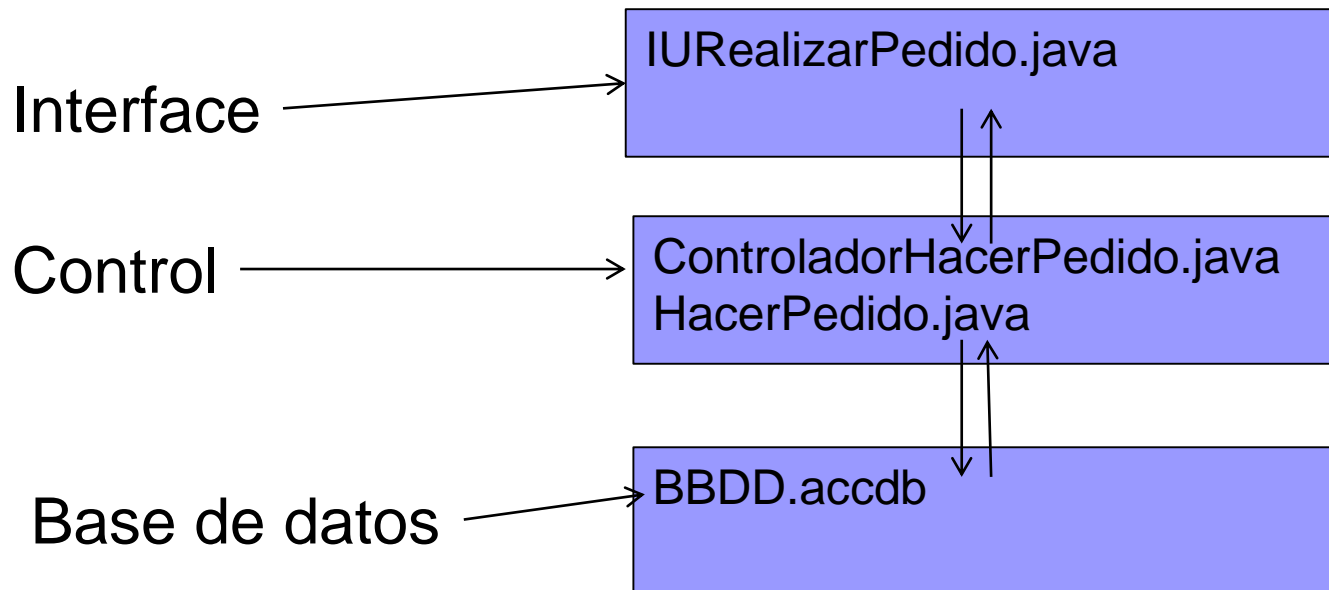


Implementación física

- La capa de presentación puede implementarse en el nodo cliente.
- La capa de negocio puede implementarse en un nodo servidor de aplicaciones.
- La capa de datos en el nodo servidor de DB.

Implementación Lógica-Ejemplo

- Sistema: Máquinas Vending
 - Caso de Uso: “Realizar pedido”



IURealizarPedido.java

```
package IU;

import java.awt.EventQueue;

public class IURealizarPedido {

    // Declaraciones de elementos gráficos
    [
        .....
        .....
    ]

    // Creamos el objeto controlador de la clase ControladorHacerPedido
    private ControladorHacerPedido controlador = new ControladorHacerPedido();
    // Creamos un modelo para la lista de proveedores, para manejar la lista con mayor facilidad
    private DefaultListModel modelListProv=new DefaultListModel();
    // Creamos un modelo para la lista de productos, para manejar la lista con mayor facilidad
    private DefaultListModel modelListProd=new DefaultListModel();

    // Variable que almacenará el Proveedor que se seleccione de la lista de los proveedores
    String ProveedorSeleccionado;
    // Variable que almacenará el Producto que se seleccione de la lista de los productos
    String ProductoSeleccionado;
    [
        .....
        .....
    ]
}
```

IURealizarPedido.java

```
private void initialize() {  
    frmMquinasVending = new JFrame();  
    frmMquinasVending.setTitle("M\u00E1quinas Vending - Realizar pedido manual");  
    [...]  
  
    JLabel label_1 = new JLabel("Pulse aqu\u00ED para mostrar los proveedores:");  
    label_1.setBounds(27, 65, 249, 14);  
    frmMquinasVending.getContentPane().add(label_1);  
  
    /* Con este botón llamamos al controlador, el cuál llamará a la clase que se conectará a la base de datos y  
    hará la consulta que nos devolverá una serie de datos que imprimiremos aquí en una lista. El botón solo podrá ser pulsado una vez para evitar llamadas innecesarias al resto de clases. */  
  
    btnIniciar = new JButton("Iniciar");  
    btnIniciar.addActionListener(new ActionListener() {  
        public void actionPerformed(ActionEvent e) {  
  
            controlador.Conectar();  
            ControladorHacerPedido.queryMostrarProveedor();  
            listaProveedores = controlador.rellenarListaProveedores(modelListProv, listaProveedores);  
            btnIniciar.setEnabled(false);  
        }  
    });  
    btnIniciar.setBounds(271, 61, 88, 23);  
    frmMquinasVending.getContentPane().add(btnIniciar);  
  
    JLabel label_2 = new JLabel("Seleccione el proveedor al que quiere pedir:");
```

Llamamos al controlador para realizar las tareas

IURealizarPedido.java

```
public void actionPerformed(ActionEvent arg0) {  
    // Utilizaremos un JDialog para mostrar un aviso en el caso de que se intente seleccionar u  
    if(listaProveedores.getSelectedValue()==null) {  
        JOptionPane.showMessageDialog(null,"Para seleccionar un proveedor tiene que existir alg  
        return;  
    }  
  
    ProveedorSeleccionado = listaProveedores.getSelectedValue().toString(); // Obtenemos el ele  
    labelProveedor.setText(ProveedorSeleccionado); // Lo imprimimos en una label  
  
    // A partir de aqui, se realizará una nueva conexión a la base de datos para obtener los pr  
    controlador.Conectar();  
    ControladorHacerPedido.queryMostrarProductos(ProveedorSeleccionado);  
    listaProductos = controlador.rellenarListaProductos(modelListProd,listaProductos,ProveedorS  
  
    btnSeleccionar.setEnabled(false); // Dejamos inactivo el botón hasta que se indique que se  
    btnSeleccionarOtroProveedor.setEnabled(true); // Nos aseguramos de que el botón de "Selecci  
}  
  
    btnSeleccionar.setBounds(291, 207, 125, 23);  
    jQuinasVending.getContentPane().add(btnSeleccionar);  
  
    lblFecha = new JLabel("Fecha:");  
    lblFecha.setBounds(388, 273, 46, 14);  
    jQuinasVending.getContentPane().add(lblFecha);
```

Solicitamos al controlador
la lista de productos

IURealizarPedido.java

```
btnRealizarPedido.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        // Utilizaremos un JDialog para mostrar un aviso en el caso de que se intente realizar  
        if(table.getValueAt(1,0)==null) {  
            JOptionPane.showMessageDialog(null,"No puede hacer un pedido sin seleccionar producto");  
            return;  
        }  
  
        // Llamamos a este método donde le pasamos toda la información con la finalidad de crear el pedido  
        IdPedido = controlador.RealizarPedido(table, textPane, formattedTextField);  
        if(IdPedido > 0)  
            JOptionPane.showMessageDialog(null,"El pedido '"+IdPedido+"' se ha realizado correctamente");  
    }  
});  
btnRealizarPedido.setBounds(359, 513, 149, 23);  
frmMquinasVending.getContentPane().add(btnRealizarPedido);  
  
JLabel lblTablaDePedido = new JLabel("Tabla de pedido:");  
lblTablaDePedido.setBounds(307, 332, 135, 14);  
frmMquinasVending.getContentPane().add(lblTablaDePedido);  
  
// Tabla de pedido  
modeloTabla = new DefaultTableModel(); // Atributo de clase  
table = new JTable(modeloTabla);  
table.setFont(new Font("Tahoma", Font.PLAIN, 11));
```

Solicitamos al controlador dar el alta del pedido si todo es correcto

ControladorHacerPedido.java

Gestiona las ordenes recibidas desde el interface

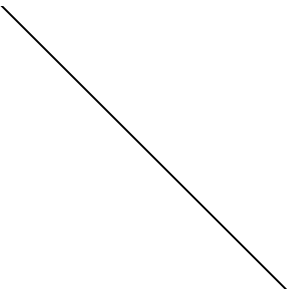
```
public class ControladorHacerPedido {  
  
    // Creamos el objeto hp de la clase HacerPedido  
    static HacerPedido hp = new HacerPedido();  
  
    /**  
     * Iniciamos la aplicación  
     * @param args  
     */  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
    }  
  
    /**  
     * Método que recibe la orden de conectar y la envía a la clase HacerPedido donde realiza la  
     */  
    public void Conectar() {  
        hp.Conectar();  
    }  
}
```

Se crea la instancia de la clase con la que vamos a trabajar

ControladorHacerPedido.java

Gestiona las ordenes recibidas desde el interface

```
public int RealizarPedido(JTable table, JTextPane textPane, JFormattedTextField formattedTe  
    int IdPedido = hp.RealizarPedido(table, textPane, formattedTextField);  
    return IdPedido;  
}
```



Gestión de un nuevo pedido,
Se recibe la petición del
Interface y se redirige hacia
la clase correspondiente.

HacerPedido.java

Clase con la parte de negocio que realiza las tareas

```
public void Conectar() {  
  
    try {  
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  
        conexion = DriverManager.getConnection("jdbc:odbc:bbddVending");  
        if(!conexion.isClosed()){  
            System.out.println("La Base de Datos se ha conectado con éxito");  
        }  
        else{  
            System.out.println("La Base de Datos NO se ha conectado");  
        }  
    }  
    catch (Exception ex)  
    {  
        ex.printStackTrace();  
    }  
}
```

Conexión con la base datos

HacerPedido.java

```
public ResultSet SelectNombreProveedor() {  
    try {  
        Statement stmt = conexion.createStatement();  
        rs = stmt.executeQuery("select Nombre from Proveedor");  
    }  
    catch (SQLException e) {  
        System.out.println("Fallo al obtener los datos de la base de datos: "+e);  
    }  
    return rs;  
}
```

Realización de una consulta
a la base de datos
(Petición inicial desde el
interface para tener la lista de
Proveedores)

HacerPedido.java

```
private int ActualizarTablaPedido(String fecha, String estado2, String text) {
    ResultSet rs = null;
    int IdPedido = 0;

    try {
        Statement stmt = conexion.createStatement();
        // Realizamos el insert en la tabla Pedido
        stmt.executeUpdate("INSERT INTO Pedido(Fecha, Estado, Notas) VALUES" +
            "(" + fecha + ", " + estado2 + ", " + text + ")");

        // Ahora tenemos que obtener el identificador de la tabla Pedido, ya que tenemos que in
        // y sin el identificador no podriamos hacerlo, lo obtenemos de la siguiente manera:
        rs = stmt.executeQuery("SELECT IdPedido FROM Pedido");
        // Una vez que tenemos todos los identificadores lo más sencillo es ir al final y coger
        while(rs.next())
            IdPedido =(int)rs.getObject("IdPedido");
    }
    catch (SQLException e) {
        System.out.println("Error al insertar datos en la tab
    }

    return IdPedido;
}
```

Alta del pedido en la base de datos