

# **PRÁCTICA DE PROGRAMACIÓN ORIENTADA A OBJETOS JUNIO 2015**

GRADO EN INGENIERÍA EN  
TECNOLOGÍAS DE LA INFORMACIÓN

Francisco Javier Crespo Jiménez

e-mail: [fcrespo14@alumno.uned.es](mailto:fcrespo14@alumno.uned.es)  
Móvil: 687768000

|   |           |
|---|-----------|
| <b>1.- Análisis de la aplicación.....</b>                                 | <b>3</b>  |
| 1.1.- Introducción.....   | 3         |
| 1.2.- Diseño.....   | 3         |
| 1.3.- Descripción y justificación de las clases intervinientes.....       | 4         |
| <b>2.- Diagramas de clases.....</b>                                       | <b>17</b> |
| <b>3.- Manual de usuario de la aplicación por niveles de acabado.....</b> | <b>20</b> |
| <b>Anexo 1.- Código Fuente.....</b>                                       | <b>36</b> |

## 1.- Análisis de la aplicación.

### 1.1.- Introducción.

La práctica que se detalla en la presente memoria consiste en el desarrollo de un terminal punto de venta (TPV) utilizando el lenguaje de programación Java y con una metodología de Programación Orientada a Objetos.

Para su desarrollo se han tenido en cuenta las funcionalidades esperadas de un TPV desde labores simples de gestión de una venta, hasta operaciones más complejas como es la gestión de almacén o inventario, gestión de facturación o gestión de clientes.

**Se han cubierto todos los niveles de acabado**, especificados en el enunciado de la práctica, tal y como se desarrollará más adelante.

**NOTA:** Para compilar correctamente el proyecto en BlueJ, puede ser necesario incorporar la librería de validación de documentos. Para incorporar la librería desde BlueJ se hace desde el menú "Herramientas -> Preferencias -> Librerías". La librería se llama "valnif.jar" y se encuentra en el subdirectorio "+libs" dentro del proyecto tal y como indica la documentación del entorno BlueJ.  
[http://www.bluej.org/faq.html#faq\\_How\\_do\\_I\\_use\\_custom\\_class\\_libraries\\_JARs](http://www.bluej.org/faq.html#faq_How_do_I_use_custom_class_libraries_JARs)

### 1.2.- Diseño.

Para la mejor organización de las clases que componen el proyecto del TPV se ha llevado a cabo una separación en paquetes distinguiendo las responsabilidades de cada una de las clases. Así los paquetes que componen la solución son:

- **poouned.** Clases de inicio de la aplicación. Este es el paquete raíz que contiene el resto de paquetes.
- **Poouned.gui.** Paquete con los elementos que conforman el interfaz gráfico (GUI) de la aplicación.
- **poouned.modelos.** Paquete con las clases pertenecientes a la lógica de negocio.
- **poouned.modelos.tablemodel.** Paquete con las clases que hacen de adaptador entre el modelo y los objetos Jtable.
- **poouned.util.** Paquete que aglutina clases que no corresponden con ninguno de los otros paquetes. Las pequeñas utilidades y clases auxiliares quedan agrupadas dentro de este paquete.

Se han utilizado algunos patrones de diseño de software con el objetivo de disponer de mejor organización de código y una clara separación de responsabilidades.

Uno de los patrones utilizados ha sido el MVC (Modelo-Vista-Controlador). Una descripción detallada de dicho patrón puede extraerse del artículo de wikipedia:

<http://es.wikipedia.org/wiki/Modelo-vista-controlador>

Este patrón no fuerza a una organización concreta de clases sino que establece una separación funcional de los componentes aunque, como se podrá ver examinando el código fuente, no se ha llevado a cabo una implementación totalmente rigurosa. Se ha realizado una interpretación de dicho patrón donde al menos los objetos responsables de la lógica de negocio (modelo) tienen una clara separación del resto de objetos. No obstante, no se ha llevado a cabo una separación en clases distintas las responsabilidades del controlador y la vista.

- El usuario interactúa desde de la vista con los elementos gráficos del UI (User Interface).
- La propia vista implementa las acciones a ejecutar sin enviárselas a un controlador. La implementación se lleva a cabo integrando los correspondientes ActionListener en forma de clases anónimas dentro de los componentes gráficos que son manipulados. En nuestra aplicación, se trata principalmente de la pulsación de los correspondientes botones.
- En estos métodos se realiza las comprobaciones necesarias: selección del elemento en las tablas, la correcta introducción de los valores de entrada... y se lleva a cabo la manipulación de los datos ejecutando alguna acción pública del modelo. Se incluye un ejemplo del funcionamiento de un evento, en este caso de la eliminación de un cliente:

```
btnEliminar.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        if (JOptionPane.showConfirmDialog(null, "¿Confirma la  
            eliminación del cliente?", "WARNING",  
            JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION) {  
            try {  
                directorio.remove(tabla.convertRowIndexToModel(tabla.getSelectedRow()));  
            } catch (ArrayIndexOutOfBoundsException ex) {  
                JOptionPane.showMessageDialog(null, "No se ha seleccionado un cliente.",  
                    "Error en la eliminación.", JOptionPane.ERROR_MESSAGE);  
            } catch (IndexOutOfBoundsException ex) {  
                JOptionPane.showMessageDialog(null, "No se ha podido eliminar el cliente.",  
                    "Error en la eliminación.", JOptionPane.ERROR_MESSAGE);  
            }  
        }  
    }  
});
```

- Tras llevar a cabo la eliminación del cliente, el modelo notifica a las vistas que mantienen información de los clientes que ha cambiado la información. Para llevar a cabo se utiliza el patrón observador que será descrito a continuación.
- Finalmente, las vistas notificadas actualizan sus componentes gráficos mostrando la información actualizada del modelo.

Como se ha indicado, para llevar a cabo la interacción entre el modelo y la vista-controladora se ha utilizado un patrón de diseño muy utilizado en este tipo de entornos: el patrón Observador (Observer) [http://es.wikipedia.org/wiki/Observer\\_\(patrón\\_de\\_diseño\)](http://es.wikipedia.org/wiki/Observer_(patrón_de_diseño)).

Dicho patrón determina que existirá una dependencia entre objetos, de manera que cuando uno de los objetos cambia su estado, notifica este cambio a todos los objetos registrados como escuchadores (Listener) y estos determinarán que deben hacer con los cambios producidos. En nuestro caso, los objetos observables son las clases de modelo que mantienen la lógica de negocio. Los escuchadores registrados, son las interfaces gráficas que deben actualizar los datos por pantalla cuando se producen los cambios en los modelos. Este patrón de diseño permite que se mantenga la consistencia entre clases relacionadas, pero con un bajo acoplamiento.

### 1.3.- Breve descripción y justificación de las clases intervinientes.

Para cumplir con todos los requisitos planteados en el enunciado de la práctica se disponen las siguientes clases con sus métodos públicos que se enumeran y describen brevemente:

- **poouned.**

- **TPV.java** Clase inicial con el método main. Requerida por el enunciado como obligatoria.

```
public static void main(String[] args) { ... }
```

- **Controlador.java.** Clase controlador que hereda de JFrame; inicializa las estructuras de datos utilizadas por el resto de la aplicación y pasándolas a los constructores de los paneles del paquete poouned.gui. Establece también el menú principal de la aplicación con listener como clases anónimas.

```
public Controlador() { ... }
```

El resto de métodos del controlador, son privados por lo que no se incluyen en este documento.

- **poouned.gui.** Todas las clases de este paquete heredan de JPanel por lo que se omite en la descripción de cada una de ellas.

- **BuscarProductoPanel.java.** Clase que se utiliza para mostrar una ventana modal desde la caja del TPV para buscar un producto por su código o su descripción.

```
public BuscarProductoPanel(Inventario inventario, Ticket ticket) { ... }
```

- **CajaPanel.java.** Es el panel principal del TPV desde donde se hacen realmente las ventas de los productos a los clientes.

```
public CajaPanel(Inventario inventario, Ventas ventas, Directorio directorio) { ... }
```

```
public void update(Observable o, Object arg) { ... }
```

- **ClientesPanel.java.** Panel con la tabla del inventario donde se puede ver el listado de clientes y se llama a las acciones de crear nuevos clientes o eliminar alguno ya existente.

```
public ClientesPanel(Directorio directorio) { ... }
```

```
public void update(Observable o, Object arg) { ... }
```

- **EmpresaPanel.java.** Panel que crea una ventana modal para dar de alta un nuevo cliente de tipo empresa.

```
public EmpresaPanel(Directorio directorio) { ... }
```

- **FacturaPanel.java.** Panel con ventana modal para generar una nueva factura a partir de la selección de un cliente y un período fiscal.

```
public FacturaPanel(Facturas facturas, Directorio directorio, Ventas
ventas) { ... }
```

- **FacturasPanel.java.** Panel con listado de facturas generadas y posibilidad realizar acciones con las mismas tal como ver el detalle de una factura seleccionada.

```
public FacturasPanel(Directorio directorio, Ventas ventas, Facturas
facturas) { ... }
```

```
public void update(Observable o, Object arg) { ... }
```

- **ParticularPanel.java.** Panel con ventana modal que utilizado para dar de alta un nuevo cliente de tipo particular.

```
public ParticularPanel(Directorio directorio) { ... }
```

- **ProductoPanel.java.** Panel con ventana modal con un formulario para dar de alta un nuevo producto en el inventario.

```
public ProductoPanel(Inventario inventario) { ... }
```

- **ProductosPanel.java.** Panel con la ventana principal de gestión del inventario de productos.

```
public ProductosPanel(Inventario inventario) { ... }
```

- **VentasPanel.java.** Panel con listado de las ventas realizadas por el tpv.

```
public VentasPanel(Ventas ventas) { ... }
```

```
public void update(Observable o, Object arg) { ... }
```

- **VisorFacturaPanel.java.** Panel con elemento JTextArea para ver el cuerpo de la factura.

```
public VisorFacturaPanel(Factura factura) { ... }
```

- **VisorListadosPanel.java.** Panel con elemento JTextArea para la visualización de los listados requeridos.

```
public VisorListadosPanel(String titulo, String cuerpo) { ... }
```

- **poouned.modelos.**

- **AbstractModel.java.** Esta clase abstracta, es la clase padre de todas las clases del paquete poouned.modelos. Hereda de Observable, sobrescribe e implementa el método setChanged() de manera que las clases hijas serán observables y llamarán al método setChanged() de esta clase para indicar que su datos han cambiado y notificarán este cambio a sus observadores.
- **AbstractListModel.java.** Esta clase abstracta hereda de la anterior. Implementa el interfaz Serializable y un método llamado exportar() de manera que las clases modelo que contengan listas y estas sean exportables o serializables, heredarán de esta.
- **Cliente.java.** De nuevo nos encontramos con una clase abstracta que contiene los atributos comunes de los clientes para luego ser especializados en 2 clases hija dependiendo del tipo de cliente a tratar: particular o empresa.

```
public abstract String getId();

public abstract void setId(String id);

public abstract String getDenominacion();

public abstract void setDenominacion(String denominacion);

public UUID getCodigo() { ... }

public void setCodigo() { ... }

public void setCodigo(UUID codigo) { ... }

public String getDomicilio() { ... }

public void setDomicilio(String domicilio) { ... }

public Date getFechaAlta() { ... }

public void setFechaAlta() { ... }

public void setFechaAlta(Date fechaAlta) { ... }

public String toString() { ... }
```

Para poder ser trados ambos tipos de clientes de manera análoga se crean métodos abstractos como getId() o getDenominacion() que las clases hijas implementarán de la manera correcta para cada tipo de cliente. Por ejemplo, el método getId() será sobrescrito para los particulares obteniendo el nif mientras que para las empresas retornará el cif. Haciendo uso del poliformismo proporcionado por Java.

- **Directorio.java.** Esta clase representa el conjunto de clientes de la aplicación. Internamente utiliza un ArrayList y dispone de métodos para comprobar si el cliente ya ha sido introducido en la lista.

```
public Directorio() { ... }

public Directorio(ArrayList<Cliente> clientes) { ... }
```



```
public boolean add(Cliente cliente) throws IllegalArgumentException { ... }  
public boolean addAll(ArrayList<Cliente> clientes) throws Exception { ... }  
public Cliente get(int index) { ... }  
public ArrayList<Cliente> getAll() { ... }  
public Cliente findById(String id) { ... }  
public boolean importar() throws Exception { ... }  
public boolean exportar() { ... }  
public boolean remove(Cliente cliente) { ... }  
public Cliente remove(int index) { ... }  
public int size() { ... }  
public String toString() { ... }
```

- **Empresa.java.** Hereda de la clase abstracta cliente y añade los atributos cif y razonSocial que son específicos del cliente de tipo empresa. Implementa los métodos abstractos de su superclase getId y getDenominacion.

```
public Empresa() { ... }  
  
public Empresa(UUID codigo, Date fechaAlta, String domicilio, String  
razonSocial, String cif) { ... }  
  
public String getId() { ... }  
  
public String getDenominacion() { ... }  
  
public String getCif() { ... }  
  
public void setCif(String cif) { ... }  
  
public String getRazonSocial() { ... }  
  
public void setRazonSocial(String razonSocial) { ... }  
  
public void setId(String id) { ... }  
  
public void setDenominacion(String denominacion) { ... }  
  
public String toString() { ... }
```

- **Factura.java.** Por la definición proporcionada en el enunciado, una factura se compone de un conjunto de tickets o ventas realizadas a un mismo cliente durante un periodo fiscal concreto. Internamente, almacena los tickets en un ArrayList.

```
public Factura() { ... }

public Factura (UUID numeroFactura, String cif, String razonSocial, Date
fecha, Cliente cliente, String periodo) { ... }

public UUID getNumeroFactura() { ... }

public String getCif() { ... }

public void setCif(String cif) { ... }

public String getRazonSocial() { ... }

public void setRazonSocial(String razonSocial) { ... }

public Date getFecha() { ... }

public void setFecha(Date fecha) { ... }

public Cliente getCliente() { ... }

public void setCliente(Cliente cliente) { ... }

public boolean addTicket(Ticket ticket) { ... }

public void setTickets(ArrayList<Ticket> tickets) { ... }

public boolean removeTicket(Ticket ticket) { ... }

public void clearTickets() { ... }

public ArrayList<ProductoVendido> getProductos() { ... }

public String getPeriodoFiscal() { ... }

public void setPeriodoFiscal(String periodo) { ... }

public double getImporte() { ... }

public String toString() { ... }
```

- **Facturas.java.** Lista de facturas. Internamente se almacenan en un ArrayList. Esta clase hereda de AbstractListModel de manera que se permite su exportación hacia archivos.

```
public Facturas() { ... }

public Facturas(ArrayList<Factura> facturas) { ... }

public boolean add(Factura factura) throws Exception { ... }

public boolean addAll(ArrayList<Factura> facturas) throws Exception
{ ... }

public Factura findByNumeroFactura(UUID numeroFactura) { ... }
```

```
public Factura get(int index) { ... }  
public ArrayList<Factura> getAll() { ... }  
public boolean remove(Factura factura) { ... }  
public Factura remove(int index) { ... }  
public int size() { ... }  
public boolean importar() throws Exception { ... }  
public boolean exportar() { ... }  
public String toString() { ... }
```

- **Inventario.java.** Clase que mantiene un ArrayList con todos los productos que forman parte del inventario del tpv. Dispone de métodos para comprobar si el producto ya existe, buscar un producto por su código o descripción y al heredar de AbstractListModel, permite se exportado hacia archivo.

```
public Inventario() { ... }  
public Inventario(ArrayList<Producto> productos) { ... }  
public boolean add(Producto producto) throws Exception { ... }  
public boolean addAll(ArrayList<Producto> productos) throws Exception  
{ ... }  
public Producto get(int index) { ... }  
public Producto findByCodigo(String codigo) { ... }  
public Producto findByDescripcion(String descripcion) { ... }  
public boolean remove(Producto producto) { ... }  
public Producto remove(int index) { ... }  
public int size() { ... }  
public boolean importar() throws Exception { ... }  
public boolean exportar() { ... }  
public String toString() { ... }
```

- **Particular.java.** Clase que hereda de la clase abstracta cliente y que añade los atributos nif, nombre, apellidos así como implementar los métodos abstractos getID() o getDenominacion().

```
public Particular() { ... }
```

```
public Particular(UUID codigo, Date fechaAlta, String domicilio, String
nif, String nombre, String apellidos) { ... }

public String getId() { ... }

public String getDenominacion() { ... }

public String getNif() { ... }

public void setNif(String nif) { ... }

public String getNombre() { ... }

public void setNombre(String nombre) { ... }

public String getApellidos() { ... }

public void setApellidos(String apellidos) { ... }

public void setId(String id) { ... }

public void setDenominacion(String denominacion) { ... }

public String toString() { ... }
```

- **Producto.java.** A la hora de decidir el esquema de clases intervinientes en la aplicación se decide crear una clase producto abstracta que implemente los atributos y métodos comunes a todos los productos (código, descripción, precio sin iva, tipos de iva...) sin embargo se deja abstracta para que sean ProductoInventariado y ProductoVendido las clases que concreten los detalles específicos de cada tipo de producto.

```
public void setCodigo(String codigo) { ... }

public String getCodigo() { ... }

public void setDescripcion(String descripcion) { ... }

public String getDescripcion() { ... }

public void setPrecio0(double precio) { ... }

public double getPrecio0() { ... }

public void setIva(int iva) { ... }

public int getIva() { ... }

public double getPrecio() { ... }
```

La decisión detrás de esta división se encuentra en que un producto inventariado mantiene un número de unidades en stock, mientras que un producto que ya ha sido

vendido no mantiene ningún tipo de información sobre el stockaje del mismo y, por contra, almacena información sobre cuantas unidades del mismo han sido vendidas al mismo tiempo y además añade información sobre el ticket en el cual ha sido vendido.

- **ProductoInventariado.java.** Clase que hereda de la clase abstracta producto, añade el atributo stock y sus getter-setter correspondientes.

```
public ProductoInventariado(String codigo, String descripcion, double
precio0) { ... }

public ProductoInventariado(String codigo, String descripcion, double
precio0, int stock) { ... }

public ProductoInventariado(String codigo, String descripcion, double
precio0, int stock, int iva) { ... }

public void setStock(int stock) { ... }

public int getStock() { ... }

public String toString() { ... }
```

- **ProductoVendido.java.** Esta clase hereda de la clase abstracta producto añadiendo los atributos unidades y ticket para mantener la información sobre las unidades que se vendieron del mismo y el ticket en el que se realizó la venta. Y los correspondientes getter-setter.

```
public ProductoVendido(String codigo, String descripcion, double precio0,
int iva, int unidades) { ... }

public void setUnidades(int unidades) { ... }

public int getUnidades() { ... }

public Ticket getTicket() { ... }

public void setTicket(Ticket ticket) { ... }
```

- **Ticket.java.** Clase que representa una venta del tpv. Mantiene los atributos fecha, cliente, y un ArrayList de ProductoVendido que conforman una venta.

```
public Ticket() { ... }

public String getCodigo() { ... }

public Date getFecha() { ... }

public void setFecha(Date fecha) { ... }

public void setCliente(Cliente cliente) { ... }

public Cliente getCliente() { ... }

public double getImporte() { ... }
```

```
public String getPeriodoFiscal() { ... }

public boolean getFacturado() { ... }

public void setFacturado(boolean valor) { ... }

public void facturar() { ... }

public ArrayList<ProductoVendido> getProductos() { ... }

public void addProducto(ProductoInventariado producto, int unidades)
throws QuantityWrongException { ... }

public String toString() { ... }
```

- **Ventas.java.** Clase que mantiene un ArrayList de tickets. Hereda de AbstractListModel de modo que tiene la posibilidad de ser exportado hacia un archivo.

```
public Ventas(Inventario inventario) { ... }

public Ventas(ArrayList<Ticket> tickets) { ... }

public boolean add(Ticket ticket) throws Exception { ... }

public boolean addAll(ArrayList<Ticket> tickets) throws Exception { ... }

public Ticket findByCodigo(String codigo) { ... }

public Ticket get(int index) { ... }

public ArrayList<Ticket> getAll() { ... }

public ArrayList<Ticket> getFacturables(Cliente cliente, String periodo)
{ ... }

public ArrayList<Cliente> getClientes() { ... }

private Cliente findById(ArrayList<Cliente> clientes, String id) { ... }

public ArrayList<Ticket> getIntervalo(Date fechaInicial, Date fechaFinal)
{ ... }

public ArrayList<Ticket> getIntervaloCliente(Date fechaInicial, Date
fechaFinal, Cliente cliente) { ... }

public boolean remove(Ticket ticket) { ... }

public Ticket remove(int index) { ... }

public void devolucion(int index) throws IllegalArgumentException { ... }

public int size() { ... }

public boolean importar() throws Exception { ... }
```

```
public boolean exportar() { ... }

public String toString() { ... }
```

Dispone de métodos para añadir tickets comprobando que no exista previamente, buscar tickets por su código, obtener los tickets facturables, y algunos métodos que resultan útiles para implementar los listados requeridos en el enunciado: `getClientes()`, `getIntervalo()`, `getIntervaloCliente()`.

- **poouned.modelos.tablemodel.** Tal y como se ha indicado previamente, este paquete tiene las clases que actúan de adaptador entre los modelos y los objetos `Jtable`. Heredan de `DefaultTableModel`. De manera común para todos, establecen los títulos de las columnas, devuelven el número de filas y columnas de que disponen, a través del método `getValueAt(int fila, int columna)` son capaces de obtener el valor apropiado para esa combinación de fila, columna y de manera análoga el método `setValueAt(int fila, int columna)` establece el valor para esa casilla. Dependiendo del tipo de tabla concreto, el método `isCellEditable()` comprueba si se pueden editar las columnas.

- **CientesTM.java.** `TableModel` que se utiliza para adaptar el modelo cliente a `Jtable`.

```
public CientesTM(Directorio clientes) { ... }

public int getColumnCount() { ... }

public int getRowCount() { ... }

public String getColumnName(int col) { ... }

public Object getValueAt(int fila, int columna) { ... }

public boolean isCellEditable(int fila, int columna) { ... }

public void setValueAt(Object valor, int fila, int columna) { ... }

public Class<?> getColumnClass(int columnIndex) { ... }
```

- **FacturasTM.java.** Utilizado para adaptar el modelo de facturas y poder ser enlazado con un `Jtable`.

```
public FacturasTM(Facturas facturas) { ... }

public int getColumnCount() { ... }

public int getRowCount() { ... }

public String getColumnName(int col) { ... }

public Object getValueAt(int fila, int columna) { ... }

public boolean isCellEditable(int fila, int columna) { ... }

public Class<?> getColumnClass(int columnIndex) { ... }
```

- **ProductosTM.java.** Usado para enlazar el modelo producto con su correspondiente tabla Jtable.

```
public ProductosTM(Inventario productos) { ... }

public int getColumnCount() { ... }

public int getRowCount() { ... }

public String getColumnName(int col) { ... }

public Object getValueAt(int fila, int columna) { ... }

public boolean isCellEditable(int fila, int columna) { ... }

public void setValueAt(Object valor, int fila, int columna) { ... }

public Class<?> getColumnClass(int columnIndex) { ... }
```

- **ProductosROTM.java.** El aspecto más destacable de esta clase, sería que hereda de la anterior ProductosTM.java y sobrescribe el método isCellEditable(int fila, int columna) forzando a que devuelva un valor false en cualquier caso. Es un ejemplo de utilización de herencia para "especializar" una clase sobrescribiendo solamente el método necesario y conservando el resto de funcionalidad intacta. El motivo de la creación de esta clase es utilizar el tableModel de productos pero no permitir la modificación de los mismos directamente. Se utiliza en la lista de productos de TPV a la hora de realizar la venta.

```
public ProductosROTM(Inventario productos) { ... }

public boolean isCellEditable(int fila, int columna) { ... }
```

- **ProductosVendidosTM.java.** TableModel que se usa para enlazar el ticket actual de la venta, con el ArrayList de ProductoVendido.

```
public ProductosVendidosTM(ArrayList<ProductoVendido> ventas) { ... }

public int getColumnCount() { ... }

public int getRowCount() { ... }

public String getColumnName(int col) { ... }

public Object getValueAt(int fila, int columna) { ... }

public boolean isCellEditable(int fila, int columna) { ... }

public Class<?> getColumnClass(int columnIndex) { ... }
```

- **TicketTM.java.** Usado para adaptar el modelo de Ticket en el JTable que muestra la lista de ventas.

```
public TicketTM(Ventas ventas) { ... }
```



```
public int getColumnCount() { ... }

public int getRowCount() { ... }

public String getColumnName(int col) { ... }

public Object getValueAt(int fila, int columna) { ... }

public boolean isCellEditable(int fila, int columna) { ... }

public Class<?> getColumnClass(int columnIndex) { ... }
```

- **pooned.util.**

- **Listados.java.** Clase con métodos útiles para mostrar los 3 listados requeridos por el enunciado de la práctica:

```
public String getIntervaloGroupByClientes(Date fechaInicial, Date
fechaFinal) { ... }

public String getIntervaloCliente(Date fechaInicial, Date fechaFinal,
Cliente cliente) { ... }

public String rankingProductosIntervalo(Date fechaInicial, Date
fechaFinal) { ... }
```

- **Numericas.java.** Método de redondeo de los precios de los productos.

```
public static double redondeo(double val) {
    return Math.round(val * 100.0) / 100.0;
}
```

- **Persistible.java.** Interfaz que define los métodos que deberán implementar las clases creadoras de métodos de exportación. Se ha buscado este modelo como ejemplo de patrón de diseño abierto-cerrado. Es decir, se deja cerrado ya que las clases que implementen este interfaz tiene que cumplir los métodos definidos en el mismo pero abierto a la extensión. Si en el futuro se quisiera implementar un "PersistidorRed.java", "PersistidorBD.java" por ejemplo, bastaría con volver a implementar los métodos importar y exportar sin importar como lo haga internamente.

```
public void exportar(Object objeto, String destino) throws Exception
{ ... }

public ArrayList<Object> importar(String origen) throws Exception { ... }
```

- **PersistidorArchivo.java.** Implementación concreta del interfaz Persistible de manera que se puedan realizar las operaciones de importación y exportación hacia y desde archivo respectivamente.

```
public void exportar(Object objeto, String archivo) throws Exception {...}

public ArrayList<Object> importar(String archivo) throws Exception { ... }
```

- **QuantityWrongException.java.** Clase que define un nuevo tipo de excepción personalizada en caso de incluir una cantidad de artículos en una venta superior a lo existente en el stock.

```
public QuantityWrongException(String msg) { ... }
```

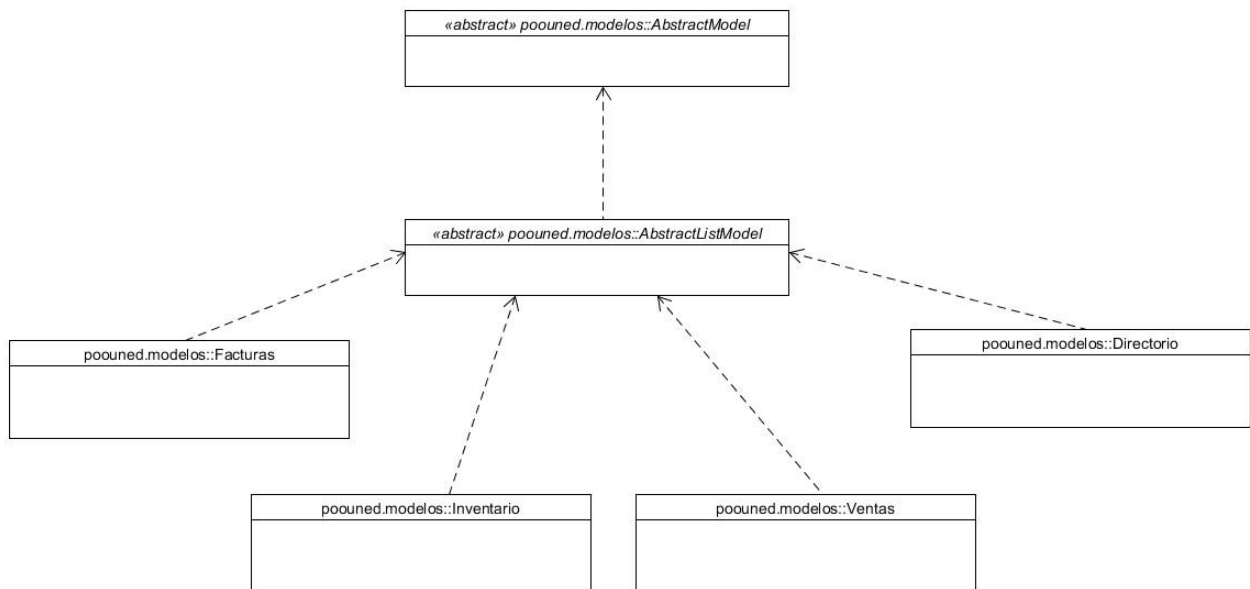
Se implementa prácticamente a modo de demostración de esta técnica ya que se podría haber hecho uso de algunas excepciones ya existentes en Java como `IllegalArgumentException` o `InvalidParameterException`.

- **Texto.java.** Clase con utilidades comunes para formatear texto adaptándolo a nuestras necesidades concretas. Se define un formato decimal de 5 dígitos y 2 decimales y una función para truncar textos limitándolos a 24 caracteres.

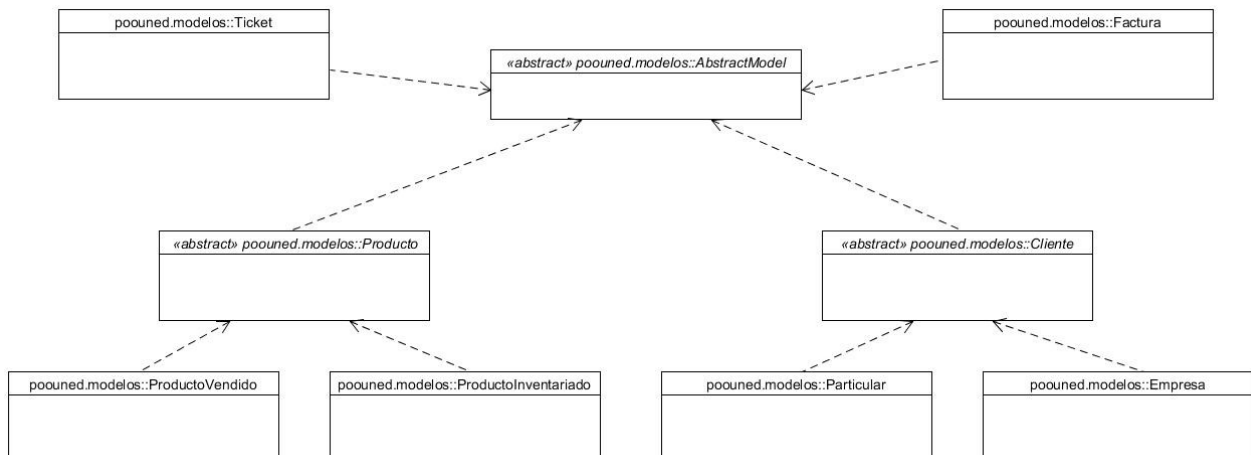
```
public static final DecimalFormat FORMATO_DECIMAL = new  
DecimalFormat("####0.##");  
  
private static int TAMANIO_MAX = 24;
```

## 2.- Diagramas de clases.

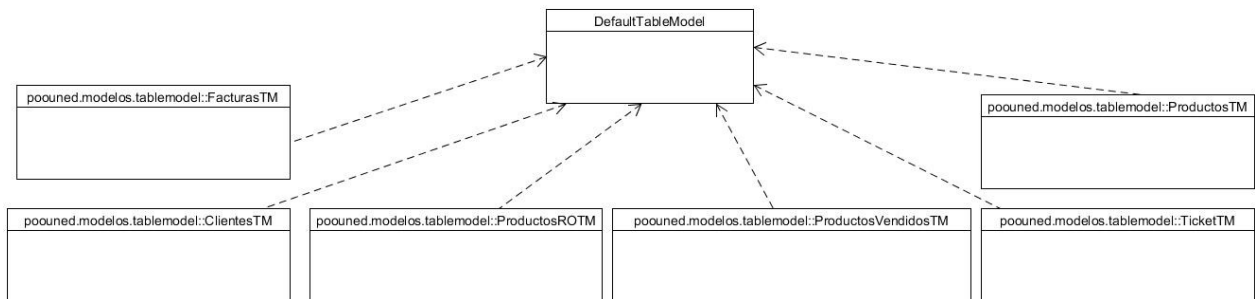
### Paquete poouned.modelos - Clases que herendan de AbstractListModel



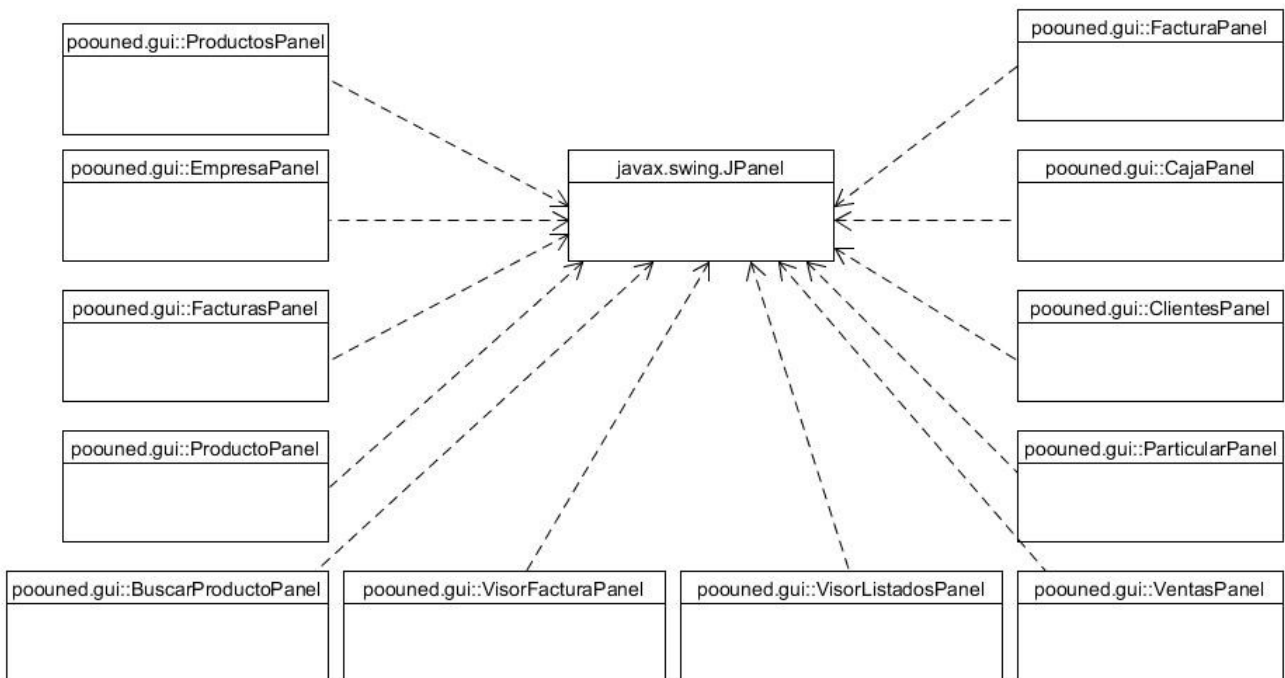
### Paquete poouned.modelos – Clases que heredan de AbstractModel



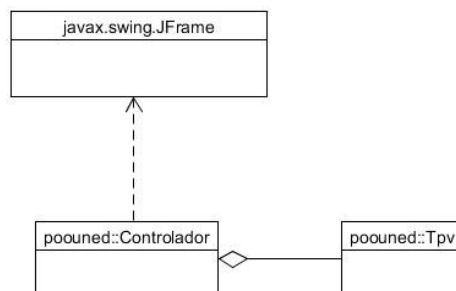
### Paquete poouned.modelos.tablemodel – Clases que heredan de DefaultTableModel



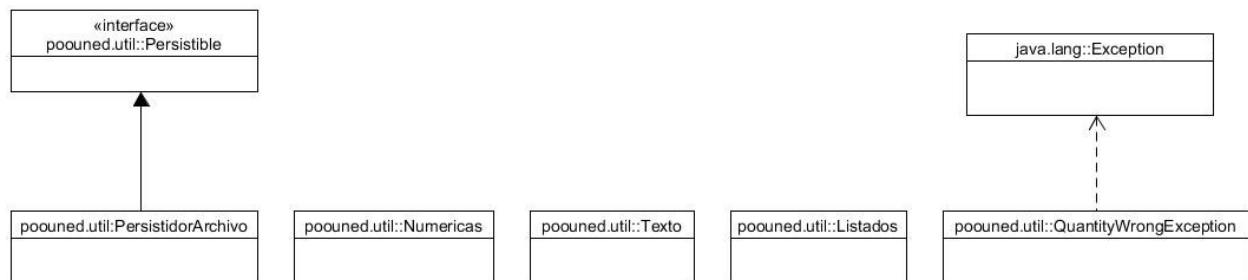
### Paquete poouned.gui – Clases que heredan de JPanel



### Paquete poouned



### Paquete pooned.util



### 3.- Manual de usuario de la aplicación por niveles de acabado.

**Nivel 1: Planteamiento del Problema: actores participantes, relaciones entre actores, funcionalidad a cumplir por la práctica a desarrollar. Establecimiento de diferentes clases a intervenir, relaciones de dependencia.**

Se plantea el desarrollo de un sistema Terminal Punto de Venta con distintos niveles de acabado para cumplir con las funcionalidades descritas en cada nivel. En esta práctica, se han cubierto todos los niveles propuestos tal y como se detalla en cada uno de los siguientes apartados que describen la funcionalidad por niveles.

Los actores intervinientes en la operativa del TPV son 2 claramente identificados: vendedor y comprador. El vendedor realiza toda la operativa del TPV mientras que el comprador no tiene ningún tipo de interacción con la aplicación.

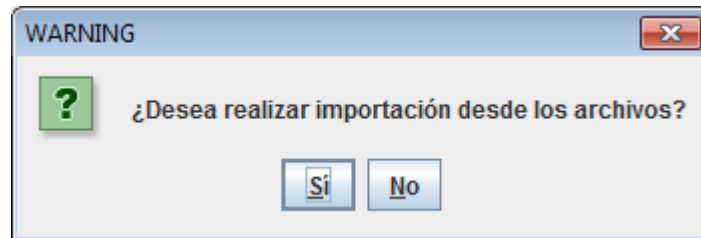
Las funcionalidades cubiertas por la aplicación son:

- **Productos**
  - Alta.
  - Modificación.
  - Eliminación.
  - Importación y exportación de y hacia disco.
- **Clientes**
  - Alta.
  - Modificación.
  - Eliminación.
  - Importación y exportación de y hacia disco.
- **Caja.**
  - Venta de productos por TPV con posibilidad de búsqueda por código o descripción.
  - Listado de ventas.
  - Devoluciones de productos no facturados con reintegro al stock.
  - Facturación de ventas en un determinado período fiscal.
  - Visualización del detalle de las facturas.
  - Eliminación de facturas.
- **Listados.**
  - Ventas agrupadas por clientes.
  - Ventas a un cliente.
  - Ranking de productos más vendidos durante un periodo de tiempo.

El diseño de la aplicación así como la justificación de las clases y componentes necesarios han sido cubiertos en los puntos 1 y 2 de esta memoria de manera que quedan cubiertos los requisitos del nivel 1 de acabado.

A continuación se describe el procedimiento de inicio de la aplicación así como el comienzo de una sesión de trabajo.

Al iniciar la aplicación, lo primero que aparece es una ventana donde se nos pregunta si queremos realizar la importación desde los archivos.

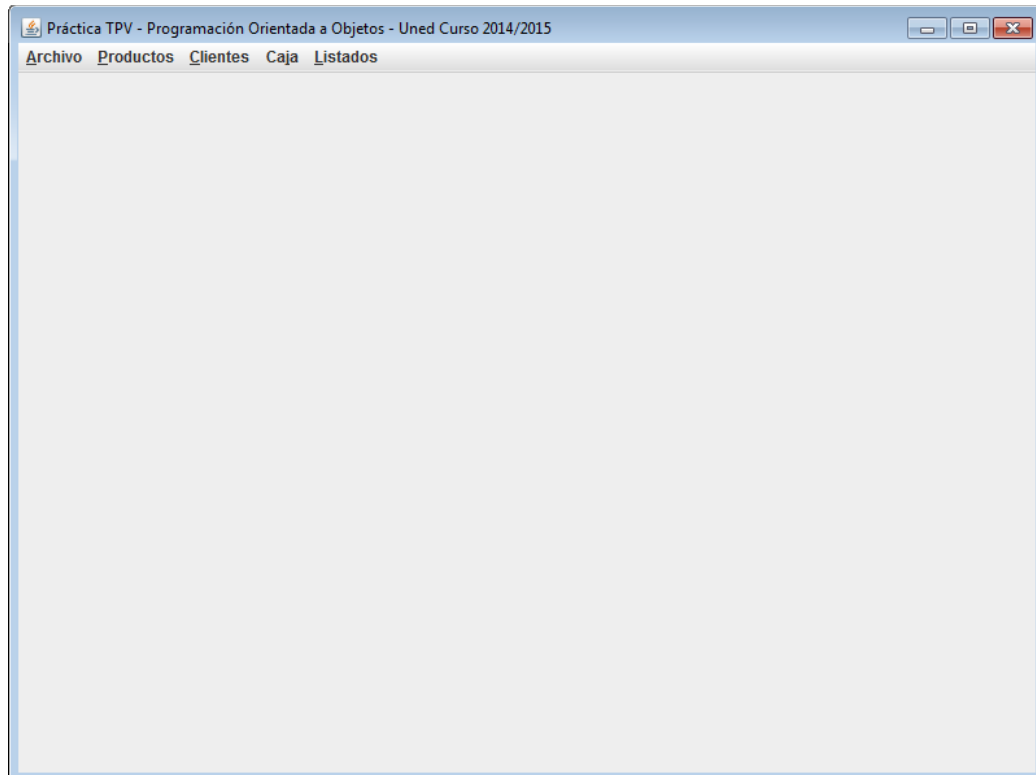


En caso de seleccionar "No" el programa comienza sin cargar ningún tipo de dato. Si seleccionamos la opción "Si" se importan los datos existentes en los archivos correspondientes:

- directorio.dat
- facturas.dat
- inventario.dat
- ventas.dat

Para poder ser importados, estos archivos deben existir en el mismo directorio donde se encuentre el ejecutable de la aplicación. Se mostrará un error en caso de no existir alguno de los mencionados archivos o en caso de existir el elemento a importar (personas, productos...).

En cualquiera de los casos, el programa mostrará la pantalla principal



Las opciones del menú están dispuestas en el orden que se ha considerado más lógico de la operativa del tpv, es decir, no tiene sentido hacer una venta en caja si previamente no se han dado de alta productos y clientes.

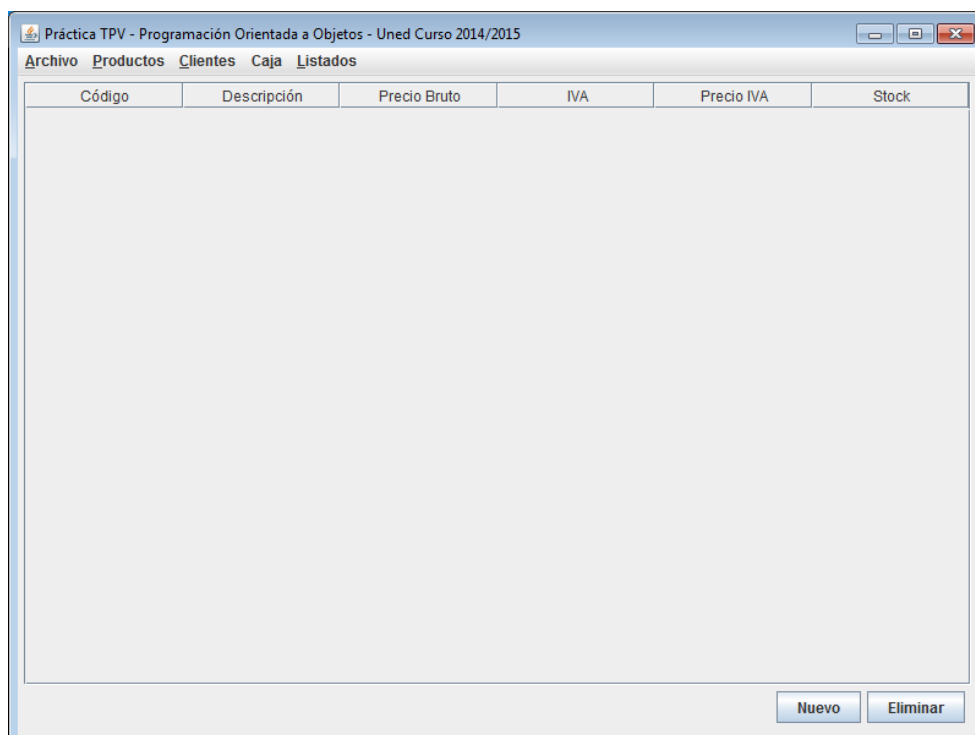
**Menú Archivo.** Con las opciones para importar-exportar y salir de la aplicación. Cuando se sale de la aplicación se realiza una exportación completa para obtener un "snapshot" del estado del tpv en la próxima sesión, siempre que no se indique que no se desea importar los datos desde los archivos. Importar y Exportar le permite efectuar estas operaciones directamente. Por ejemplo, nos permite exportar los datos del programa sin tener que salir del mismo o importar los datos añadiéndose a los ya existentes en memoria. Hay que indicar que si el dato que se pretende importar ya existiese en el tpv, se indicará con un error de importación para que el usuario tenga constancia y se obviará el registro correspondiente.

## Nivel 2: Gestión de inventario del establecimiento comercial.

Este nivel de acabado implementa la parte de gestión de inventario del establecimiento comercial. Se accede desde el Menú "Productos". Desde este menú muestra la pantalla principal de mantenimiento de productos.

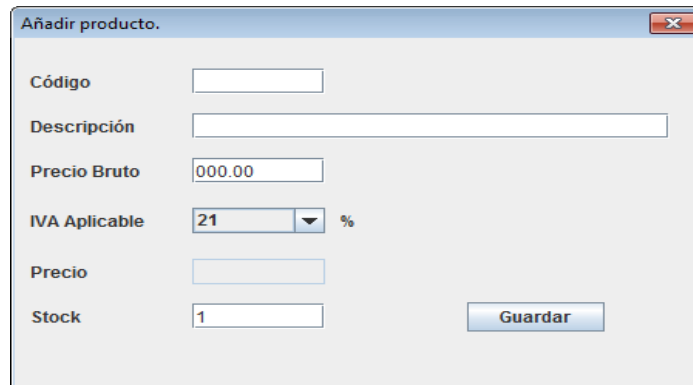
Permite llevar un control de los elementos que existen en nuestro establecimiento. Tal y como se indica en el enunciado los están identificados en el sistema por, los siguientes datos: código descriptivo, descripción, precio unitario sin IVA (Precio Bruto), IVA aplicable, precio unitario con IVA, cantidad disponible en stock.

- El sistema permite dar de alta nuevos productos, dar de baja productos existentes así como modificar los datos del mismo.
- Se permite realizar la importación y/o exportación de los productos a/desde ficheros.





Para **crear** un nuevo producto se pulsará sobre el botón "Nuevo" el cual nos mostrará una ventana donde se solicitan los datos a introducir:



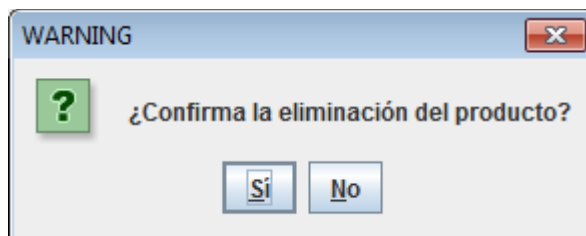
La imagen muestra una ventana de diálogo titulada "Añadir producto." con un botón de cerrar en la esquina superior derecha. Dentro de la ventana, hay seis campos de entrada y un botón "Guardar". Los campos son: "Código" (un campo de texto corto), "Descripción" (un campo de texto largo), "Precio Bruto" (un campo de texto con "000.00" pre-llenado), "IVA Aplicable" (un menú desplegable con "21" seleccionado y un símbolo de porcentaje a la derecha), "Precio" (un campo de texto corto) y "Stock" (un campo de texto con "1" pre-llenado). El botón "Guardar" está situado a la derecha de los campos "Precio" y "Stock".

- Código: Un código único para el producto.
- Descripción: Una descripción detallada del mismo.
- Precio Bruto: El precio sin IVA del producto. Importante notar que el separador de decimales es el punto.
- IVA: Dependiendo del tipo de producto, tendrá un tipo de IVA aplicable.
- Stock: Número de unidades a incluir en el stock.
- El precio final se calcula de manera automática en función del precio bruto y el tipo de IVA aplicable.

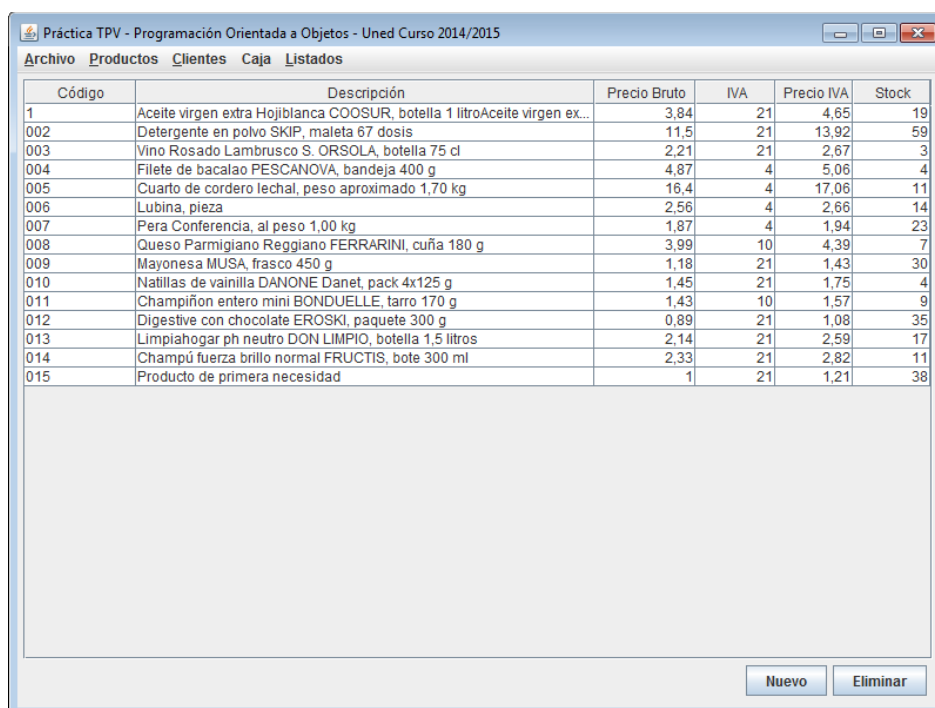
Cuando se pulsa el botón "Guardar" el producto se incorpora al stock y la ventana limpia sus campos para introducir un nuevo producto.

Para **modificar** los datos del producto, pensando en una edición sencilla, se puede hacer directamente sobre la tabla haciendo doble click sobre el dato a modificar. Se considera que el código del producto actúa como si fuera una clave primaria y por tanto es inmutable. En caso de necesitar modificarla se haría necesario eliminar y volver a crear el producto.

Para **eliminar** un producto del inventario, basta con seleccionarlo en la tabla del inventario y pulsar sobre el botón "Eliminar". Se mostrará una ventana como la siguiente pidiendo confirmación de la eliminación para evitar borrados accidentales.



A continuación se muestra una pantalla de ejemplo con algunos artículos introducidos.



| Código | Descripción  | Precio Bruto | IVA | Precio IVA | Stock |
|--------|--|--------------|-----|------------|-------|
| 1      | Aceite virgen extra Hojiblanca COOSUR, botella 1 litro | 3,84         | 21  | 4,65       | 19    |
| 002    | Detergente en polvo SKIP, maleta 67 dosis              | 11,5         | 21  | 13,92      | 59    |
| 003    | Vino Rosado Lambrusco S. ORSOLA, botella 75 cl         | 2,21         | 21  | 2,67       | 3     |
| 004    | Filete de bacalao PESCANOVA, bandeja 400 g             | 4,87         | 4   | 5,06       | 4     |
| 005    | Cuarto de cordero lechal, peso aproximado 1,70 kg      | 16,4         | 4   | 17,06      | 11    |
| 006    | Lubina, pieza  | 2,56         | 4   | 2,66       | 14    |
| 007    | Pera Conferencia, al peso 1,00 kg                      | 1,87         | 4   | 1,94       | 23    |
| 008    | Queso Parmigiano Reggiano FERRARINI, cuña 180 g        | 3,99         | 10  | 4,39       | 7     |
| 009    | Mayonesa MUSA, frasco 450 g                            | 1,18         | 21  | 1,43       | 30    |
| 010    | Natillas de vainilla DANONE Danet, pack 4x125 g        | 1,45         | 21  | 1,75       | 4     |
| 011    | Champiñon entero mini BONDUELLE, tarro 170 g           | 1,43         | 10  | 1,57       | 9     |
| 012    | Digestive con chocolate EROSKI, paquete 300 g          | 0,89         | 21  | 1,08       | 35    |
| 013    | Limpiahogar ph neutro DON LIMPIO, botella 1,5 litros   | 2,14         | 21  | 2,59       | 17    |
| 014    | Champú fuerza brillo normal FRUCTIS, bote 300 ml       | 2,33         | 21  | 2,82       | 11    |
| 015    | Producto de primera necesidad                          | 1            | 21  | 1,21       | 38    |

### Nivel 3: Gestión de la venta del establecimiento comercial.

En este nivel de acabado las funcionalidades añadidas son las siguiente:

- Llevar un control de las diferentes ventas que se producen. Así, el sistema lleva un control de tickets generados, de modo que cada ticket se considerará una venta. En esta práctica se considersn análogos ambos términos. Cada ticket tiene un código de identificador único con formato: AAAAMMDDHHMMSS.
- La venta consiste en la inclusión de varios productos en una lista, generándose una línea por cada producto vendido. Cada línea muestra, el código del producto, la descripción del producto, la cantidad de unidades vendidas, el precio unitario con IVA, el IVA que se le aplica y el importe total de la venta de ese producto según el número de unidades vendidas.
- El proceso de venta implica automáticamente un proceso de actualización del inventario tal y como se ha definido en el Nivel 2. De este modo, si se introduce un código que no pertenece a ningún producto, o si se introduce un producto que no existe en stock (o más unidades de las existentes), el programa deberá muestra los errores correspondientes.
- El sistema deberá permite también introducir un producto a vender en el ticket haciendo una búsqueda por la descripción, además de con el código que lo identifica.
- Se permite la importación y/o exportación de los diferentes tickets de ventas a/desde ficheros.

Para acceder a estas funcionalidades debemos entrar a través del menú "Caja". Hay que hacer constar que para comenzar a realizar ventas, debemos tener previamente introducidos datos tanto de productos como de clientes.

**TPV.** Desde esta ventana se realiza la venta de los productos a los clientes del sistema. Se ha realizado un diseño de ventana pensando en la venta de manera rápida y ágil.

De manera predeterminada nada más abrir la ventana existe un ticket abierto lo que quiere decir que el sistema se encuentra preparado para introducir artículos a vender directamente. En la parte superior de la ventana, se encuentra una lista desplegable para seleccionar el cliente al que se realizará la venta. Si el comercio va a permitir realizar la venta a clientes sin identificar, se recomienda la creación en primer lugar de un cliente genérico al que asignar estas ventas. En cualquier caso, con el cliente seleccionado, tan sólo con seleccionar el artículo elegido en la tabla de la izquierda y pulsando el botón "Añadir ->" se añadirá al ticket en curso. En caso de que la venta sea de más de una unidad, tan sólo se debe introducir la cantidad en la casilla correspondiente. Por defecto, está ajustada a 1 unidad.

En la tabla de la izquierda no se puede modificar ningún dato de los productos. Tan solo se muestran los datos para tener una mayor información. Recordamos que el mantenimiento de los productos debe realizarse desde su menú correspondiente.

Práctica TPV - Programación Orientada a Objetos - Uned Curso 2014/2015

Archivo Productos Clientes Caja Listados

12345678Z - Aarón Burrell Cabañas

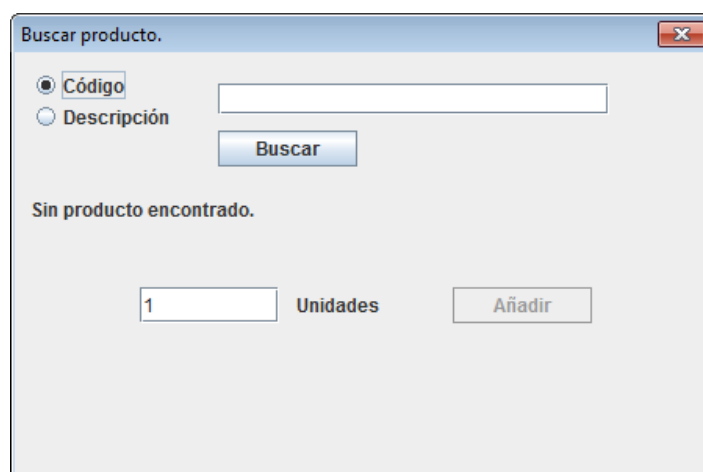
| Código | Descripción             | Precio Bruto | IVA | Precio IVA | Stock |
|--------|-------------------------|--------------|-----|------------|-------|
| 1      | Aceite virgen extra ... | 3,84         | 21  | 4,65       | 19    |
| 002    | Detergente en pol...    | 11,5         | 21  | 13,92      | 59    |
| 003    | Vino Rosado Lam...      | 2,21         | 21  | 2,67       | 3     |
| 004    | Filete de bacalao ...   | 4,87         | 4   | 5,06       | 4     |
| 005    | Cuarto de cordero ...   | 16,4         | 4   | 17,06      | 11    |
| 006    | Lubina, pieza           | 2,56         | 4   | 2,66       | 14    |
| 007    | Pera Conferencia, ...   | 1,87         | 4   | 1,94       | 23    |
| 008    | Queso Parmigian...      | 3,99         | 10  | 4,39       | 7     |
| 009    | Mayonesa MUSA, f...     | 1,18         | 21  | 1,43       | 30    |
| 010    | Natillas de vainilla... | 1,45         | 21  | 1,75       | 4     |
| 011    | Champiñon enter...      | 1,43         | 10  | 1,57       | 7     |
| 012    | Digestive con cho...    | 0,89         | 21  | 1,08       | 35    |
| 013    | Limpiahogar ph n...     | 2,14         | 21  | 2,59       | 17    |
| 014    | Champú fuerza bri...    | 2,33         | 21  | 2,82       | 11    |
| 015    | Producto de prime...    | 1            | 21  | 1,21       | 38    |

| Código | Descripción          | unidades | Precio Unit... | IVA | Importe |
|--------|----------------------|----------|----------------|-----|---------|
| 011    | Champiñon entero ... | 2        | 1,73           | 10  | 3,46    |

Buscar 2 Unidades Añadir -> Finalizar

Por otro lado, como puede observarse, cada producto que se incorpora al ticket en curso realiza la correspondiente actualización del stock del mismo en el inventario. No se permite la inclusión de una cantidad de productos superior al stock existente en ese momento.

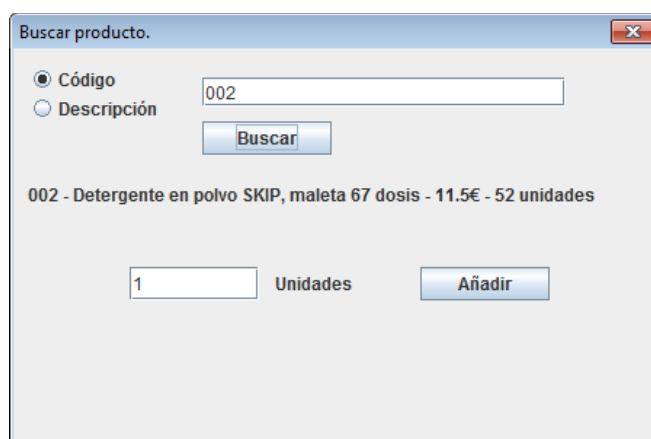
Existe además un botón para buscar los productos en lugar de seleccionarlos de la tabla.



Se puede realizar la búsqueda tanto por el código del producto como por los caracteres iniciales de su descripción de manera que no es imprescindible la escritura de la descripción completa. En el caso del código si que es necesario escribirlo literalmente. Si los caracteres introducidos de la descripción correspondiese con más de un producto, aparecerá el primero de ellos. Por ejemplo, si en el inventario tenemos 2 productos: "Botella de Aceite..." y "Botella de Refresco..."

Si buscamos por descripción la palabra "Botella" aparecería el producto "Botella de Aceite".

Una vez localizado el producto, se activa el botón "Añadir" que por defecto aparece deshabilitado y nos permite introducir el número de unidades que indiquemos en la correspondiente casilla.



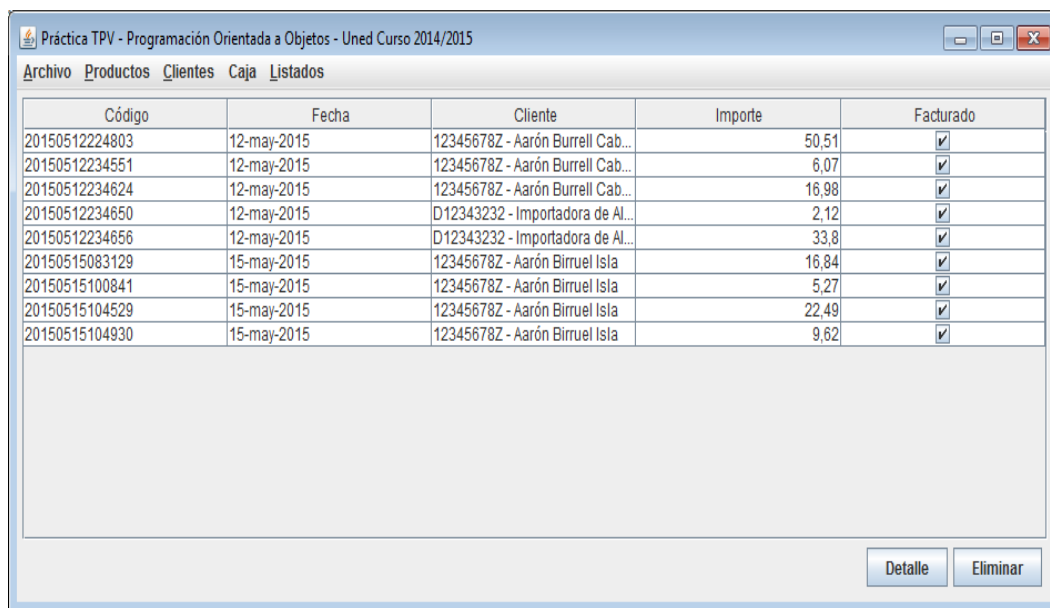
Esta manera de introducir productos en el ticket tiene el mismo efecto que hacerlo desde la tabla de productos.

Pulsando el botón "Finalizar" cierra la venta al cliente seleccionado en la parte superior con los productos incluidos en la tabla de la derecha.

Adicionalmente para agilizar el proceso de venta, el sistema abre un nuevo ticket de manera automática para que no haya que realizar ningún procedimiento manual.

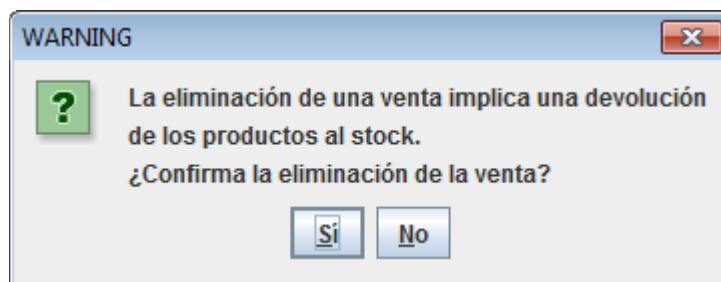
En el caso de clientes dados de alta en el sistema, habría que cambiar el mismo en el selector superior; no obstante en el caso de tener un cliente genérico este paso no sería necesario. La selección del cliente puede realizarse en cualquier momento de la venta hasta la pulsación del botón "Finalizar" momento en el cual la venta se hace efectiva.

**Ventas.** Las ventas realizadas por la caja del tpv pasar a este apartado donde en cualquier momento se pueden consultar. Cada ticket de venta dispone de un código único con formato AAAAMMDDHHMMSS, donde AAAA es el año en curso, MM el mes en que se genera la venta, DD el día de la venta, HHMMSS representan las horas, minutos y segundos en el que se realiza la venta. Este dato, junto con los datos del cliente y el importe total de la venta puede verse en la siguiente ventana:



| Código         | Fecha       | Cliente                          | Importe | Facturado                           |
|----------------|-------------|----------------------------------|---------|-------------------------------------|
| 20150512224803 | 12-may-2015 | 12345678Z - Aarón Burrell Cab... | 50,51   | <input checked="" type="checkbox"/> |
| 20150512234551 | 12-may-2015 | 12345678Z - Aarón Burrell Cab... | 6,07    | <input checked="" type="checkbox"/> |
| 20150512234624 | 12-may-2015 | 12345678Z - Aarón Burrell Cab... | 16,98   | <input checked="" type="checkbox"/> |
| 20150512234650 | 12-may-2015 | D12343232 - Importadora de Al... | 2,12    | <input checked="" type="checkbox"/> |
| 20150512234656 | 12-may-2015 | D12343232 - Importadora de Al... | 33,8    | <input checked="" type="checkbox"/> |
| 20150515083129 | 15-may-2015 | 12345678Z - Aarón Birruel Isla   | 16,84   | <input checked="" type="checkbox"/> |
| 20150515100841 | 15-may-2015 | 12345678Z - Aarón Birruel Isla   | 5,27    | <input checked="" type="checkbox"/> |
| 20150515104529 | 15-may-2015 | 12345678Z - Aarón Birruel Isla   | 22,49   | <input checked="" type="checkbox"/> |
| 20150515104930 | 15-may-2015 | 12345678Z - Aarón Birruel Isla   | 9,62    | <input checked="" type="checkbox"/> |



El sistema no permite modificar una venta ya realizada. Sin embargo, es posible hacer la devolución de una venta reintegrando los productos incluidos en el stock. Si pulsamos sobre el botón "Eliminar" se informa de esta posibilidad:



Indicar que se considera que las ventas que ya han sido facturadas, no pueden ser devueltas por tanto no puede ser eliminada la venta.

También se incluye la posibilidad de ver el detalle de cada venta realizada.

Seleccionando una venta desde la table y pulsando el botón "Detalle" se podrá ver el detalle del ticket:

 Detalle ticket 20150515104529 

**Ticket**

ticket nº 20150515104529 fecha: Fri May 15 10:45:29 CEST 2015

| Código | Descripción              | Unidades | Precio | IVA | Importe |
|--------|--------------------------|----------|--------|-----|---------|
| 1      | Aceite virgen extra H... | 3        | 5,62   | 21% | 16,87   |
| 1      | Aceite virgen extra H... | 1        | 5,62   | 21% | 5,62    |

#### **Nivel 4: Facturación y clientes del establecimiento comercial.**

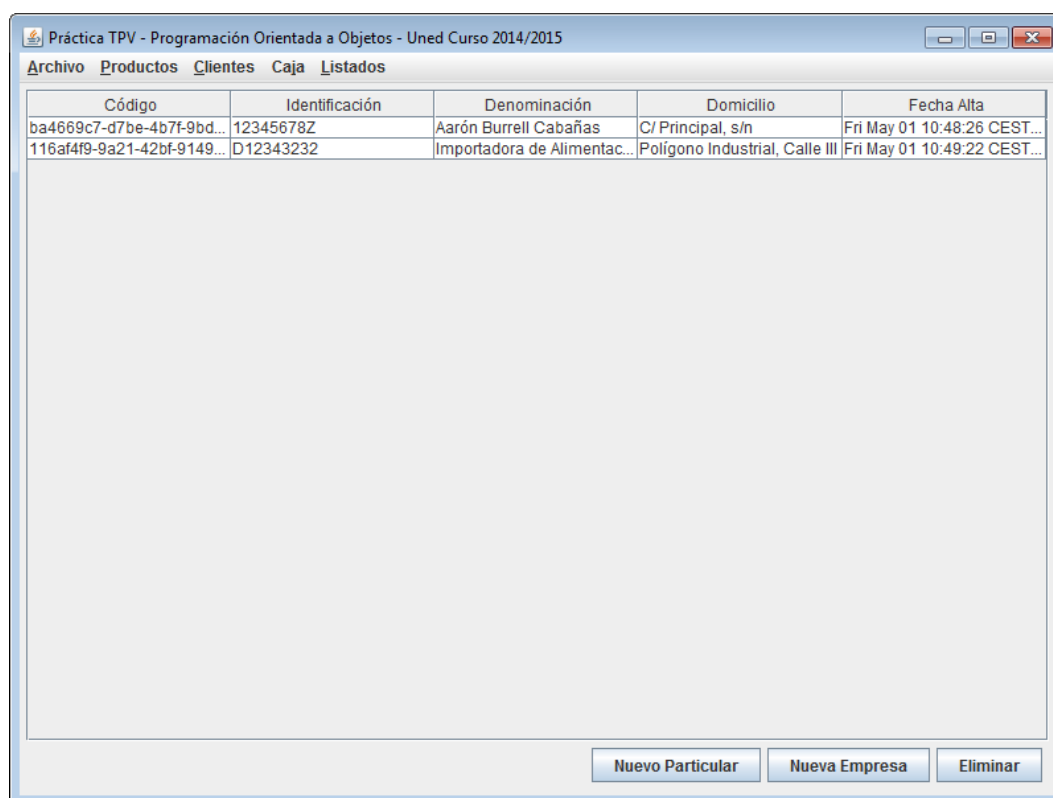
En este nivel de acabado, se implementan las funcionalidades de facturación y gestión de clientes de la aplicación. Asimismo, se generan los listados requeridos en el enunciado como veremos a continuación.

- El sistema mantiene un control de los diferentes clientes que trabajan con el establecimiento comercial. Los clientes están identificados en el sistema por los siguientes datos: código identificativo del cliente, NIF o CIF, nombre y apellidos / razón social, domicilio, fecha de alta en el sistema.
- El sistema permite dar de alta nuevos clientes, dar de baja clientes existentes así como modificar los datos de los mismos.
- Se permite realizar la importación y exportación de los clientes a/desde ficheros.
- Se permite generar facturas a partir de un conjunto de tickets. Puede generar facturas agrupando diferentes tickets siempre y cuando pertenezcan al mismo cliente y se han realizado dentro del mismo periodo fiscal (es decir, dentro del mismo año). La información que se incluye en cada factura es, la siguiente: número de la factura (identificador único), CIF del vendedor, razón social del vendedor, fecha de emisión de la factura, datos del cliente, listado de los diferentes productos vendidos especificando para cada producto, el ticket en el que se encuentra, su cantidad vendida e importe total así como suma del total de la venta.
- Se permite realizar la importación y exportación de las facturas a/desde ficheros.
- Se generan 3 listados:
  - 1.- ventas realizadas en un intervalo de tiempo determinado agrupadas estas ventas por clientes.
  - 2.- ventas realizadas en un intervalo de tiempo determinado a un cliente.
  - 3.- ranking de productos más vendidos en un intervalo de tiempo determinado.

A continuación se describe la operativa de este nivel de acabado:

**Menu Clientes.** De manera análoga al menú de Productos, al pulsar la correspondiente opción de menú aparece la ventana principal del mantenimiento de clientes. Desde este apartado tenemos las opciones para creación, modificación y eliminación de clientes del tpv.

Es este caso, podemos encontrar una principal diferencia respecto al mantenimiento de productos; ya que podemos dar de alta de manera separada los clientes si se tratan de clientes particulares o empresas a través de los correspondientes botones.



Para **crear** un nuevo cliente **particular**, pulsamos sobre el botón "Nuevo Particular" de manera que nos aparece la siguiente ventana:

Añadir cliente particular.

NIF

Nombre

Apellidos

Domicilio

Guardar

Los campos a completar son:

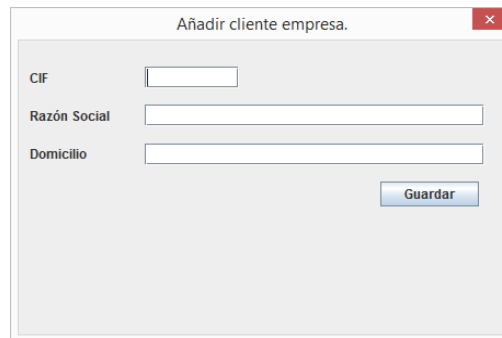
- Nif: Con el número de identificación del cliente. Se realiza una validación para introducir NIFs con formato válido.
- Nombre: Nombre del cliente.
- Apellidos: Apellido o apellidos del cliente.
- Domicilio: Dirección completa del cliente.

En caso de completarse correctamente la inserción de los respectivos campos, la ventana limpia



los campos de texto esperando que sea introducido un nuevo cliente particular.

Para **crear** un nuevo cliente de tipo **empresa**, pulsamos sobre el botón "Nueva Empresa" y nos aparecerá la correspondiente ventana donde poder introducir los datos de la empresa:

Una ventana de diálogo con el título "Añadir cliente empresa." y un botón de cerrar (X) en la esquina superior derecha. El interior de la ventana contiene tres campos de texto etiquetados como "CIF", "Razón Social" y "Domicilio". Debajo de estos campos hay un botón "Guardar".

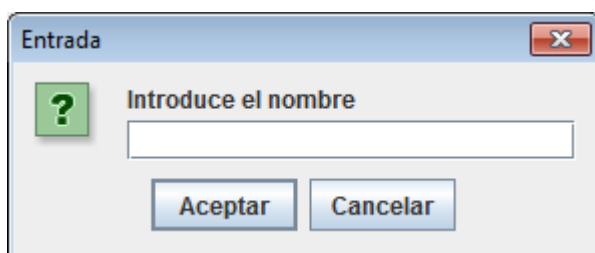
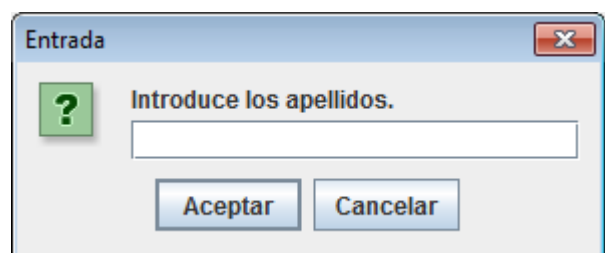
Los datos a completar son:

- Cif: Código de identificación de la empresa. Se realiza una validación para comprobar que se introduce un CIF con formato válido.
- Razón Social: Denominación de la empresa.
- Domicilio: Dirección completa de la empresa.

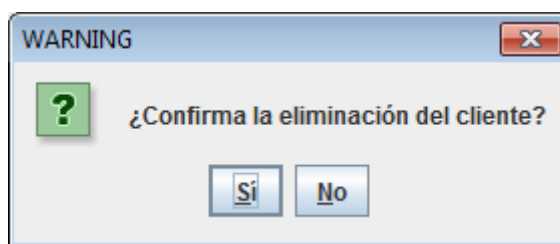
En caso de completarse correctamente la inserción de los respectivos campos, la ventana limpia los campos de texto esperando que sea introducido un nuevo cliente de tipo empresa.

Con respecto a la validación de números de NIF y CIF se ha utilizado una librería para tal fin publicada por la Agencia Tributaria como archivo .jar importado en el directorio lib del proyecto.

Para **modificar** los datos del cliente producto, se puede hacer directamente sobre la tabla haciendo doble click sobre el dato a modificar. De este modo se simplifica la edición de los clientes. En el caso de un cliente particular, dado que el nombre y apellidos aparecen juntos en la columna denominación aparecen las respectivas ventanas modales para introducir por separado el nombre y los apellidos del cliente.

Una ventana modal con el título "Entrada" y un botón de cerrar (X) en la esquina superior derecha. El interior contiene un icono de interrogación verde, el texto "Introduce el nombre" y un campo de texto. En la parte inferior hay dos botones: "Aceptar" y "Cancelar".Una ventana modal con el título "Entrada" y un botón de cerrar (X) en la esquina superior derecha. El interior contiene un icono de interrogación verde, el texto "Introduce los apellidos." y un campo de texto. En la parte inferior hay dos botones: "Aceptar" y "Cancelar".

Para **eliminar** un cliente del directorio, basta con seleccionarlo en la tabla del inventario y pulsar sobre el botón "Eliminar". Se mostrará una ventana como la siguiente pidiendo confirmación de la eliminación para evitar borrados accidentales.



**Facturas.** La última opción del menú caja se corresponde con las facturas. Se ofrece la posibilidad de generar facturas a los clientes dentro de un periodo fiscal.

Como se puede observar en la siguiente imagen, para cada factura generada se muestra un código de factura único generado por el sistema. Una descripción del cliente de la factura y el importe total de la misma.

Práctica TPV - Programación Orientada a Objetos - Uned Curso 2014/2015

Archivo Productos Clientes Caja Listados

| Factura N°                           | Cliente                                      | Importe |
|--------------------------------------|--|---------|
| 3b66b4dd-aa92-4611-a068-c46e9859627d | D12343232 - Importadora de Alimentación S... | 96.56   |
| 94636d0a-1b3a-43cf-ab0c-ffa372ee457e | D12343232 - Importadora de Alimentación S... | 193.29  |
| 8d1c4f04-7d7c-4820-9be1-04de35d150bb | 12345678Z - Aarón Burrell Cabañas            | 40.01   |
| 5cd24f8e-ec60-4eaa-ac48-2395f6c2cf51 | 12345678Z - Aarón Burrell Cabañas            | 40.01   |
| a59a0f1a-70cc-4445-bd3e-bf823a3c4560 | 12345678Z - Aarón Burrell Cabañas            | 5.54    |
| 189ddb4f-d13a-4c1e-be44-a41c0d7752a5 | D12343232 - Importadora de Alimentación S... | 265.04  |

Nueva Factura Detalle

Para generar una nueva factura, se pulsa el botón "Nueva Factura" y el tpv mostrará la ventana siguiente:

La generación de facturas consiste en la selección del cliente a facturar e introducir el año correspondiente del que se quiere obtener su factura. Pulsando el botón "Generar Factura" el proceso queda completado.

Cada venta solamente puede ser facturada una única vez de modo que las ventas incluidas en la factura a generar será únicamente las que se hayan producido desde la última vez que se hubiera facturado a este cliente en el mismo periodo fiscal.

Una vez generada la factura, se puede consultar su detalle seleccionándola en la tabla de facturas y pulsando sobre el botón "Detalle".

| Código | Ticket         | Descripción              | Unidades | Precio | Importe |
|--------|----------------|--------------------------|----------|--------|---------|
| 008    | 20150501105141 | Queso Parmigiano Regg... | 20       | 4,83   | 96,56   |
| 1      | 20150501105320 | Aceite virgen extra H... | 2        | 5,62   | 11,24   |
| 003    | 20150501105320 | Vino Rosado Lambrusco... | 2        | 3,24   | 6,47    |
| 004    | 20150501105320 | Filete de bacalao PES... | 2        | 5,27   | 10,53   |
| 005    | 20150501105320 | Cuarto de cordero lec... | 2        | 17,74  | 35,48   |
| 006    | 20150501105320 | Lubina, pieza            | 2        | 2,77   | 5,54    |
| 007    | 20150501105320 | Pera Conferencia, al ... | 2        | 2,02   | 4,05    |
| 008    | 20150501105320 | Queso Parmigiano Regg... | 2        | 4,83   | 9,66    |
| 009    | 20150501105320 | Mayonesa MUSA, frasco... | 2        | 1,73   | 3,46    |
| 010    | 20150501105320 | Natillas de vainilla ... | 2        | 2,12   | 4,25    |
| 011    | 20150501105320 | Champiñon entero mini... | 2        | 1,73   | 3,46    |
| 012    | 20150501105320 | Digestive con chocola... | 2        | 1,3    | 2,61    |
| 1      | 20150501141915 | Aceite virgen extra H... | 1        | 5,62   | 5,62    |
| 002    | 20150501141915 | Detergente en polvo S... | 1        | 16,84  | 16,84   |
| 003    | 20150501141915 | Vino Rosado Lambrusco... | 1        | 3,24   | 3,24    |
| 004    | 20150501141915 | Filete de bacalao PES... | 1        | 5,27   | 5,27    |
| 005    | 20150501141915 | Cuarto de cordero lec... | 1        | 17,74  | 17,74   |
| 006    | 20150501141915 | Lubina, pieza            | 1        | 2,77   | 2,77    |
| 007    | 20150501141915 | Pera Conferencia, al ... | 1        | 2,02   | 2,02    |
| 008    | 20150501141915 | Queso Parmigiano Regg... | 1        | 4,83   | 4,83    |
| 009    | 20150501141915 | Mayonesa MUSA, frasco... | 1        | 1,73   | 1,73    |
| 010    | 20150501141915 | Natillas de vainilla ... | 1        | 2,12   | 2,12    |
| 011    | 20150501141915 | Champiñon entero mini... | 1        | 1,73   | 1,73    |
| 012    | 20150501141915 | Digestive con chocola... | 1        | 1,3    | 1,3     |
| 013    | 20150501141915 | Limpiahogar ph neutro... | 1        | 3,13   | 3,13    |
| 014    | 20150501141915 | Champú fuerza brillo ... | 1        | 3,41   | 3,41    |

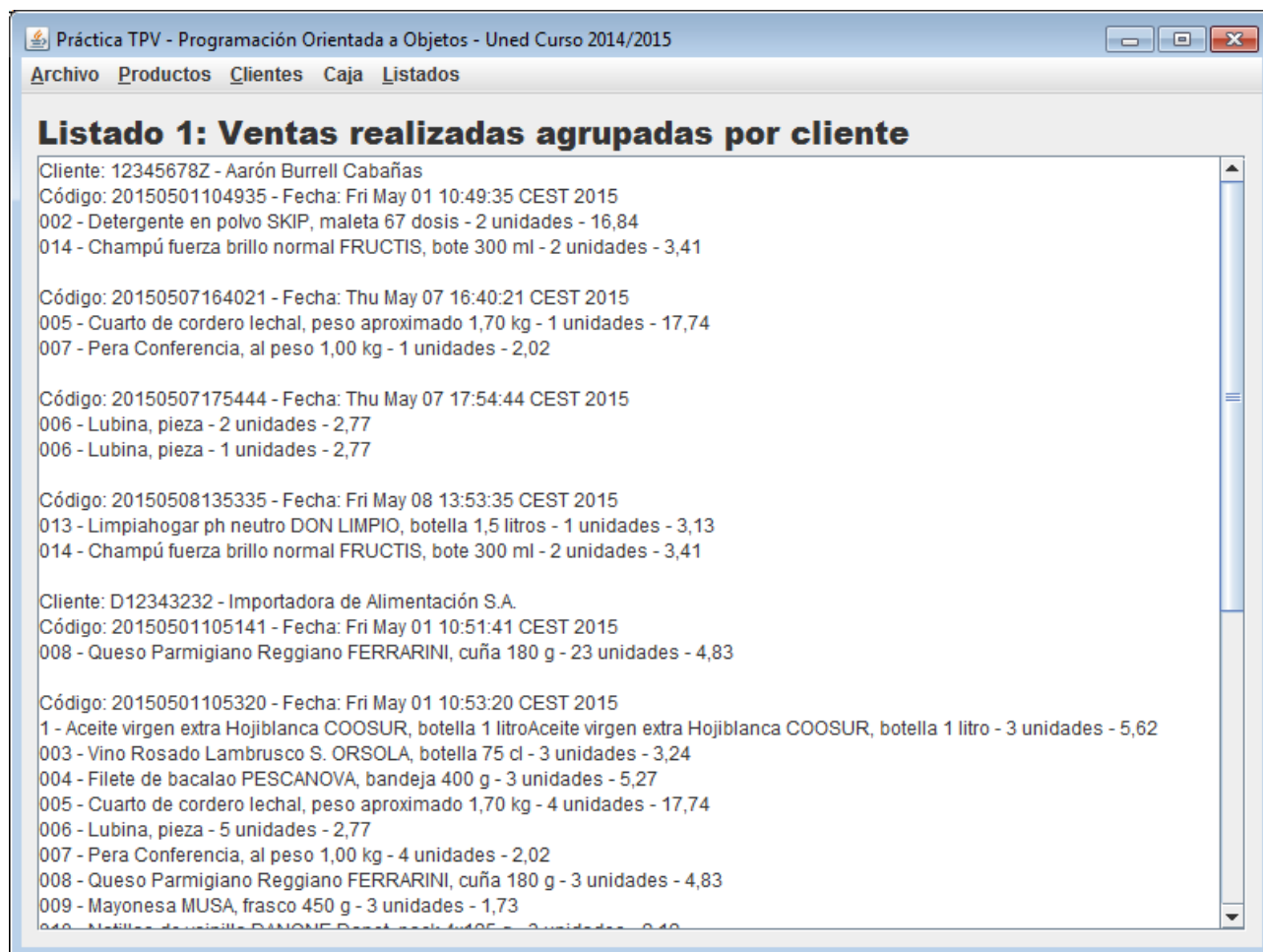
Total Factura: 265,04

Como se puede comprobar, para cada producto vendido, se incluye una línea en la factura incluyendo su código, el ticket en el que fué vendido, su descripción, las unidades y los precios

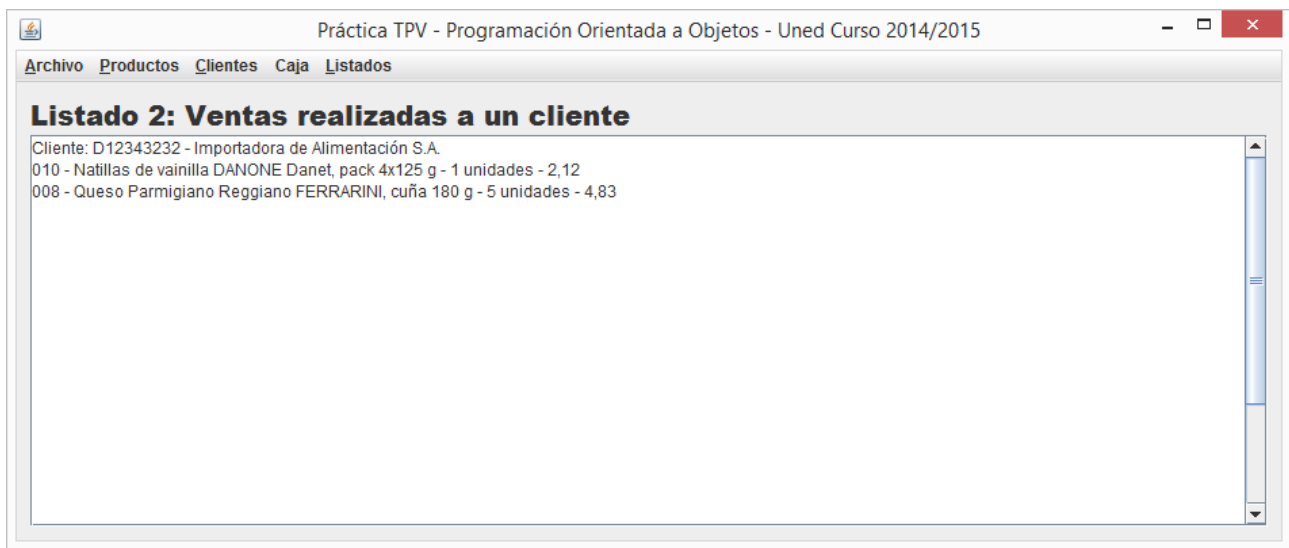
unitario e importe. Asimismo también se incluye el importe total de la factura.

**Menu Listados.** Se incluyen algunos listados que pueden resultar de utilidad a saber:

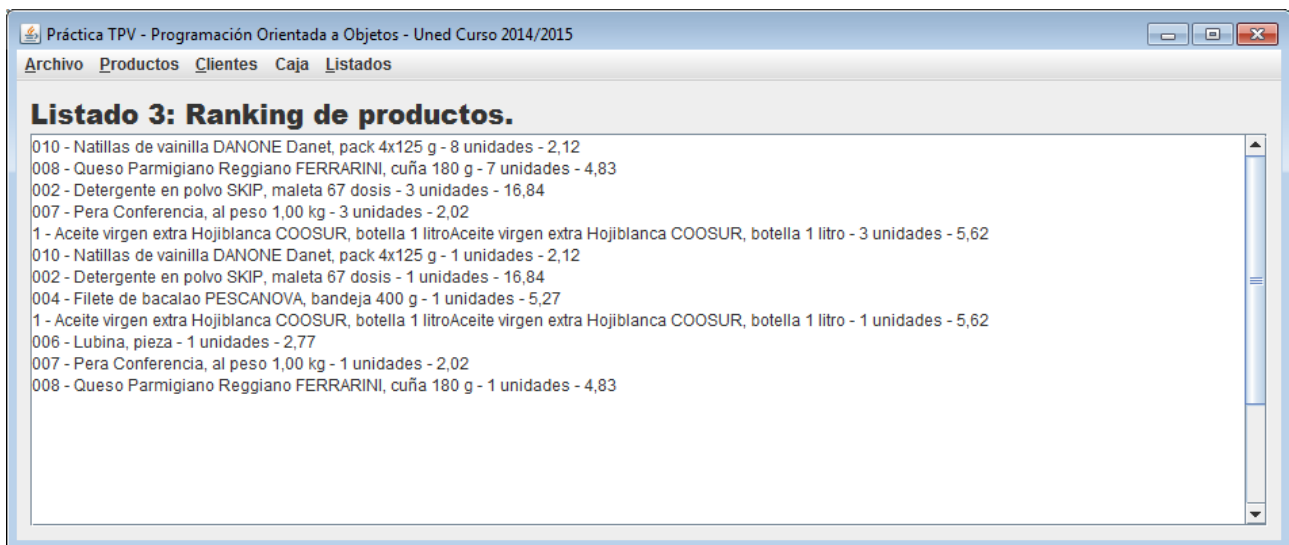
1.- ventas realizadas en un intervalo de tiempo determinado agrupadas por clientes. Para este primer listado, los parámetros a introducir son: la fecha inicial y la fecha final del intervalo. Es importante introducir las fechas en el formato correcto mm/dd/aaaa ya que en caso contrario, obtendremos un mensaje de error.



2.- ventas realizadas en un intervalo de tiempo determinado a un cliente. Para este segundo listado, los parámetros a introducir son: la fecha inicial, la fecha final del intervalo y el nif-cif del cliente. Es importante introducir las fechas en el formato correcto mm/dd/aaaa y el identificador correcto del cliente ya que en caso contrario, obtendremos un mensaje de error.

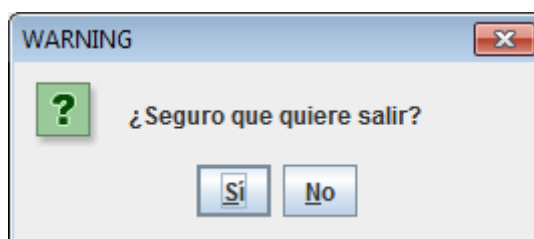


3.- ranking de productos más vendidos en un intervalo de tiempo determinado. De nuevo para este listado, los parámetros a introducir son: la fecha inicial y la fecha final del intervalo en el mismo formato mm/dd/aaaa.



### Finalización del Programa.

La salida del programa se realiza de manera natural pulsando para cerrar la ventana o sobre la opción del menú "Archivo -> Salir". En cualquiera de las opciones, se realiza una exportación de los datos con el fin de generar una "Snapshot" del estado del programa en el momento de la finalización de la ejecución.



## Anexo 1.- Código Fuente.

### Tpv.java

```
package poowned;

/**
 * Sistema Terminal Punto de Venta Práctica de la asignatura Programación
 * Orientada a Objetos Curso 2014-2015 Clase requerida para el inicio de la
 * aplicación por el enunciado.
 *
 * @author Francisco Javier Crespo Jiménez
 */
public class Tpv {

    /**
     * Método principal de la aplicación
     *
     * @param args No se utilizan parámetros de entrada.
     */
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new Controlador();
            }
        });
    }
}
```

## Controlador.java

```
package pooned;

import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.text.SimpleDateFormat;
import java.util.Date;

import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

import pooned.gui.CajaPanel;
import pooned.gui.ClientesPanel;
import pooned.gui.FacturasPanel;
import pooned.gui.ProductosPanel;
import pooned.gui.VentasPanel;
import pooned.gui.VisorListadosPanel;
import pooned.modelos.Cliente;
import pooned.modelos.Directorio;
import pooned.modelos.Facturas;
import pooned.modelos.Inventario;
import pooned.modelos.Ventas;
import pooned.util.Listados;

/**
 * Clase Controlador de la aplicación. Ejecuta la ventana principal, los menús e
 * inicializa las estructuras de datos necesarias pasándolas como parámetros a
 * los paneles.
 */
@SuppressWarnings("serial")
public class Controlador extends JFrame {

    private Inventario inventario;
    private Directorio directorio;
    private Ventas ventas;
    private Facturas facturas;
    private JPanel panelProductos, panelClientes, panelCaja, panelVentas,
        panelFacturas;

    /**
     * Constructor de la clase. Inicializa las estructuras de datos y las
     * características de la GUI
     */
    public Controlador() {

        inventario = new Inventario();
```

```
    directorio = new Directorio();
    ventas = new Ventas(inventario);
    facturas = new Facturas();

    setPreferredSize(new Dimension(950, 600));
    setMinimumSize(new Dimension(950, 400));
    setLocationRelativeTo(null);
    setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);
    setTitle("Práctica TPV - Programación Orientada a Objetos - Uned Curso
2014/2015");
    initMenu();
    setVisible(true);
    importacion();

    addWindowListener(new WindowAdapter() {
        @Override
        public void windowClosing(WindowEvent ev) {
            cierre();
        }
    });
}

/**
 * Método auxiliar privado que crea el menú de la aplicación. Es llamado por
 * el constructor al inicio de la aplicación. Contiene los listener en forma
 * de clases anónimas con la comprobación de parámetros previa a mostrar el
 * panel correspondiente.
 */
private void initMenu() {

    JMenuBar menuBar = new JMenuBar();
    setJMenuBar(menuBar);

    JMenu menuArchivo = new JMenu("Archivo");
    menuArchivo.setMnemonic(KeyEvent.VK_A);

    menuBar.add(menuArchivo);

    JMenuItem menuImportar = new JMenuItem("Importar", KeyEvent.VK_I);
    menuImportar.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            importacion();
        }
    });
    menuArchivo.add(menuImportar);

    JMenuItem menuExportar = new JMenuItem("Exportar", KeyEvent.VK_E);
    menuExportar.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            exportacion();
        }
    });
    menuArchivo.add(menuExportar);
}
```



```
JMenuItem menuSalir = new JMenuItem("Salir", KeyEvent.VK_S);
menuSalir.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        cierre();
    }
});
menuArchivo.add(menuSalir);

JMenu menuProductos = new JMenu("Productos");
menuProductos.setMnemonic(KeyEvent.VK_P);
menuBar.add(menuProductos);

JMenuItem menuProductoInventario = new JMenuItem("Inventario",
KeyEvent.VK_V);
menuProductoInventario.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        panelProductos = new ProductosPanel(inventario);
        setContentPane(panelProductos);
        setVisible(true);
    }
});
menuProductos.add(menuProductoInventario);

JMenu menuClientes = new JMenu("Clientes");
menuClientes.setMnemonic(KeyEvent.VK_C);
menuBar.add(menuClientes);
JMenuItem menuDirectorioClientes = new JMenuItem("Directorio",
KeyEvent.VK_D);
menuDirectorioClientes.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        panelClientes = new ClientesPanel(directorio);
        setContentPane(panelClientes);
        setVisible(true);
    }
});
menuClientes.add(menuDirectorioClientes);

JMenu menuCaja = new JMenu("Caja");
menuCaja.setMnemonic(KeyEvent.VK_J);
menuBar.add(menuCaja);

JMenuItem menuCajaTpv = new JMenuItem("TPV", KeyEvent.VK_T);
menuCajaTpv.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if ((inventario.size() == 0 || directorio.size() == 0)) {
            JOptionPane.showMessageDialog(
                null,
                "No se puede hacer ventas sin productos y clientes.",
                "Error en la venta.",
                JOptionPane.ERROR_MESSAGE);
        } else {
            panelCaja = new CajaPanel(inventario, ventas,
directorio);
            setContentPane(panelCaja);
        }
    }
});
```

```
        setVisible(true);
    }
}
});
menuCaja.add(menuCajaTpv);

JMenuItem menuVentas = new JMenuItem("Ventas", KeyEvent.VK_V);
menuVentas.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        panelVentas = new VentasPanel(ventas);
        setContentPane(panelVentas);
        setVisible(true);
    }
});
menuCaja.add(menuVentas);

JMenuItem menuFacturas = new JMenuItem("Facturas", KeyEvent.VK_F);
menuFacturas.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        panelFacturas = new FacturasPanel(directorio, ventas,
facturas);
        setContentPane(panelFacturas);
        setVisible(true);
    }
});
menuCaja.add(menuFacturas);

JMenu menuListados = new JMenu("Listados");
menuListados.setMnemonic(KeyEvent.VK_L);
menuBar.add(menuListados);

JMenuItem menuListado1 = new JMenuItem(
    "1.- Ventas agrupadas por clientes.");
menuListado1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String s1 = JOptionPane
            .showInputDialog("Fecha Inicial (mm/dd/yyyy)");
        String s2 = JOptionPane
            .showInputDialog("Fecha Final (mm/dd/yyyy)");
        try {
            SimpleDateFormat date = new SimpleDateFormat("yyyy-MM-
dd");
            Date fechaInicial = new Date(s1);
            Date fechaFinal = new Date(s2);
            Listados listados = new Listados(ventas);
            VisorListadosPanel panelListado1 = new
                "Listado 1: Ventas realizadas agrupadas
por cliente",
            listados.getIntervaloGroupByClientes(fechaInicial,
                fechaFinal));
            setContentPane(panelListado1);
            setVisible(true);
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(null,
```

```
fechas.",
                                "No se ha podido crear intervalo de
                                "Error.", JOptionPane.ERROR_MESSAGE);
        }
    });
    menuListados.add(menuListado1);

    JMenuItem menuListado2 = new JMenuItem("2.- Ventas a un cliente.");
    menuListado2.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            String s1 = JOptionPane
                .showInputDialog("Fecha Inicial (mm/dd/yyyy)");
            String s2 = JOptionPane
                .showInputDialog("Fecha Final (mm/dd/yyyy)");
            String c = JOptionPane.showInputDialog("Nif-Cif del
cliente");

            try {
                Date fechaInicial = new Date(s1);
                Date fechaFinal = new Date(s2);
                Cliente cliente = directorio.findById(c);
                Listados listados = new Listados(ventas);
                VisorListadosPanel panellListado2 = new
VisorListadosPanel(
                                "Listado 2: Ventas realizadas a un
cliente",

                                listados.getIntervaloCliente(fechaInicial,
                                                                fechaFinal, cliente));
                setContentPane(panellListado2);
                setVisible(true);
            } catch (IllegalArgumentException ex) {
                JOptionPane.showMessageDialog(null,
                                "Error en los parámetros.", "Error.",
                                JOptionPane.ERROR_MESSAGE);
            }
        }
    });
    menuListados.add(menuListado2);

    JMenuItem menuListado3 = new JMenuItem("3.- Ranking de productos.");
    menuListado3.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            String s1 = JOptionPane
                .showInputDialog("Fecha Inicial (mm/dd/yyyy)");
            String s2 = JOptionPane
                .showInputDialog("Fecha Final (mm/dd/yyyy)");

            try {
                Date fechaInicial = new Date(s1);
                Date fechaFinal = new Date(s2);
                Listados listados = new Listados(ventas);
                VisorListadosPanel panellListado3 = new
VisorListadosPanel(
                                "Listado 2: Ranking de productos.",
                                listados
                                                                .rankingProductosIntervalo(f
```

```
echaInicial,

                                                                    fechaFinal));

        setContentPane(panelListado3);
        setVisible(true);
    } catch (IllegalArgumentException ex) {
        JOptionPane.showMessageDialog(null,
            "No se ha podido crear intervalo de
fechas.",
                                                                    "Error.", JOptionPane.ERROR_MESSAGE);
    }
}

});
menuListados.add(menuListado3);

}

/**
 * Método auxiliar privado que importa los datos de la aplicación desde los
 * archivos. Es llamado por el constructor al inicio de la aplicación.
 */
private void importacion() {
    if (JOptionPane.showConfirmDialog(null,
        "¿Desea realizar importación desde los archivos?", "WARNING",
        JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION) {
        try {
            inventario.importar();
            directorio.importar();
            ventas.importar();
            facturas.importar();
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(null,
                ex.getMessage(), "Error.",
JOptionPane.ERROR_MESSAGE);
        }
    }
}

/**
 * Método auxiliar privado que exporta los datos de la aplicación desde los
 * archivos.
 */
private void exportacion() {
    try {
        inventario.exportar();
        directorio.exportar();
        ventas.exportar();
        facturas.exportar();
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(null,
            ex.getMessage(), "Error.",
JOptionPane.ERROR_MESSAGE);
    }
}

/**
 * Método auxiliar privado que realiza un cierre ordenado de la aplicación.
```

```
* Llama al método para exportar los datos antes de salir.  
*/  
private void cierre() {  
    if (JOptionPane.showConfirmDialog(null,  
        "¿Seguro que quiere salir?", "WARNING",  
        JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION) {  
        exportacion();  
        System.exit(0);  
    }  
}  
}
```

## BuscarProductoPanel.java

```
package pooned.gui;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JTextField;
import javax.swing.border.EmptyBorder;

import pooned.modelos.Inventario;
import pooned.modelos.Ticket;
import pooned.modelos.ProductoInventariado;
import pooned.util.QuantityWrongException;

/**
 * Clase del panel de búsqueda de los artículos
 * para ser incluidos en la venta.
 *
 * @author Francisco Javier Crespo Jiménez
 */
@SuppressWarnings({ "serial", "unused" })
public class BuscarProductoPanel extends JPanel {

    private JTextField txtBusqueda, txtCantidad;
    private JLabel lblProducto;
    private JButton btnInsertar;
    private Inventario inventario;
    private ProductoInventariado producto;
    private Ticket ticket;

    /**
     * Constructor de la clase
     * @param inventario gestor de productos
     * @param ticket ticket en curso
     */
    public BuscarProductoPanel(Inventario inventario, Ticket ticket) {

        this.inventario = inventario;
        this.ticket = ticket;

        setBounds(100, 100, 450, 200);
        setBorder(new EmptyBorder(5, 5, 5, 5));
        setLayout(null);

        JRadioButton rdbtnCodigo = new JRadioButton("Código");
        rdbtnCodigo.setSelected(true);
        rdbtnCodigo.setBounds(10, 10, 112, 23);
        add(rdbtnCodigo);
    }
}
```

```
JRadioButton rdbtnDescripcion = new JRadioButton("Descripción");
rdbtnDescripcion.setBounds(10, 30, 112, 23);
add(rdbtnDescripcion);

ButtonGroup btnGrp = new ButtonGroup();
btnGrp.add(rdbtnCodigo);
btnGrp.add(rdbtnDescripcion);

txtBusqueda = new JTextField();
txtBusqueda.setBounds(130, 20, 250, 20);
add(txtBusqueda);
txtBusqueda.setColumns(10);

JButton btnBuscar = new JButton("Buscar");
btnBuscar.setBounds(130, 50, 89, 23);
add(btnBuscar);
btnBuscar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if (rdbtnCodigo.isSelected()) {
            producto = (ProductoInventariado) inventario
                .findByCodigo(txtBusqueda.getText());
        } else {
            producto = (ProductoInventariado) inventario
                .findByDescripcion(txtBusqueda.getText());
        }
        if (producto != null) {
            lblProducto.setText(producto.toString());
            btnInsertar.setEnabled(true);
        } else {
            lblProducto.setText("Sin producto encontrado.");
            btnInsertar.setEnabled(false);
        }
    }
});

lblProducto = new JLabel("Sin producto encontrado.");
lblProducto.setBounds(10, 90, 400, 20);
lblProducto.setVisible(true);
add(lblProducto);

txtCantidad = new JTextField(10);
txtCantidad.setBounds(80, 150, 89, 23);
txtCantidad.setText("1");
add(txtCantidad);

JLabel lblUnidades = new JLabel("Unidades");
lblUnidades.setBounds(180, 150, 89, 23);
add(lblUnidades);

btnInsertar = new JButton("Añadir");
btnInsertar.setBounds(280, 150, 89, 23);
btnInsertar.setEnabled(false);
add(btnInsertar);
btnInsertar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
```

```
        try {
            ticket.addProducto(producto,
Integer.parseInt(txtCantidad.getText().toString()));
        } catch (NumberFormatException | QuantityWrongException ex) {
            JOptionPane.showMessageDialog(null,
                "Formato o cantidad incorrectos.",
                "Error en la venta.",
JOptionPane.ERROR_MESSAGE);
        }

        lblProducto.setText(producto.toString());
    }
}
}
```



## CajaPanel.java

```
package poouned.gui;

import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Observable;
import java.util.Observer;

import javax.swing.DefaultComboBoxModel;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.ListSelectionModel;
import javax.swing.border.EmptyBorder;

import poouned.modelos.Cliente;
import poouned.modelos.Directorío;
import poouned.modelos.Inventario;
import poouned.modelos.ProductoInventariado;
import poouned.modelos.Ticket;
import poouned.modelos.Ventas;
import poouned.modelos.tablemodel.ProductosROTM;
import poouned.modelos.tablemodel.ProductosVendidosTM;
import poouned.util.QuantityWrongException;

/**
 * Clase para el panel de caja del TPV
 * Desde esta ventana se produce la venta de los productos.
 *
 * @author Francisco Javier Crespo Jiménez
 */
@SuppressWarnings({ "serial", "rawtypes", "unchecked" })
public class CajaPanel extends JPanel implements Observer {

    private JTable tabla, tablaTicket;
    private JComboBox comboCliente;
    private DefaultComboBoxModel comboModel;
    private ProductosROTM modelo;
    private ProductosVendidosTM modeloVentas;
    private JTextField txtCantidad;
    private Inventario inventario;
    private Directorío directorío;
    private Ventas ventas;
    private Ticket ticket;

    /**
     * Constructor de la clase
     */
}
```

```
*
* @param inventario gestor de productos
* @param ventas gestor de tickets
* @param directorio gestor de clientes
*/
public CajaPanel(Inventario inventario, Ventas ventas, Directorio directorio) {
    this.inventario = inventario;
    this.ventas = ventas;
    this.directorio = directorio;
    this.directorio.addObserver(this);
    this.ventas.addObserver(this);
    this.inventario.addObserver(this);
    initComponents();
    inicioVenta();
    setVisible(true);
}

/**
 * Método privado auxiliar para agrupar la inicialización de los componentes
gráficos.
 */
private void initComponents() {

    setBorder(new EmptyBorder(5, 5, 5, 5));
    setLayout(new BorderLayout());

    JPanel panelSuperior, panelInventario, panelTicket, panelTabla1,
panelTabla2, panelBotones1, panelBotones2;
    panelSuperior = new JPanel(new FlowLayout(FlowLayout.LEFT));

    panelInventario = new JPanel(new BorderLayout());
    panelTabla1 = new JPanel(new BorderLayout());
    panelBotones1 = new JPanel(new FlowLayout(FlowLayout.RIGHT));

    panelTicket = new JPanel(new BorderLayout());
    panelTabla2 = new JPanel(new BorderLayout());
    panelBotones2 = new JPanel(new FlowLayout(FlowLayout.RIGHT));

    add(panelSuperior, BorderLayout.NORTH);

    panelInventario.add(panelTabla1, BorderLayout.CENTER);
    panelInventario.add(panelBotones1, BorderLayout.SOUTH);
    add(panelInventario, BorderLayout.WEST);

    panelTicket.add(panelTabla2, BorderLayout.CENTER);
    panelTicket.add(panelBotones2, BorderLayout.SOUTH);
    add(panelTicket, BorderLayout.EAST);

    // Parte Superior
    comboCliente = new JComboBox();
    comboModel = new DefaultComboBoxModel(directorio.getAll().toArray());
    comboCliente.setModel(comboModel);
    comboCliente.setVisible(true);
    panelSuperior.add(comboCliente);
}
```

```
// Panel Inferior

// Inventario
JScrollPane scrollPane = new JScrollPane();
panelTabla1.add(scrollPane);
tabla = new JTable();
tabla.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
scrollPane.setViewportView(tabla);
modelo = new ProductosROTM(inventario);
tabla.setModel(modelo);

JButton btnFind = new JButton("Buscar");
btnFind.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        BuscarProductoPanel buscarProductoPanel = new
BuscarProductoPanel(
                                inventario, ticket);
        JDialog buscarProductoDialog = new JDialog();
        buscarProductoDialog.setModal(true);
        buscarProductoDialog.setResizable(false);
        buscarProductoDialog.setTitle("Buscar producto.");
        buscarProductoDialog.setBounds(100, 100, 450, 300);
        buscarProductoDialog.setContentPane(buscarProductoPanel);
        buscarProductoDialog.setVisible(true);
    }
});
panelBotones1.add(btnFind);

txtCantidad = new JTextField();
txtCantidad.setText("1");
panelBotones1.add(txtCantidad);
txtCantidad.setColumns(10);

JLabel lblUnidades = new JLabel("Unidades");
panelBotones1.add(lblUnidades);

JButton btnAdd = new JButton("Añadir ->");
btnAdd.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            if (tabla.getSelectedRowCount() > 0
                && Integer.parseInt(txtCantidad.getText()
                                    .toString()) > 0) {
                ProductoInventariado pi = (ProductoInventariado)
inventario
                                .get(tabla.getSelectedRow());
                ticket.addProducto(pi,
Integer.parseInt(txtCantidad
                                .getText().toString()));
            } else {
                JOptionPane.showMessageDialog(null,
"Formato de cantidad incorrecto.",
"Error en la venta.",
JOptionPane.ERROR_MESSAGE);
            }
        }
    }
});
```

```
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(null,
                "Formato de cantidad incorrecto.",
                "Error en la venta.",
                JOptionPane.ERROR_MESSAGE);
        } catch (QuantityWrongException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "Error en la venta.",
                JOptionPane.ERROR_MESSAGE);
        } catch (ArrayIndexOutOfBoundsException ex) {
            JOptionPane.showMessageDialog(null,
                "No se ha seleccionado un producto.",
                "Error en la venta.",
                JOptionPane.ERROR_MESSAGE);
        }
    }
});
panelBotones1.add(btnAdd);

// Ticket
JScrollPane scrollPane2 = new JScrollPane();
tablaTicket = new JTable();
scrollPane2.setViewportViewView(tablaTicket);
panelTicket.add(scrollPane2);

JButton btnEnd = new JButton("Finalizar");
btnEnd.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if (ticket.getProductos().size() > 0) {
            ticket.setCliente((Cliente)
directorio.get(comboCliente
                                .getSelectedIndex()));
            ticket.setFacturado(false);
            try {
                ventas.add(ticket);
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(null,
                    ex.getMessage(), "Error.",
                JOptionPane.ERROR_MESSAGE);
            }
            inicioVenta();
        } else {
            JOptionPane.showMessageDialog(null,
                "No se puede finalizar un ticket sin
productos.");
        }
    }
});
panelBotones2.add(btnEnd);
}

/**
 * Método privado auxiliar que se ejecuta al inicio del panel y cuando finaliza
una venta.
 * De este modo, se crea un nuevo ticket donde ir insertando artículos.
```

```
*/  
private void inicioVenta() {  
    ticket = new Ticket();  
    ticket.addObserver(this);  
    modeloVentas = new ProductosVendidosTM(ticket.getProductos());  
    tablaTicket.setModel(modeloVentas);  
}  
  
/**  
 * Método que implementa el método update interfaz Observable  
 * Cuando se produce un cambio en los modelos, se actualiza  
 * el interfaz de usuario para reflejar los cambios en la GUI  
 */  
@Override  
public void update(Observable o, Object arg) {  
    tablaTicket.updateUI();  
    tabla.updateUI();  
}  
  
}
```

## CientesPanel.java

```
package poouned.gui;

import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Observable;
import java.util.Observer;

import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.ListSelectionModel;
import javax.swing.border.EmptyBorder;

import poouned.modelos.Directorio;
import poouned.modelos.tablemodel.CientesTM;

/**
 * Clase para el panel de mantenimiento de clientes.
 *
 * @author Francisco Javier Crespo Jiménez
 */
@SuppressWarnings("serial")
public class CientesPanel extends JPanel implements Observer {

    private JTable tabla;
    private CientesTM modelo;
    private Directorio directorio;

    /**
     * Constructor de la clase.
     *
     * @param directorio gestor de clientes
     */
    public CientesPanel(Directorio directorio) {
        this.directorio = directorio;
        this.directorio.addObserver(this);
        setBorder(new EmptyBorder(5, 5, 5, 5));
        setLayout(new BorderLayout());

        JScrollPane scrollPane = new JScrollPane();
        add(scrollPane, BorderLayout.CENTER);
        setVisible(true);

        tabla = new JTable();
        tabla.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        scrollPane.setViewportView(tabla);
        modelo = new CientesTM(directorio);
        tabla.setModel(modelo);
    }
}
```

```
JPanel botonesPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));

JButton btnNuevoParticular = new JButton("Nuevo Particular");
botonesPanel.add(btnNuevoParticular);
btnNuevoParticular.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        JDialog d = makeDialog(new ParticularPanel(directorio));
        d.setTitle("Añadir cliente particular.");
        d.setVisible(true);
    }
});

JButton btnNuevaEmpresa = new JButton("Nueva Empresa");
botonesPanel.add(btnNuevaEmpresa);
btnNuevaEmpresa.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        JDialog d = makeDialog(new EmpresaPanel(directorio));
        d.setTitle("Añadir cliente empresa.");
        d.setVisible(true);
    }
});

JButton btnEliminar = new JButton("Eliminar");
botonesPanel.add(btnEliminar);
btnEliminar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if (JOptionPane.showConfirmDialog(null, "¿Confirma la
eliminación del cliente?", "WARNING",
JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION)
{
            try {
                directorio.remove(tabla.convertRowIndexToModel(tabla.getSelectedRow()));
            } catch (ArrayIndexOutOfBoundsException ex) {
                JOptionPane
                    .showMessageDialog(
                        null, "No se ha seleccionado un
cliente.",
                        "Error en la eliminación.",
JOptionPane.ERROR_MESSAGE);
            } catch (IndexOutOfBoundsException ex) {
                JOptionPane.showMessageDialog(
                        null, "No se ha podido eliminar el
cliente.",
                        "Error en la eliminación.",
JOptionPane.ERROR_MESSAGE);
            }
        }
    }
});
```

```
        add(botonesPanel, BorderLayout.SOUTH);
    }

    /**
     * Método que implementa el método update interfaz Observable
     * Cuando se produce un cambio en los modelos, se actualiza
     * el interfaz de usuario para reflejar los cambios en la GUI
     */
    @Override
    public void update(Observable o, Object arg) {
        tabla.updateUI();
    }

    /**
     * Método auxiliar para crear el cuadro de diálogo modal basado
     * en un panel
     *
     * @param p interfaz JPanel
     * @return ventana JDialog con las dimensiones especificadas y modal
     */
    private JDialog makeDialog(JPanel p) {
        JDialog d = new JDialog();
        d.setModal(true);
        d.setResizable(false);
        d.setBounds(100, 100, 450, 300);
        d.setContentPane(p);

        return d;
    }
}
```



## EmpresaPanel.java

```
package poouned.gui;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.border.EmptyBorder;

import poouned.modelos.Directorio;
import poouned.modelos.Empresa;

/**
 * Clase del panel para introducir nueva empresa.
 *
 * @author Francisco Javier Crespo Jiménez
 */
@SuppressWarnings({ "serial", "unused" })
public class EmpresaPanel extends JPanel {

    private JTextField txtCif;
    private JTextField txtRazonSocial;
    private JTextField txtDomicilio;
    private Directorio directorio;

    /**
     * Constructor de la clase.
     * @param directorio gestor de clientes.
     */
    public EmpresaPanel(Directorio directorio) {
        this.directorio = directorio;
        setBorder(new EmptyBorder(5, 5, 5, 5));
        setLayout(null);

        JLabel lblCif = new JLabel("CIF");
        lblCif.setBounds(10, 27, 83, 14);
        add(lblCif);

        txtCif = new JTextField();
        txtCif.setBounds(115, 24, 86, 20);
        txtCif.setColumns(13);
        txtCif.requestFocus();
        add(txtCif);

        JLabel lblRazonSocial = new JLabel("Razón Social");
        lblRazonSocial.setBounds(10, 62, 83, 14);
        add(lblRazonSocial);

        txtRazonSocial = new JTextField();
        txtRazonSocial.setBounds(115, 59, 310, 20);
        txtRazonSocial.setColumns(10);
    }
}
```

```
add(txtRazonSocial);

JLabel lblDomicilio = new JLabel("Domicilio");
lblDomicilio.setBounds(10, 97, 83, 14);
add(lblDomicilio);

txtDomicilio = new JTextField();
txtDomicilio.setBounds(115, 95, 310, 20);
txtDomicilio.setColumns(10);
add(txtDomicilio);

JButton btnGuardar = new JButton("Guardar");
btnGuardar.setBounds(330, 130, 90, 23);
add(btnGuardar);
btnGuardar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if (validarFormulario()) {
            try {
                Empresa empresa = new Empresa();
                empresa.setCif(txtCif.getText().toString());

                empresa.setRazonSocial(txtRazonSocial.getText());
                empresa.setDomicilio(txtDomicilio.getText());
                directorio.add(empresa);
                txtCif.setText("");
                txtRazonSocial.setText("");
                txtDomicilio.setText("");
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(null,
                    ex.getMessage(), "Error.",
JOptionPane.ERROR_MESSAGE);
            } finally {
                txtCif.requestFocus();
            }
        }
    }
});

/**
 * Método auxiliar para realizar comprobaciones sobre los datos introducidos en
el formulario
 *
 * @return boolean con el resultado de la validación
 */
private boolean validarFormulario() {
    if (txtCif.getText().trim().length() == 0) {
        JOptionPane.showMessageDialog(this,
            "El campo CIF no puede estar vacio.");
        txtCif.requestFocus();
        return false;
    } else if (txtCif.getText().trim().length() > 9) {
        JOptionPane.showMessageDialog(this,
            "El campo CIF no puede tener más de 9 caracteres.");
        txtCif.requestFocus();
        return false;
    }
}
```

```
} else if (txtRazonSocial.getText().trim().length() == 0) {  
    JOptionPane.showMessageDialog(this,  
        "El campo Razón Social no puede estar vacío.");  
    txtRazonSocial.requestFocus();  
    return false;  
} else if (txtDomicilio.getText().trim().length() == 0) {  
    JOptionPane.showMessageDialog(this,  
        "El campo Domicilio no puede estar vacío.");  
    txtDomicilio.requestFocus();  
    return false;  
} else {  
    return true;  
}  
  
}  
  
}
```

## FacturaPanel.java

```
package pooned.gui;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;

import javax.swing.DefaultComboBoxModel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;

import pooned.modelos.Cliente;
import pooned.modelos.Directorío;
import pooned.modelos.Factura;
import pooned.modelos.Facturas;
import pooned.modelos.ProductoInventariado;
import pooned.modelos.Ticket;
import pooned.modelos.Ventas;

import com.aeat.valida.Validador;

/**
 * Clase para generar las facturas.
 *
 * @author Francisco Javier Crespo Jiménez
 */
@SuppressWarnings({"serial", "rawtypes", "unused", "unchecked"})
public class FacturaPanel extends JPanel {
    private JTextField txtCifVendedor, txtRSVendedor, txtPeriodo;
    private JComboBox comboCliente;
    private Facturas facturas;
    private Ventas ventas;
    private Directorío directorío;

    /**
     * Constructor de la clase.
     *
     * @param facturas gestor de facturas.
     * @param directorío gestor de clientes.
     * @param ventas gestor de tickets (ventas).
     */
    public FacturaPanel(Facturas facturas, Directorío directorío, Ventas ventas) {
        this.facturas = facturas;
        this.directorio = directorío;
        this.ventas = ventas;
        setLayout(null);

        JLabel lblCliente = new JLabel("Cliente");
        lblCliente.setBounds(10, 49, 113, 14);
        add(lblCliente);
    }
}
```

```
        comboCliente = new JComboBox();
        comboCliente.setModel(new
DefaultComboBoxModel(directorio.getAll().toArray()));
        comboCliente.setBounds(179, 46, 234, 20);
        add(comboCliente);

        JLabel lblCifVendedor = new JLabel("Cif Vendedor");
        lblCifVendedor.setBounds(10, 97, 133, 14);
        add(lblCifVendedor);

        txtCifVendedor = new JTextField();
        txtCifVendedor.setBounds(179, 94, 234, 20);
        add(txtCifVendedor);
        txtCifVendedor.setColumns(10);

        JLabel lblRazonSocialVendedor = new JLabel("Razón Social Vendedor");
        lblRazonSocialVendedor.setBounds(10, 145, 159, 14);
        add(lblRazonSocialVendedor);

        txtRSVendedor = new JTextField();
        txtRSVendedor.setBounds(179, 142, 234, 20);
        add(txtRSVendedor);
        txtRSVendedor.setColumns(10);

        JLabel lblPeriodoFiscal = new JLabel("Período Fiscal");
        lblPeriodoFiscal.setBounds(10, 193, 133, 14);
        add(lblPeriodoFiscal);

        txtPeriodo = new JTextField();
        txtPeriodo.setBounds(180, 190, 86, 20);
        add(txtPeriodo);
        txtPeriodo.setColumns(10);

        JButton btnGenerarFactura = new JButton("Generar Factura");
        btnGenerarFactura.setBounds(276, 189, 137, 23);
        add(btnGenerarFactura);
        btnGenerarFactura.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if (validarFormulario()) {
                    try {
                        Cliente cliente = (Cliente)
directorio.get(comboCliente.getSelectedIndex());
                        String periodo =
txtPeriodo.getText().toString();
                        ArrayList<Ticket> ticketsFacturables =
ventas.getFacturables(cliente, periodo);
                        if (ticketsFacturables != null &&
ticketsFacturables.size() > 0) {
                            Factura factura = new Factura();
                            factura.setCliente(cliente);

                            factura.setCif(txtCifVendedor.getText().toString());

                            factura.setRazonSocial(txtRSVendedor.getText().toString());
                            factura.setPeriodoFiscal(periodo);
                            for (Ticket ticket : ticketsFacturables)
```

```
ticket.facturar();

        factura.setTickets(ticketsFacturables);
        facturas.add(factura);
        txtCifVendedor.setText("");
        txtRSVendedor.setText("");
        txtPeriodo.setText("");
    } else {
        JOptionPane.showMessageDialog(null,
            "No existen ventas para facturar
para este cliente en el período " + periodo,
            "Error.",
JOptionPane.ERROR_MESSAGE);
    }
} catch (Exception ex) {
    JOptionPane.showMessageDialog(null,
        ex.getMessage(), "Error.",
JOptionPane.ERROR_MESSAGE);
} finally {
    comboCliente.requestFocus();
}
}
}
});
}

/**
 * Método auxiliar para realizar comprobaciones sobre los datos introducidos en
el formulario
 *
 * @return boolean con el resultado de la validación
 */
private boolean validarFormulario() {
    if (txtCifVendedor.getText().trim().length() == 9) {
        Validador validador = new Validador();
        int i = validador.checkNif(txtCifVendedor.getText());
        if (!(i > 0)) {
            JOptionPane.showMessageDialog(this,
                "El campo CIF no tiene un formato válido.");
            txtCifVendedor.requestFocus();
            return false;
        }
    }
    if (txtCifVendedor.getText().trim().length() != 9) {
        JOptionPane.showMessageDialog(this,
            "El campo CIF del vendedor debe tener 9 caracteres.");
        txtCifVendedor.requestFocus();
        return false;
    }
    else if (txtRSVendedor.getText().trim().length() == 0) {
        JOptionPane.showMessageDialog(this,
            "El campo Razón Social no puede estar vacío.");
        txtRSVendedor.requestFocus();
        return false;
    }
    else if (txtPeriodo.getText().trim().length() == 0) {
        JOptionPane.showMessageDialog(this,
            "El campo Período Fiscal no puede estar vacío.");
    }
}
```

```
        txtPeriodo.requestFocus();  
        return false;  
    } else {  
        return true;  
    }  
}  
}
```

## FacturasPanel.java

```
package pooned.gui;

import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Observable;
import java.util.Observer;

import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.ListSelectionModel;
import javax.swing.border.EmptyBorder;

import pooned.modelos.Directorio;
import pooned.modelos.Factura;
import pooned.modelos.Facturas;
import pooned.modelos.Ventas;
import pooned.modelos.tablemodel.FacturasTM;

/**
 * Clase con el panel de mantenimiento de facturas.
 *
 * @author Francisco Javier Crespo Jiménez
 */
@SuppressWarnings("serial")
public class FacturasPanel extends JPanel implements Observer {

    private JTable tabla;
    private Facturas facturas;
    private FacturasTM modelo;
    private Directorio directorio;
    private Ventas ventas;

    /**
     * Constructor de la clase.
     *
     * @param directorio gestor de clientes.
     * @param ventas gestor de ventas.
     * @param facturas gestor de facturas.
     */
    public FacturasPanel(Directorio directorio, Ventas ventas, Facturas facturas) {
        this.ventas = ventas;
        this.facturas = facturas;
        this.directorio = directorio;
        this.ventas.addObserver(this);
        this.facturas.addObserver(this);
        this.directorio.addObserver(this);

        setBorder(new EmptyBorder(5, 5, 5, 5));
    }
}
```



```
setLayout(new BorderLayout());

JScrollPane scrollPane = new JScrollPane();
add(scrollPane, BorderLayout.CENTER);
setVisible(true);

tabla = new JTable();
tabla.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
scrollPane.setViewportView(tabla);
modelo = new FacturasTM(facturas);
tabla.setModel(modelo);

JPanel botonesPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));

JButton btnNuevaEmpresa = new JButton("Nueva Factura");
botonesPanel.add(btnNuevaEmpresa);
btnNuevaEmpresa.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        FacturaPanel facturaPanel = new FacturaPanel(facturas,
directorio, ventas);
        JDialog facturaDialog = new JDialog();
        facturaDialog.setModal(true);
        facturaDialog.setResizable(false);
        facturaDialog.setTitle("Añadir factura.");
        facturaDialog.setBounds(100, 100, 450, 300);
        facturaDialog.setContentPane(facturaPanel);
        facturaDialog.setVisible(true);
    }
});

JButton btnDetalle = new JButton("Detalle");
botonesPanel.add(btnDetalle);
btnDetalle.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Factura factura = facturas.get(tabla.getSelectedRow());
        JPanel vf = new VisorFacturaPanel(factura);
        JDialog v = new JDialog();
        v.setModal(true);
        v.setTitle("Detalle factura " + factura.getNumeroFactura());
        v.setBounds(100, 100, 800, 600);
        v.setContentPane(vf);
        v.setVisible(true);
        vf.setVisible(true);
    }
});

JButton btnEliminar = new JButton("Eliminar");
botonesPanel.add(btnEliminar);
btnEliminar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if (JOptionPane.showConfirmDialog(null, "¿Confirma la
```

```
eliminación de la factura?", "WARNING",
    JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION)
{
    try {
        facturas.remove(tabla.getSelectedRow());
    } catch (ArrayIndexOutOfBoundsException ex) {
        JOptionPane
            .showMessageDialog(
                null, "No se ha seleccionado una
factura.",
                "Error en la eliminación.",
                JOptionPane.ERROR_MESSAGE);
    } catch (IndexOutOfBoundsException ex) {
        JOptionPane.showMessageDialog(
            null, "No se ha podido eliminar la
factura.",
            "Error en la eliminación.",
            JOptionPane.ERROR_MESSAGE);
    }
}

});

add(botonesPanel, BorderLayout.SOUTH);
}

/**
 * Método que implementa el método update interfaz Observable
 * Cuando se produce un cambio en los modelos, se actualiza
 * el interfaz de usuario para reflejar los cambios en la GUI
 */
@Override
public void update(Observable o, Object arg) {
    tabla.updateUI();
}

}
```

## ParticularPanel.java

```
package poouned.gui;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.border.EmptyBorder;

import poouned.modelos.Directorio;
import poouned.modelos.Particular;

/**
 * Clase del panel para introducir nuevo cliente particular.
 *
 * @author Francisco Javier Crespo Jiménez
 */
@SuppressWarnings({ "serial", "unused" })
public class ParticularPanel extends JPanel {

    private JTextField txtNif;
    private JTextField txtNombre;
    private JTextField txtApellidos;
    private JTextField txtDomicilio;
    private Directorio directorio;

    /**
     * Constructor de la clase
     *
     * @param directorio gestor de clientes
     */
    public ParticularPanel(Directorio directorio) {
        this.directorio = directorio;
        setBounds(100, 100, 450, 300);
        setBorder(new EmptyBorder(5, 5, 5, 5));
        setLayout(null);

        JLabel lblNif = new JLabel("NIF");
        lblNif.setBounds(10, 27, 83, 14);
        add(lblNif);

        txtNif = new JTextField();
        txtNif.setBounds(115, 24, 100, 20);
        txtNif.setColumns(13);
        txtNif.requestFocus();
        add(txtNif);

        JLabel lblNombre = new JLabel("Nombre");
        lblNombre.setBounds(10, 62, 83, 14);
        add(lblNombre);
```

```
txtNombre = new JTextField();
txtNombre.setBounds(115, 59, 300, 20);
txtNombre.setColumns(10);
add(txtNombre);

JLabel lblApellidos = new JLabel("Apellidos");
lblApellidos.setBounds(10, 97, 83, 14);
add(lblApellidos);

txtApellidos = new JTextField();
txtApellidos.setBounds(115, 94, 300, 20);
txtApellidos.setColumns(10);
add(txtApellidos);

JLabel lblDomicilio = new JLabel("Domicilio");
lblDomicilio.setBounds(10, 132, 83, 14);
add(lblDomicilio);

txtDomicilio = new JTextField();
txtDomicilio.setBounds(115, 130, 300, 20);
txtDomicilio.setColumns(10);
add(txtDomicilio);

JButton btnGuardar = new JButton("Guardar");
btnGuardar.setBounds(320, 170, 89, 23);
add(btnGuardar);
btnGuardar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if (validarFormulario()) {
            try {
                Particular particular = new Particular();
                particular.setNif(txtNif.getText().toString());
                particular.setNombre(txtNombre.getText());
                particular.setApellidos(txtApellidos.getText());
                particular.setDomicilio(txtDomicilio.getText());
                directorio.add(particular);
                txtNif.setText("");
                txtNombre.setText("");
                txtApellidos.setText("");
                txtDomicilio.setText("");
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(null,
                    ex.getMessage(), "Error.",
JOptionPane.ERROR_MESSAGE);
            } finally {
                txtNif.requestFocus();
            }
        }
    }
});
}

/**
 * Método auxiliar para realizar comprobaciones sobre los datos introducidos en
 * el formulario
 */
```

```
* @return boolean con el resultado de la validación
*/
private boolean validarFormulario() {
    if (txtNif.getText().trim().length() == 0) {
        JOptionPane.showMessageDialog(this,
            "El campo NIF no puede estar vacio.");
        txtNif.requestFocus();
        return false;
    } else if (txtNif.getText().trim().length() > 9) {
        JOptionPane.showMessageDialog(this,
            "El campo NIF no puede tener más de 9 caracteres.");
        txtNif.requestFocus();
        return false;
    } else if (txtNombre.getText().trim().length() == 0) {
        JOptionPane.showMessageDialog(this,
            "El campo Nombre no puede estar vacio.");
        txtNombre.requestFocus();
        return false;
    } else if (txtApellidos.getText().trim().length() == 0) {
        JOptionPane.showMessageDialog(this,
            "El campo Apellidos no puede estar vacio.");
        txtApellidos.requestFocus();
        return false;
    } else if (txtDomicilio.getText().trim().length() == 0) {
        JOptionPane.showMessageDialog(this,
            "El campo Domicilio no puede estar vacio.");
        txtDomicilio.requestFocus();
        return false;
    } else {
        return true;
    }
}
}
```

## ProductoPanel.java

```
package pooned.gui;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.ParseException;

import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFormattedTextField;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.border.EmptyBorder;
import javax.swing.text.MaskFormatter;

import pooned.modelos.Inventario;
import pooned.modelos.Producto;
import pooned.modelos.ProductoInventariado;

/**
 * Clase para el panel de introducir nuevo producto en el inventario.
 *
 * @author Francisco Javier Crespo Jiménez
 */
@SuppressWarnings({ "serial", "unused" })
public class ProductoPanel extends JPanel {

    private JTextField txtCodigo, txtDescripcion, txtPrecio;
    private JFormattedTextField txtPrecio0, txtStock;

    private Inventario inventario;

    /**
     * Constructor de la clase.
     *
     * @param inventario gestor de productos
     */
    public ProductoPanel(Inventario inventario) {
        this.inventario = inventario;
        setBounds(100, 100, 450, 300);
        setBorder(new EmptyBorder(5, 5, 5, 5));
        setLayout(null);

        JLabel lblCodigo = new JLabel("Código");
        lblCodigo.setBounds(10, 27, 83, 14);
        add(lblCodigo);

        txtCodigo = new JTextField(13);
        txtCodigo.setBounds(115, 24, 86, 20);
        add(txtCodigo);

        JLabel lblDescripcion = new JLabel("Descripción");
        lblDescripcion.setBounds(10, 62, 83, 14);
```

```
add(lblDescripcion);

txtDescripcion = new JTextField();
txtDescripcion.setBounds(115, 59, 309, 20);
add(txtDescripcion);

JLabel lblPrecio0 = new JLabel("Precio Bruto");
lblPrecio0.setBounds(10, 97, 83, 14);
add(lblPrecio0);

try {
    MaskFormatter mascara = new MaskFormatter("###.##");
    txtPrecio0 = new JFormattedTextField(mascara);
    txtPrecio0.setValue("000.00");
    txtPrecio0.setBounds(115, 94, 86, 20);
    add(txtPrecio0);
} catch (ParseException ex) {
    JOptionPane.showMessageDialog(null, "Precio incorrecto.", "Error.",
        JOptionPane.ERROR_MESSAGE);
}

JLabel lblIvaAplicable = new JLabel("IVA Aplicable");
lblIvaAplicable.setBounds(10, 136, 83, 14);
add(lblIvaAplicable);

String[] tipoIva = { String.valueOf(Producto.IVA_SUPERREDUCIDO),
    String.valueOf(Producto.IVA_REDUCIDO),
    String.valueOf(Producto.IVA_GENERAL) };
@SuppressWarnings({ "unchecked", "rawtypes" })
JComboBox cmbIva = new JComboBox(tipoIva);
cmbIva.setSelectedIndex(2);
cmbIva.setBounds(115, 133, 86, 20);
add(cmbIva);

JLabel label = new JLabel("%");
label.setBounds(211, 136, 21, 14);
add(label);

JLabel lblPrecio = new JLabel("Precio");
lblPrecio.setBounds(10, 176, 83, 14);
add(lblPrecio);

txtPrecio = new JTextField();
txtPrecio.setEditable(false);
txtPrecio.setFocusable(false);
txtPrecio.setBounds(115, 173, 86, 20);
add(txtPrecio);

JLabel lblStock = new JLabel("Stock");
lblStock.setBounds(10, 211, 83, 14);
add(lblStock);

txtStock = new JFormattedTextField(new Integer(1));

txtStock.setBounds(115, 208, 86, 20);
add(txtStock);
```

```

        JButton btnGuardar = new JButton("Guardar");
        btnGuardar.setBounds(293, 207, 89, 23);
        add(btnGuardar);
        btnGuardar.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if (validarFormulario()) {
                    ProductoInventariado producto = new
ProductoInventariado();

                    producto.setCodigo(txtCodigo.getText().toString());
                    producto.setDescripcion(txtDescripcion.getText());
                    try {

                        producto.setPrecio0(Double.parseDouble(txtPrecio0
                            .getText()));
                    } catch (NumberFormatException ex) {
                        JOptionPane.showMessageDialog(null,
                            "Formato del campo precio
incorrecto.",
                            "Error.",
                            JOptionPane.ERROR_MESSAGE);
                    } catch (NullPointerException ex) {
                        JOptionPane.showMessageDialog(null,
                            "El campo precio no puede estar en
blanco.",
                            "Error.",
                            JOptionPane.ERROR_MESSAGE);
                    }

                    try {
                        producto.setIva(Integer.parseInt(tipoIva[cmbIva
                            .getSelectedIndex()]));
                    } catch (NumberFormatException ex) {
                        JOptionPane.showMessageDialog(null,
                            "Formato del campo precio
incorrecto.",
                            "Error.",
                            JOptionPane.ERROR_MESSAGE);
                    }

                    try {

                        producto.setStock(Integer.parseInt(txtStock.getText()));
                    } catch (NumberFormatException ex) {
                        JOptionPane.showMessageDialog(null,
                            "Formato del campo stock
incorrecto.",
                            "Error.",
                            JOptionPane.ERROR_MESSAGE);
                    }

                    try {
                        inventario.add(producto);
                        txtCodigo.setText("");
                        txtDescripcion.setText("");
                        txtPrecio0.setValue("000.00");
                    }
                }
            }
        });
    }
}
```



```
        txtStock.setText("1");
        txtCodigo.requestFocus();
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(null,
ex.getMessage(),
                                "Error.",
JOptionPane.ERROR_MESSAGE);
    }
}

});
}

}

/**
 * Método auxiliar para realizar comprobaciones sobre los datos introducidos en
el formulario
 *
 * @return boolean con el resultado de la validación
 */
private boolean validarFormulario() {
    if (txtCodigo.getText().trim().length() == 0) {
        JOptionPane.showMessageDialog(this,
                                "El campo código no puede estar vacio.");
        txtCodigo.requestFocus();
        return false;
    } else if (txtCodigo.getText().trim().length() > 13) {
        JOptionPane.showMessageDialog(this,
                                "El campo código no puede tener más de 13
caracteres.");
        txtCodigo.requestFocus();
        return false;
    } else if (txtDescripcion.getText().trim().length() == 0) {
        JOptionPane.showMessageDialog(this,
                                "El campo descripción no puede estar vacio.");
        txtDescripcion.requestFocus();
        return false;
    } else if (txtPrecio0.getText().trim().length() == 0) {
        JOptionPane.showMessageDialog(this,
                                "El campo precio bruto no puede estar vacio.");
        txtPrecio0.requestFocus();
        return false;
    } else if (txtStock.getText().trim().length() == 0) {
        JOptionPane.showMessageDialog(this,
                                "El campo stock no puede estar vacio.");
        txtStock.requestFocus();
        return false;
    } else {
        return true;
    }
}
}
```

## ProductosPanel.java

```
package pooned.gui;

import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Observable;
import java.util.Observer;

import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.ListSelectionModel;
import javax.swing.border.EmptyBorder;

import pooned.modelos.Inventario;
import pooned.modelos.tablemodel.ProductosTM;

/**
 * Clase con el panel para el mantenimiento de productos.
 *
 * @author Francisco Javier Crespo Jiménez
 */
@SuppressWarnings("serial")
public class ProductosPanel extends JPanel implements Observer {

    private JTable tabla;
    private ProductosTM modelo;
    private Inventario inventario;

    /**
     * Constructor de la clase.
     *
     * @param inventario gestor de productos
     */
    public ProductosPanel(Inventario inventario) {
        this.inventario = inventario;
        this.inventario.addObserver(this);
        setBorder(new EmptyBorder(5, 5, 5, 5));
        setLayout(new BorderLayout());

        JScrollPane scrollPane = new JScrollPane();
        add(scrollPane, BorderLayout.CENTER);
        setVisible(true);

        tabla = new JTable();
        tabla.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        scrollPane.setViewportView(tabla);
        modelo = new ProductosTM(inventario);
        tabla.setModel(modelo);
    }
}
```

```
JPanel botonesPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));

JButton btnNuevo = new JButton("Nuevo");
botonesPanel.add(btnNuevo);
btnNuevo.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        ProductoPanel productoPanel = new ProductoPanel(inventario);
        JDialog productoDialog = new JDialog();
        productoDialog.setModal(true);
        productoDialog.setResizable(false);
        productoDialog.setTitle("Añadir producto.");
        productoDialog.setBounds(100, 100, 450, 300);
        productoDialog.setContentPane(productoPanel);
        productoDialog.setVisible(true);
    }
});

JButton btnEliminar = new JButton("Eliminar");
botonesPanel.add(btnEliminar);
btnEliminar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if (JOptionPane.showConfirmDialog(null, "¿Confirma la
eliminación del producto?", "WARNING",
JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION)
{
            try {
                inventario.remove(tabla.convertRowIndexToModel(tabla.getSelectedRow()));
            } catch (ArrayIndexOutOfBoundsException ex) {
                JOptionPane
                    .showMessageDialog(
                        null, "No se ha seleccionado un
producto.",
                        "Error en la eliminación.",
                        JOptionPane.ERROR_MESSAGE);
            } catch (IndexOutOfBoundsException ex) {
                JOptionPane.showMessageDialog(
                    null, "No se ha podido eliminar el
producto.",
                    "Error en la eliminación.",
                    JOptionPane.ERROR_MESSAGE);
            }
        }
    }
});

add(botonesPanel, BorderLayout.SOUTH);
}

/**
 * Método que implementa el método update interfaz Observable
 * Cuando se produce un cambio en los modelos, se actualiza
 * el interfaz de usuario para reflejar los cambios en la GUI
 */
```

```
@Override  
public void update(Observable o, Object arg) {  
    tabla.updateUI();  
}  
}
```

## VentasPanel.java

```
package poouned.gui;

import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Observable;
import java.util.Observer;

import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.ListSelectionModel;
import javax.swing.border.EmptyBorder;

import poouned.modelos.Ticket;
import poouned.modelos.Ventas;
import poouned.modelos.tablemodel.TicketTM;

/**
 * Clase con el panel de ventas del TPV
 *
 * @author Francisco Javier Crespo Jiménez
 */
@SuppressWarnings("serial")
public class VentasPanel extends JPanel implements Observer {

    private Ventas ventas;
    protected JTable tabla;
    private TicketTM modelo;

    /**
     * Constructor de la clase.
     *
     * @param ventas gestor de ventas.
     */
    public VentasPanel(Ventas ventas) {
        this.ventas = ventas;
        this.ventas.addObserver(this);
        setBorder(new EmptyBorder(5, 5, 5, 5));
        setLayout(new BorderLayout());

        JScrollPane scrollPane = new JScrollPane();
        add(scrollPane, BorderLayout.CENTER);
        setVisible(true);

        tabla = new JTable();
        tabla.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        scrollPane.setViewportView(tabla);
        modelo = new TicketTM(ventas);
        tabla.setModel(modelo);
    }
}
```

```
JPanel botonesPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));

JButton btnDetalle = new JButton("Detalle");
botonesPanel.add(btnDetalle);
btnDetalle.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            Ticket ticket = ventas.get(tabla.getSelectedRow());
            JPanel vt = new VisorTicketPanel(ticket);
            JDialog v = new JDialog();
            v.setModal(true);
            v.setTitle("Detalle ticket " + ticket.getCodigo());
            v.setBounds(100, 100, 800, 600);
            v.setContentPane(vt);
            v.setVisible(true);
            vt.setVisible(true);
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(
                null, "Error en mostrar el detalle del
ticket.",
                "Error.", JOptionPane.ERROR_MESSAGE);
        }
    }
});

JButton btnEliminar = new JButton("Eliminar");
botonesPanel.add(btnEliminar);
btnEliminar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if (JOptionPane
            .showConfirmDialog(
                null,
                "La eliminación de una venta
implica una devolución\n"
                + "de los productos al stock.\n"
                + "¿Confirma la eliminación de la
venta?",
                "WARNING",
                JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION) {
            try {
                ventas.devolucion(tabla.getSelectedRow());
            }
            catch (ArrayIndexOutOfBoundsException ex) {
                JOptionPane
                    .showMessageDialog(
                        null, "No se ha seleccionado
una venta.",
                        "Error en la eliminación de
la venta.", JOptionPane.ERROR_MESSAGE);
            }
            catch (IndexOutOfBoundsException ex) {
                JOptionPane.showMessageDialog(
                    null, "No se ha podido
eliminar la venta.",
                    "Error en la eliminación de
la venta.", JOptionPane.ERROR_MESSAGE);
            }
        }
    }
});
```

```
        }  
        catch (IllegalArgumentException ex) {  
            JOptionPane.showMessageDialog(null,  
ex.getMessage(),  
                                "Error en la eliminación de la  
venta.", JOptionPane.ERROR_MESSAGE);  
        }  
    }  
});  
  
add(botonesPanel, BorderLayout.SOUTH);  
  
}  
  
/**  
 * Método que implementa el método update interfaz Observable Cuando se  
 * produce un cambio en los modelos, se actualiza el interfaz de usuario  
 * para reflejar los cambios en la GUI.  
 */  
@Override  
public void update(Observable o, Object arg) {  
    tabla.updateUI();  
}  
}
```

## VisorFacturaPanel.java

```
package poouned.gui;

import java.awt.BorderLayout;
import java.awt.Font;

import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

import poouned.modelos.Factura;
import poouned.modelos.ProductoVendido;
import poouned.util.Texto;

/**
 * Clase con el panel para visualizar facturas.
 *
 * @author Francisco Javier Crespo Jiménez
 */
@SuppressWarnings("serial")
public class VisorFacturaPanel extends JPanel {

    private Factura factura;

    /**
     * Constructor de la clase.
     *
     * @param factura factura para la que van a mostrar sus datos.
     */
    public VisorFacturaPanel(Factura factura) {
        this.factura = factura;
        setLayout(new BorderLayout());

        JLabel titulo = new JLabel("Factura");
        titulo.setFont(new Font("Arial Black", Font.PLAIN, 20));
        add(titulo, BorderLayout.NORTH);
        JScrollPane scrollPane = new JScrollPane();
        JTextArea txtFactura = new JTextArea(25, 55);
        txtFactura.setText(cuerpoFactura());
        txtFactura.setEditable(false);
        scrollPane.add(txtFactura);
        scrollPane.setViewportView(txtFactura);
        add(scrollPane, BorderLayout.CENTER);
    }

    /**
     * Método auxiliar para componer el string del cuerpo de la factura.
     *
     * @return cuerpo String con el cuerpo de la factura.
     */
    private String cuerpoFactura() {

        StringBuffer bfr = new StringBuffer();
```



```
        bfr.append("Vendedor: CIF " + factura.getCif() + " - " +
factura.getRazonSocial() + "\n");
        bfr.append("Cliente: " + factura.getCliente() + "\n");
        bfr.append("Factura nº " + factura.getNumeroFactura() + "Fecha: " +
factura.getFecha()+ "\n");

        bfr.append("Código\tTicket\t\tDescripción\t\tUnidades\tPrecio\tImporte\n");

        for (ProductoVendido pv : factura.getProductos()) {
            bfr.append(pv.getCodigo() + "\t" +
                pv.getTicket().getCodigo() + "\t" +
                Texto.truncar(pv.getDescripcion()) + "\t" +
                pv.getUnidades() + "\t" +
                Texto.FORMATO_DECIMAL.format(pv.getPrecio()) + "\t"
+
                Texto.FORMATO_DECIMAL.format((pv.getUnidades() *
pv.getPrecio())) + "\n");
        }

        bfr.append("\n\nTotal Factura: " +
Texto.FORMATO_DECIMAL.format(factura.getImporte()));

        return bfr.toString();
    }
}
```

## VisorTicketPanel.java

```
package pooned.gui;

import java.awt.BorderLayout;
import java.awt.Font;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import pooned.modelos.Ticket;

/**
 * Clase con el panel para visualizar tickets.
 *
 * @author Francisco Javier Crespo Jiménez
 */
@SuppressWarnings("serial")
public class VisorTicketPanel extends JPanel {

    private Ticket ticket;

    /**
     * Constructor de la clase.
     * @param factura factura para la que van a mostrar sus datos.
     */
    public VisorTicketPanel(Ticket ticket) {
        this.ticket = ticket;
        setLayout(new BorderLayout());

        JLabel titulo = new JLabel("Ticket");
        titulo.setFont(new Font("Arial Black", Font.PLAIN, 20));
        add(titulo, BorderLayout.NORTH);
        JScrollPane scrollPane = new JScrollPane();
        JTextArea txtFactura = new JTextArea(25, 55);
        txtFactura.setText(cuerpoTicket());
        txtFactura.setEditable(false);
        scrollPane.add(txtFactura);
        scrollPane.setViewportView(txtFactura);
        add(scrollPane, BorderLayout.CENTER);
    }

    /**
     * Método auxiliar para componer el string del cuerpo del ticket.
     *
     * @return cuerpo String con el cuerpo de la factura.
     */
    private String cuerpoTicket() {

        StringBuffer bfr = new StringBuffer();

        bfr.append(ticket);

        return bfr.toString();
    }
}
```

## VisorListadosPanel.java

```
package poouned.gui;

import java.awt.BorderLayout;
import java.awt.Font;

import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.border.EmptyBorder;

/**
 * Clase con el panel para visualizar los listados requeridos.
 *
 * @author Francisco Javier Crespo Jiménez
 */
@SuppressWarnings("serial")
public class VisorListadosPanel extends JPanel {

    /**
     * Constructor de la clase.
     *
     * @param titulo String con el título a mostrar en el JLabel superior.
     * @param cuerpo String con el cuerpo del listado a ser incluido en el JTextArea.
     */
    public VisorListadosPanel(String titulo, String cuerpo) {
        setLayout(new BorderLayout());
        setBorder(new EmptyBorder(10, 10, 10, 10) );

        JLabel lTitulo = new JLabel(titulo);
        lTitulo.setFont(new Font("Arial Black", Font.PLAIN, 20));
        add(lTitulo, BorderLayout.NORTH);
        JScrollPane scrollPane = new JScrollPane();
        JTextArea txtArea = new JTextArea(25, 55);
        txtArea.setText(cuerpo);
        txtArea.setEditable(false);
        scrollPane.add(txtArea);
        scrollPane.setViewportView(txtArea);
        add(scrollPane, BorderLayout.CENTER);
    }
}
```

## AbstractListModel.java

```
package poowned.modelos;

import java.io.Serializable;
import java.util.ArrayList;

import poowned.util.PersistidorArchivo;

/**
 * Esta clase abstracta se utiliza como clase padre de las clases que mantienen
 * ArrayList y deben tener la funcionalidad de importar y exportar datos.
 * A su vez, se hereda de AbstractModel que implementa el método setChanged
 * para informar a los observadores que el modelo ha cambiado sus datos.
 *
 * @author Francisco Javier Crespo Jiménez
 * @category modelos
 * @see AbstractModel
 */
public abstract class AbstractListModel extends AbstractModel implements Serializable {

    private static final long serialVersionUID = -5393390236406617476L;

    /**
     * Método que exporta un ArrayList a un archivo utilizando
     * el PersistidorArchivo
     *
     * @param list lista a exportar
     * @param file archivo en el cual se exportará la lista
     * @return boolean el resultado de la exportación
     */
    @SuppressWarnings("rawtypes")
    public boolean exportar(ArrayList list, String file) {
        PersistidorArchivo p = new PersistidorArchivo();
        try {
            p.exportar(list, file);
            return true;
        } catch (Exception e) {
            System.err.println(e.toString());
            return false;
        }
    }
}
```

## AbstractModel.java

```
package pooned.modelos;

import java.util.Observable;

/**
 * Esta clase abstracta se utiliza como clase padre de las clases que representan
 * el modelo de negocio de la aplicación.
 * A su vez, se hereda de Observable y se implementa el método setChanged
 * para informar a los observadores que el modelo ha cambiado sus datos.
 *
 * @author Francisco Javier Crespo Jiménez
 * @category modelos
 */
public abstract class AbstractModel extends Observable {

    /**
     * Método que reimplementa el método setChanged de manera que
     * cuando se produce un cambio en los modelos, se notifique
     * a los observadores que tenga asociados.
     */
    @Override
    protected synchronized void setChanged() {
        super.setChanged();
        notifyObservers();
    }
}
```

## Cliente.java

```
package poouned.modelos;

import java.io.Serializable;
import java.util.Date;
import java.util.UUID;

import com.aeat.valida.Validador;

/**
 * Esta clase abstracta representa a los clientes de la empresa.
 * En ella se mantienen los atributos comunes a los clientes
 * particulares y empresas.
 *
 * @author Francisco Javier Crespo Jiménez
 * @category modelos
 */
public abstract class Cliente extends AbstractModel implements Serializable {

    protected UUID codigo;
    protected String domicilio;
    protected Date fechaAlta;
    protected static final long serialVersionUID = 7060094205581317736L;

    /**
     * Constructor de la clase.
     */
    protected Cliente() {
        setCodigo();
        setFechaAlta();
    }

    /**
     * Constructor de la clase.
     *
     * @param codigo código único del cliente
     * @param fechaAlta fecha de alta en el sistema
     * @param domicilio domicilio del cliente
     */
    protected Cliente(UUID codigo, Date fechaAlta, String domicilio) {
        this.codigo = codigo;
        this.fechaAlta = fechaAlta;
        this.domicilio = domicilio;
    }

    /**
     * Método para obtener el id del cliente
     * Se implementa en las clases concretas que hereden de esta
     * en el caso de un particular retornará el nif
     * en el caso de una empresa retornará el cif
     *
     * @return el id del usuario
     */
    public abstract String getId();
}
```

```
/**
 * Ajusta el id del cliente
 *
 * @param id String con un identificador del cliente
 */
public abstract void setId(String id);

/**
 * Obtiene la denominación del cliente.
 * Se implementa en las clases concretas que hereden de esta
 * en el caso de un particular retornará el nombre y apellidos,
 * en el caso de una empresa retornará la razón social
 *
 * @return denominacion String con la denominación del cliente
 */
public abstract String getDenominacion();

/**
 * Ajusta la denominación del cliente
 *
 * Al no poder distinguir en el caso de clientes particulares
 * el nombre y los apellidos, esta función deja de utilizarse
 *
 * @param denominacion
 */
public abstract void setDenominacion(String denominacion);

/**
 * Obtiene el código único del cliente.
 * @return el código único del cliente.
 */
public UUID getCodigo() {
    return codigo;
}

/**
 * Crea un código único aleatorio para el cliente.
 * Llama al método setChanged para que los observadores
 * asociados, ejecuten las acciones necesarias cuando
 * este dato haya cambiado.
 */
public void setCodigo() {
    this.codigo = UUID.randomUUID();
    setChanged();
}

/**
 * Ajusta el código para el cliente.
 * Llama al método setChanged para que los observadores
 * asociados, ejecuten las acciones necesarias cuando
 * este dato haya cambiado.
 *
 */
public void setCodigo(UUID codigo) {
    this.codigo = codigo;
    setChanged();
}
```

```
}

/**
 * Obtiene el domicilio del cliente.
 * @return el domicilio del cliente.
 */
public String getDomicilio() {
    return domicilio;
}

/**
 * Ajusta el domicilio del cliente.
 * @param domicilio del cliente.
 *
 * Llama al método setChanged para que los observadores
 * asociados, ejecuten las acciones necesarias cuando
 * este dato haya cambiado.
 */
public void setDomicilio(String domicilio) {
    this.domicilio = domicilio;
    setChanged();
}

/**
 * Obtiene la fecha de alta del cliente en el sistema.
 * @return la fecha de alta del cliente en el sistema.
 *
 * Llama al método setChanged para que los observadores
 * asociados, ejecuten las acciones necesarias cuando
 * este dato haya cambiado.
 */
public Date getFechaAlta() {
    return fechaAlta;
}

/**
 * Ajusta la fecha de alta del cliente en el sistema.
 *
 * Llama al método setChanged para que los observadores
 * asociados, ejecuten las acciones necesarias cuando
 * este dato haya cambiado.
 */
public void setFechaAlta() {
    this.fechaAlta = new Date();
    setChanged();
}

/**
 * Ajusta la fecha de alta del cliente
 * @param fechaAlta fecha de alta del cliente
 *
 * Ejecuta el método setChanged para que los observadores
 * asociados, ejecuten las acciones necesarias cuando
 * este dato haya cambiado.
 */
public void setFechaAlta(Date fechaAlta) {
```



```
        this.fechaAlta = fechaAlta;
        setChanged();
    }

    /**
     * Método para imprimir el cliente.
     *
     * @return cliente String que representa el cliente.
     */
    public String toString() {
        StringBuffer bfr = new StringBuffer();

        bfr.append("Código: " + codigo + "\n");
        bfr.append("Fecha Alta: " + fechaAlta + "\n");
        bfr.append("Domicilio: " + domicilio + "\n");

        return bfr.toString();
    }

    /**
     * Método para validar los documentos de indentidad tanto de
     * particulares (nif) como de empresas (cif) proporcionado
     * como librería externa por la Agencia Tributaria.
     * @param str el documento a validar.
     * @return si el documento ha sido validado.
     * @throws IllegalArgumentException en caso de que el documento no tenga un
     formato adecuado.
     */
    protected boolean validaNif(String str) throws IllegalArgumentException {
        Validador validador = new Validador();
        int i = validador.checkNif(str);
        if (i > 0) return true;
        throw new IllegalArgumentException("El documento no tiene un formato
válido.");
    }
}
```

## Directorio.java

```
package poouned.modelos;

import java.util.ArrayList;

import poouned.util.PersistidorArchivo;

/**
 * Esta clase representa el conjunto de clientes de la empresa.
 * Internamente trabaja con ArrayList pero hace ciertas comprobaciones
 * adicionales (por ejemplo comprobar si existe el cliente antes de
 * ser insertado). Los métodos públicos se llaman del mismo modo que los
 * de ArrayList.
 *
 * @author Francisco Javier Crespo Jiménez
 * @category modelos
 * @see ArrayList
 */
@SuppressWarnings("serial")
public class Directorio extends AbstractListModel {

    private ArrayList<Cliente> clientes;
    private String archivo = "directorio.dat";

    /**
     * Constructor de la clase.
     */
    public Directorio() {
        super();
        this.clientes = new ArrayList<Cliente>();
    }

    /**
     * Constructor de la clase.
     */
    public Directorio(ArrayList<Cliente> clientes) {
        super();
        this.clientes = clientes;
    }

    /**
     * Añade un cliente al directorio
     *
     * @param cliente cliente a añadir
     * @return boolean el resultado de la operación de añadir el cliente.
     * @throws IllegalArgumentException si el identificador del cliente ya existía se
     lanza esta excepción genérica
     */
    public boolean add(Cliente cliente) throws IllegalArgumentException {
        if (findById(cliente.getId()) == null) {
            setChanged();
            return this.clientes.add(cliente);
        } else {
            throw new IllegalArgumentException("El cliente con id " +
```

```
cliente.getId() + " ya existe.");
    }
}

/**
 * Añade un ArrayList de clientes al directorio.
 * @param clientes ArrayList de clientes a añadir al directorio.
 * @return boolean el resultado de la inserción
 * @throws Exception Si existe un error en la inserción se lanza esta excepción
 */
public boolean addAll(ArrayList<Cliente> clientes) throws Exception {
    for (Cliente c : clientes) {
        try { add(c); }
        catch (Exception ex) {
            throw new Exception(ex.getMessage());
        }
    }
    return true;
}

/**
 * Obtiene un cliente del directorio
 * @param index índice del array a obtener
 * @return el cliente solicitado
 */
public Cliente get(int index) {
    return this.clientes.get(index);
}

/**
 * Obtiene un ArrayList con todos los clientes del directorio.
 * @return ArrayList con todos los clientes del directorio.
 */
public ArrayList<Cliente> getAll() {
    return clientes;
}

/**
 * Busca por el id del cliente si existe en el directorio.
 * Al ser un identificador único basta con ser encontrado una vez
 * para no tener que seguir comprobando.
 * @param id identificador a buscar.
 * @return cliente en caso de ser encontrado.
 */
public Cliente findById(String id) {
    for (Cliente item: clientes) {
        if (item.getId().equals(id)) return item;
    }
    return null;
}

/**
 * Realiza la importación de clientes al directorio.
 */
}
```

```
* Llama al método setChanged para que los observadores
* asociados, ejecuten las acciones necesarias cuando
* este dato haya cambiado.
* @return boolean verdadero si la importación tiene éxito.
* @throws Exception lanza esta excepción si se produce un problema en la
importación.
*/
@SuppressWarnings({ "rawtypes", "unchecked" })
public boolean importar() throws Exception {
    PersistidorArchivo p = new PersistidorArchivo();
    try {
        addAll((ArrayList) p.importar(this.archivo));
        setChanged();
        return true;
    } catch (Exception e) {
        throw new Exception("Error en la importación.");
    }
}

/**
 * Exporta los clientes del directorio a un archivo.
 * @return el resultado de la exportación.
 */
public boolean exportar() {
    return super.exportar(clientes, archivo);
}

/**
 * Elimina un cliente del directorio.
 *
 * Llama al método setChanged para que los observadores
 * asociados, ejecuten las acciones necesarias cuando
 * este dato haya cambiado.
 * @param cliente El cliente a eliminar.
 * @return boolean El resultado de la eliminación.
 */
public boolean remove(Cliente cliente) {
    boolean result = this.clientes.remove(cliente);
    setChanged();
    return result;
}

/**
 * Elimina un cliente del directorio.
 *
 * Llama al método setChanged para que los observadores
 * asociados, ejecuten las acciones necesarias cuando
 * este dato haya cambiado.
 * @param index índice del array del directorio donde se encuentra el cliente a
eliminar.
 * @return el resultado de la eliminación.
 */
public Cliente remove(int index) {
    Cliente item = this.clientes.remove(index);
    setChanged();
    return item;
}
```

```
}

/**
 * Devuelve el número de clientes en el directorio.
 * @return el número de clientes en el directorio.
 */
public int size() {
    return this.clientes.size();
}

/**
 * Representación en un String del directorio.
 * Recorre el directorio delegando la tarea de representar
 * cada cliente a la clase Cliente.
 *
 * @return string representación del directorio
 */
public String toString() {
    StringBuffer bfr = new StringBuffer();

    for (Cliente item: clientes) bfr.append(bfr + " " + item.toString());

    return bfr.toString();
}
}
```

## Empresa.java

```
package pooned.modelos;

import java.io.Serializable;
import java.util.Date;
import java.util.UUID;

/**
 * Esta clase representa a los clientes de tipo empresa.
 * Hereda de la clase abstracta cliente.
 *
 * @author Francisco Javier Crespo Jiménez
 * @category modelos
 * @see Cliente
 */
public class Empresa extends Cliente implements Serializable {

    private String cif;
    private String razonSocial;
    private static final long serialVersionUID = 8665724966942981167L;

    /**
     * Constructor de la clase.
     */
    public Empresa() {
        super();
    }

    /**
     * Constructor de la clase.
     */
    public Empresa(UUID codigo, Date fechaAlta, String domicilio, String razonSocial,
String cif) {
        super(codigo, fechaAlta, domicilio);
        this.cif = cif;
        this.razonSocial = razonSocial;
    }

    /**
     * Obtiene el identificador de la empresa.
     * @return el identificador de la empresa.
     */
    @Override
    public String getId() {
        return getCif();
    }

    /**
     * Obtiene la denominación de la empresa.
     * @return la denominación de la empresa.
     */
    @Override
    public String getDenominacion() {
        return razonSocial;
    }
}
```

```
/**
 * Obtiene el cif de la empresa.
 * @return el cif de la empresa.
 */
public String getCif() {
    return cif;
}

/**
 * Ajusta el cif de la empresa.
 * @param cif el cif de la empresa.
 */
public void setCif(String cif) {
    if (super.validaNif(cif.toUpperCase().trim())) {
        this.cif = cif.toUpperCase();
    }
    setChanged();
}

/**
 * Obtiene la razón social de la empresa.
 * @return la razón social de la empresa.
 */
public String getRazonSocial() {
    return razonSocial;
}

/**
 * Ajusta la razón social de la empresa.
 * @param razonSocial la razón social de la empresa.
 * Llama al método setChanged para que los observadores
 * asociados, ejecuten las acciones necesarias cuando
 * este dato haya cambiado.
 */
public void setRazonSocial(String razonSocial) {
    this.razonSocial = razonSocial;
    setChanged();
}

/**
 * Ajusta el id de la empresa.
 * @param id el id de la empresa.
 * Llama al método setChanged para que los observadores
 * asociados, ejecuten las acciones necesarias cuando
 * este dato haya cambiado.
 */
@Override
public void setId(String id) {
    this.cif = id;
    setChanged();
}

/**
 * Ajusta la denominación de la empresa.
 * @param la denominación de la empresa.
```

```
    * Llama al método setChanged para que los observadores
    * asociados, ejecuten las acciones necesarias cuando
    * este dato haya cambiado.
    */
    @Override
    public void setDenominacion(String denominacion) {
        this.razonSocial = denominacion;
        setChanged();
    }

    /**
     * Representación en texto de la empresa.
     */
    public String toString() {
        return cif + " - " + razonSocial;
    }
}
```



## Factura.java

```
package poouned.modelos;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Date;
import java.util.UUID;

/**
 * Esta clase representa las facturas.
 * Una factura se compone de los tickets de un cliente para un mismo periodo fiscal.
 *
 * @author Francisco Javier Crespo Jiménez
 * @category modelos
 */
public class Factura extends AbstractModel implements Serializable {

    private UUID numeroFactura;
    private String cif;
    private String razonSocial;
    private Date fecha;
    private Cliente cliente;
    private String periodoFiscal;
    private ArrayList<Ticket> tickets;
    private static final long serialVersionUID = 3161759639856319292L;

    /**
     * Constructor de la clase.
     */
    public Factura() {
        setNumeroFactura();
        this.fecha = new Date();
        this.tickets = new ArrayList<Ticket>();
    }

    /**
     * Constructor de la clase.
     */
    public Factura (UUID numeroFactura, String cif, String razonSocial, Date fecha,
        Cliente cliente, String periodo) {
        this.numeroFactura = numeroFactura;
        this.cif = cif;
        this.razonSocial = razonSocial;
        this.fecha = fecha;
        this.cliente = cliente;
        this.periodoFiscal = periodo;
    }

    /**
     * Obtiene el número de factura
     * @return el número de factura
     */
    public UUID getNumeroFactura() {
        return numeroFactura;
    }
}
```

```
}

/**
 * Ajusta el número de factura
 */
protected void setNumeroFactura() {
    this.numeroFactura = UUID.randomUUID();
    setChanged();
}

/**
 * Obtiene el cif de la factura
 * @return el cif de la factura
 */
public String getCif() {
    return cif;
}

/**
 * Ajusta el cif de la factura
 * @param cif el cif de la factura
 */
public void setCif(String cif) {
    this.cif = cif;
    setChanged();
}

/**
 * Obtiene la razón social de la factura
 * @return la razón social de la factura
 */
public String getRazonSocial() {
    return razonSocial;
}

/**
 * Ajusta la razón social de la factura
 * @param la razón social de la factura
 */
public void setRazonSocial(String razonSocial) {
    this.razonSocial = razonSocial;
    setChanged();
}

/**
 * Obtiene la fecha de la factura
 * @return la fecha de la factura
 */
public Date getFecha() {
    return fecha;
}

/**
 * Ajusta la fecha de la factura
 * @param fecha a ajustar
 */
```

```
public void setFecha(Date fecha) {
    this.fecha = fecha;
    setChanged();
}

/**
 * Obtiene el cliente de la factura
 * @return el cliente de la factura
 */
public Cliente getCliente() {
    return cliente;
}

/**
 * Ajusta el cliente de la factura
 * @param cliente de la factura
 */
public void setCliente(Cliente cliente) {
    this.cliente = cliente;
    setChanged();
}

/**
 * Añade un ticket a la factura
 * @param ticket a añadir
 * @return el resultado de añadir el ticket a la lista
 */
public boolean addTicket(Ticket ticket) {
    boolean result = this.tickets.add(ticket);
    setChanged();
    return result;
}

/**
 * Añade una lista de tickets
 * @param tickets a añadir
 */
public void setTickets(ArrayList<Ticket> tickets) {
    this.tickets = tickets;
    setChanged();
}

/**
 * Elimina un ticket de la lista
 * @param ticket a eliminar
 * @return el resultado de la eliminación
 */
public boolean removeTicket(Ticket ticket) {
    boolean result = this.tickets.remove(ticket);
    setChanged();
    return result;
}

/**
 * Resetear la lista de tickets
 */
```

```
    */
    public void clearTickets() {
        this.tickets.clear();
        setChanged();
    }

    /**
     * Obtiene una lista con los productos vendidos de la factura
     * @return productos incluidos en la factura
     */
    public ArrayList<ProductoVendido> getProductos() {
        ArrayList<ProductoVendido> productosFactura = new
ArrayList<ProductoVendido>();
        for (Ticket ticket : tickets) {
            for (ProductoVendido producto : ticket.getProductos()) {
                productosFactura.add(producto);
            }
        }
        return productosFactura;
    }

    /**
     * Obtiene el período fiscal de la factura
     * @return el período fiscal de la factura
     */
    public String getPeriodoFiscal() {
        return periodoFiscal;
    }

    /**
     * Ajusta el período fiscal de la factura
     * @param periodo fiscal de la factura
     */
    public void setPeriodoFiscal(String periodo) {
        this.periodoFiscal = periodo;
        setChanged();
    }

    /**
     * Obtiene el importe total de la factura
     * @return el importe total de la factura
     */
    public double getImporte() {
        double importe = 0;
        for (ProductoVendido pv : getProductos()) importe += (pv.getUnidades() *
pv.getPrecio());
        return importe;
    }

    /**
     * Representación de un string con los datos de la factura
     */
    public String toString() {
        StringBuffer bfr = new StringBuffer();

        bfr.append("FACTURA\n");
    }
}
```

```
        for (Ticket ticket: tickets) {  
            bfr.append(bfr + ticket.toString());  
        }  
        return bfr.toString();  
    }  
  
}
```

## Facturas.java

```
package pooned.modelos;

import java.util.ArrayList;
import java.util.UUID;

import pooned.util.PersistidorArchivo;

/**
 * Esta clase representa el conjunto de facturas de la empresa.
 * Internamente trabaja con ArrayList pero hace ciertas comprobaciones
 * adicionales (por ejemplo comprobar si existe la factura antes de
 * ser insertada). Los métodos públicos se llaman del mismo modo que los
 * de ArrayList.
 *
 * @author Francisco Javier Crespo Jiménez.
 * @category modelos
 * @see ArrayList
 */
@SuppressWarnings("serial")
public class Facturas extends AbstractListModel {

    private ArrayList<Factura> facturas;
    private String archivo = "facturas.dat";

    /**
     * Constructor de la clase.
     */
    public Facturas() {
        this.facturas = new ArrayList<Factura>();
    }

    /**
     * Constructor de la clase.
     */
    public Facturas(ArrayList<Factura> facturas) {
        this.facturas = facturas;
    }

    /**
     * Añadir una factura a la lista de facturas.
     * @param factura a añadir.
     * @return el resultado de añadir la factura.
     * @throws Exception.
     */
    public boolean add(Factura factura) throws Exception {
        if (findByNumeroFactura(factura.getNumeroFactura()) == null) {
            setChanged();
            return this.facturas.add(factura);
        } else {
            throw new Exception("La factura con número " +
factura.getNumeroFactura() + " ya existe.");
        }
    }
}
```

```
/**
 * Añadir una lista de facturas a la lista existente.
 * @param facturas a añadir.
 * @return el resultado de añadir la lista.
 * @throws Exception.
 */
public boolean addAll(ArrayList<Factura> facturas) throws Exception {
    for (Factura f : facturas) {
        try { add(f); }
        catch (Exception ex) {
            throw new Exception(ex.getMessage());
        }
    }
    return true;
}

/**
 * Buscar una factura por su código.
 * @param numeroFactura código de la factura.
 * @return si existe, la factura buscada.
 */
public Factura findByNumeroFactura(UUID numeroFactura) {
    for (Factura item: facturas) {
        if (item.getNumeroFactura().equals(numeroFactura)) return item;
    }
    return null;
}

/**
 * Obtiene una factura por su índice en la lista.
 * @param index el índice a recuperar.
 * @return la factura buscada.
 */
public Factura get(int index) {
    return this.facturas.get(index);
}

/**
 * Obtiene la lista completa de facturas.
 * @return la lista completa de facturas.
 */
public ArrayList<Factura> getAll() {
    return facturas;
}

/**
 * Elimina una factura de la lista de facturas.
 * @param factura a eliminar.
 * @return el resultado de la eliminación.
 */
public boolean remove(Factura factura) {
    boolean result = this.facturas.remove(factura);
    setChanged();
    return result;
}
```

```
/**
 * Elimina una factura de la lista de facturas.
 * @param index posición en la lista a eliminar.
 * @return si ha sido encontrada, la factura eliminada.
 */
public Factura remove(int index) {
    Factura item = this.facturas.remove(index);
    setChanged();
    return item;
}

/**
 * Devuelve el tamaño de la lista de facturas.
 * @return tamaño de la lista.
 */
public int size() {
    return this.facturas.size();
}

/**
 * Importa una lista de facturas desde un archivo.
 * @return el resultado de la importación.
 * @throws Exception lanzada en caso de error en la importación.
 */
@SuppressWarnings({ "rawtypes", "unchecked" })
public boolean importar() throws Exception {
    PersistidorArchivo p = new PersistidorArchivo();
    try {
        addAll((ArrayList) p.importar(this.archivo));
        setChanged();
        return true;
    } catch (Exception e) {
        throw new Exception("Error en la importación.");
    }
}

/**
 * Exporta una lista de facturas a un archivo.
 * @return el resultado de la exportación.
 */
public boolean exportar() {
    return super.exportar(facturas, archivo);
}

/**
 * Representación en un string de la lista de facturas.
 */
public String toString() {
    StringBuffer bfr = new StringBuffer();

    for (Factura item: facturas) bfr.append(bfr + " " + item.toString());

    return bfr.toString();
}
}
```



## Inventario.java

```
package poouned.modelos;

import java.util.ArrayList;

import poouned.util.PersistidorArchivo;

/**
 * Esta clase representa el conjunto de productos dados de alta en el inventario
 * de la aplicación. Internamente trabaja con ArrayList pero hace ciertas
 * comprobaciones adicionales (como comprobar si existe el código del producto
 * antes de ser insertado o implementar métodos de búsqueda por código o descripción).
 * Los métodos públicos se llaman del mismo modo que los de ArrayList.
 *
 * @author Francisco Javier Crespo Jiménez
 * @category modelos
 * @see ArrayList
 */
@SuppressWarnings("serial")
public class Inventario extends AbstractListModel {

    private ArrayList<Producto> productos;
    private String archivo = "inventario.dat";

    /**
     * Constructor de la clase.
     */
    public Inventario() {
        this.productos = new ArrayList<Producto>();
    }

    /**
     * Constructor de la clase.
     */
    public Inventario(ArrayList<Producto> productos) {
        this.productos = productos;
    }

    /**
     * Añade un producto al inventario
     * @param producto a añadir
     * @return el resultado de añadir el producto en la lista
     * @throws Exception
     */
    public boolean add(Producto producto) throws Exception {
        if (findByCodigo(producto.getCodigo()) == null) {
            setChanged();
            return this.productos.add(producto);
        } else {
            throw new Exception("El código " + producto.getCodigo() + " ya
existe.");
        }
    }
}
```

```
/**
 * Añade una lista de productos al inventario
 * @param productos a añadir
 * @return el resultado de la operación de añadir
 * @throws Exception
 */
public boolean addAll(ArrayList<Producto> productos) throws Exception {
    for (Producto p : productos) {
        try { add(p); }
        catch (Exception ex) {
            throw new Exception(ex.getMessage());
        }
    }
    return true;
}

/**
 * Obtiene un producto por su índice en el inventario.
 * @param index el índice en el inventario.
 * @return el producto si ha sido encontrado.
 */
public Producto get(int index) {
    return this.productos.get(index);
}

/**
 * Busca la existencia del código en el inventario de productos
 * @param codigo a buscar
 * @return si se encuentra el producto es devuelto
 */
public Producto findByCodigo(String codigo) {
    for (Producto item: productos) {
        if (item.getCodigo().equals(codigo)) return item;
    }
    return null;
}

/**
 * Busca la existencia de una determinada descripción en el inventario de
productos.
 * @param descripcion a buscar.
 * @return si es encontrado, se devuelve el producto.
 */
public Producto findByDescripcion(String descripcion) {
    for (Producto item: productos)
        if (item.getDescripcion().startsWith(descripcion)) return item;
    return null;
}

/**
 * Elimina un producto del inventario.
 * @param producto el producto a eliminar.
 * @return el resultado de la operación de eliminar el producto.
 */
public boolean remove(Producto producto) {
    boolean result = this.productos.remove(producto);
}
```

```
        setChanged();
        return result;
    }

    /**
     * Elimina un producto del inventario buscándolo por su índice en el inventario.
     * @param index el índice en el inventario.
     * @return si ha sido encontrado, se devuelve el producto eliminado.
     */
    public Producto remove(int index) {
        Producto item = this.productos.remove(index);
        setChanged();
        return item;
    }

    /**
     * Retorna el tamaño del inventario de productos.
     * @return
     */
    public int size() {
        return this.productos.size();
    }

    /**
     * Importa el directorio de productos desde archivo.
     * @return la operación de importar.
     * @throws Exception es lanzada cuando existe algún problema en la importación.
     */
    @SuppressWarnings({ "unchecked", "rawtypes" })
    public boolean importar() throws Exception {

        PersistidorArchivo p = new PersistidorArchivo();
        try {
            addAll((ArrayList) p.importar(this.archivo));
            setChanged();
            return true;
        } catch (Exception e) {
            throw new Exception("Error en la importación.");
        }
    }

    /**
     * Exporta el inventario de productos a archivo.
     * @return el resultado de la exportación.
     */
    public boolean exportar() {
        return super.exportar(productos, archivo);
    }

    /**
     * Genera la representación en un string del inventario de productos.
     */
    public String toString() {
        StringBuffer bfr = new StringBuffer();
        for (Producto item: productos) bfr.append(bfr + " " + item.toString());
    }
}
```

```
        return bfr.toString();  
    }  
  
}
```

## Particular.java

```
package pooned.modelos;

import java.io.Serializable;
import java.util.Date;
import java.util.UUID;

/**
 * Esta clase representa a los clientes de tipo particular.
 * Hereda de la clase abstracta cliente.
 *
 * @author Francisco Javier Crespo Jiménez
 * @category modelos
 * @see Cliente
 */
public class Particular extends Cliente implements Serializable {

    private String nif;
    private String nombre;
    private String apellidos;
    private static final long serialVersionUID = 4542820959061139504L;

    /**
     * Constructor de la clase.
     */
    public Particular() {
        super();
    }

    /**
     * Constructor de la clase.
     */
    public Particular(UUID codigo, Date fechaAlta, String domicilio, String nif,
String nombre, String apellidos) {
        super(codigo, fechaAlta, domicilio);
        this.nif = nif;
        this.nombre = nombre;
        this.apellidos = apellidos;
    }

    /**
     * Obtiene el id del particular
     * @return el id que en caso del particular es el nif
     */
    @Override
    public String getId() {
        return nif;
    }

    /**
     * Obtiene la denominación del particular.
     * @return la denominación que en el caso del particular
     * es en nombre y apellidos.
     */
    @Override
```

```
public String getDenominacion() {
    return nombre + " " + apellidos;
}

/**
 * Obtiene el nif del cliente particular
 * @return el nif del cliente particular
 */
public String getNif() {
    return nif;
}

/**
 * Ajusta el nif del cliente particular
 * @param nif del cliente particular
 */
public void setNif(String nif) {
    if (super.validaNif(nif.toUpperCase().trim())) {
        this.nif = nif.toUpperCase();
        setChanged();
    }
}

/**
 * Obtiene el nombre del particular
 * @return el nombre del particular
 */
public String getNombre() {
    return nombre;
}

/**
 * Ajusta el nombre del particular
 * @param nombre el nombre del particular
 */
public void setNombre(String nombre) {
    this.nombre = nombre;
    setChanged();
}

/**
 * Obtiene los apellidos del particular
 * @return los apellidos del particular
 */
public String getApellidos() {
    return apellidos;
}

/**
 * Ajusta los apellidos del particular
 * @param los apellidos del particular
 */
public void setApellidos(String apellidos) {
    this.apellidos = apellidos;
    setChanged();
}
```

```
/**
 * Ajusta el id del particular
 */
@Override
public void setId(String id) {
    this.nif = id;
    setChanged();
}

/**
 * Ajusta la denominación del cliente particular
 * @param denominacion la denominación del cliente particular
 */
@Override
public void setDenominacion(String denominacion) {
    if (denominacion.startsWith(nombre)) {
        apellidos = (String) denominacion.subSequence(nombre.length() + 1,
denominacion.length());

    } else {
        nombre = (String) denominacion.subSequence(0,
(denominacion.length() - apellidos.length() + 1));
    }
    setChanged();
}

/**
 * Representación en string de un cliente particular
 */
public String toString() {
    return nif + " - " + nombre + " " + apellidos;
}
}
```

## Producto.java

```
package pooned.modelos;

import java.io.Serializable;

/**
 * Esta clase abstracta representa los productos del sistema TPV Es la clase
 * padre de la que deben heredar bien los productos que se van a inventariar
 * para formar parte del stock o bien los articulos que van a formar parte de
 * una venta.
 *
 * @author Francisco Javier Crespo Jiménez
 * @category modelos
 */
public abstract class Producto extends AbstractModel implements Serializable {

    private static final long serialVersionUID = 3259352728271118932L;

    public static final int IVA_GENERAL = 21;
    public static final int IVA_REDUCIDO = 10;
    public static final int IVA_SUPERREDUCIDO = 4;

    protected String codigo;
    protected String descripcion;
    protected double precio0;
    protected int iva;

    /**
     * Asigna un código al producto
     *
     * @param codigo Código descriptivo del producto. Un número entero que puede ser
     un código de barras.
     */
    public void setCodigo(String codigo) {
        this.codigo = codigo;
        setChanged();
    }

    /**
     * Obtiene el código del producto
     *
     * @return Código descriptivo del producto. Un número entero que puede ser un
     código de barras.
     */
    public String getCodigo() {
        return this.codigo;
    }

    /**
     * Asigna una descripción al producto
     *
     * @param descripcion Descripción detallada del producto.
     */
    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }
}
```



```
        setChanged();
    }

    /**
     * Obtiene la descripción del producto
     *
     * @return Descripción detallada del producto.
     */
    public String getDescripcion() {
        return this.descripcion;
    }

    /**
     * Asigna un precio al producto
     *
     * @param precio
     *         Precio del producto sin IVA
     */
    public void setPrecio0(double precio) {
        this.precio0 = precio;
        setChanged();
    }

    /**
     * Obtiene el precio del producto sin IVA
     *
     * @return Precio del producto sin IVA
     */
    public double getPrecio0() {
        return this.precio0;
    }

    /**
     * Asigna un porcentaje en concepto de IVA
     *
     * @param iva
     *         Impuesto sobre el Valor Añadido
     */
    public void setIva(int iva) {
        this.iva = iva;
        setChanged();
    }

    /**
     * Obtiene el porcentaje de IVA
     *
     * @return Porcentaje de IVA aplicable
     */
    public int getIva() {
        return this.iva;
    }

    /**
     * Obtiene el precio final del producto IVA incluido
     *
     * @return Precio final del producto
     */
```

```
*/  
public double getPrecio() {  
    return (this.precio0 + (this.precio0 / 100 * this.iva));  
}  
  
}
```

## ProductoInventariado.java

```
package poowned.modelos;

import java.io.Serializable;

/**
 * Esta clase extiende a la clase Producto y representa los artículos que van a
 * permanecer en el inventario del establecimiento.
 *
 * @author Francisco Javier Crespo Jiménez
 * @category modelos
 * @see Producto
 */
public class ProductoInventariado extends Producto implements Serializable {

    private int stock;
    private static final long serialVersionUID = -8815403396771636987L;

    public ProductoInventariado() {

    }

    /**
     * Constructor de la clase ProductoInventariado
     * Inicializa el IVA con el 21%
     * Inicializa el stock a 0
     */
    public ProductoInventariado(String codigo, String descripcion, double precio0) {
        this.codigo = codigo;
        this.descripcion = descripcion;
        this.precio0 = precio0;
        this.iva = IVA_GENERAL;
        this.stock = 0;
    }

    /**
     * Constructor de la clase ProductoInventariado
     * Se inicializa con el iva al 21%
     */
    public ProductoInventariado(String codigo, String descripcion, double precio0,
                                int stock) {
        this(codigo, descripcion, precio0);
        this.stock = stock;
    }

    /**
     * Constructor de la clase ProductoInventariado
     */
    public ProductoInventariado(String codigo, String descripcion, double precio0,
                                int stock, int iva) {
        this(codigo, descripcion, precio0, stock);
        this.iva = iva;
    }
}
```

```
    /**
     * Asigna una cantidad de productos
     *
     * @param stock Cantidad de productos existentes
     */
    public void setStock(int stock) {
        this.stock = stock;
        setChanged();
    }

    /**
     * Obtiene el stock existente de productos
     *
     * @return Stock de productos
     */
    public int getStock() {
        return this.stock;
    }

    /**
     * Representación en string del un producto inventariado.
     */
    public String toString() {

        StringBuffer bfr = new StringBuffer();

        bfr.append( this.codigo + " - " +
                    this.descripcion + " - " +
                    this.precio0 + "€ - " +
                    this.stock + " unidades");

        return bfr.toString();
    }
}
```

## ProductoVendido.java

```
package poowned.modelos;

import java.io.Serializable;

/**
 * Esta clase extiende a la clase Producto y representa los artículos que van a
 * formar parte de una venta.
 *
 * @author Francisco Javier Crespo Jiménez
 * @category modelos
 * @see Producto
 */
public class ProductoVendido extends Producto implements Serializable {

    private int unidades;
    private Ticket ticket;
    private static final long serialVersionUID = -7445063567731297627L;

    /**
     * Constructor de la clase ProductoInventariado
     */
    public ProductoVendido(String codigo, String descripcion, double precio0, int
iva, int unidades) {
        this.codigo = codigo;
        this.descripcion = descripcion;
        this.precio0 = precio0;
        this.iva = iva;
        this.unidades = unidades;
    }

    /**
     * Asigna una cantidad de productos
     *
     * @param unidades Cantidad de productos en un concepto de venta
     */
    public void setUnidades(int unidades) {
        this.unidades = unidades;
        setChanged();
    }

    /**
     * Obtiene la cantidad de unidades vendidas
     *
     * @return Unidades en un concepto de venta
     */
    public int getUnidades() {
        return this.unidades;
    }

    /**
     * Obtiene el ticket en el que se produjo la venta
     *
     * @return ticket de la venta
     */
}
```

```
*/  
public Ticket getTicket() {  
    return ticket;  
}  
  
/**  
 * Asigna el ticket en el que se produjo la venta  
 *  
 * @param ticket de venta  
 */  
public void setTicket(Ticket ticket) {  
    this.ticket = ticket;  
    setChanged();  
}  
  
}
```

## Ticket.java

```
package pooned.modelos;

import java.io.Serializable;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;

import pooned.util.QuantityWrongException;
import pooned.util.Texto;

/**
 * Esta clase representa un ticket de venta. Internamente utiliza
 * ArrayList para agrupar objetos de tipo ProductoVendido.
 *
 * @author Francisco Javier Crespo Jiménez.
 * @category modelos
 */
public class Ticket extends AbstractModel implements Serializable {

    private String codigo;
    private Date fecha;
    private Cliente cliente;
    private ArrayList<ProductoVendido> productos;
    private static final long serialVersionUID = 1435185431313421872L;
    private boolean facturado;

    /**
     * Constructor de la clase.
     */
    public Ticket() {
        SimpleDateFormat formatoCodigo = new SimpleDateFormat("yyyyMMddHHmmss");
        this.fecha = new Date();
        this.codigo = formatoCodigo.format(fecha);
        this.productos = new ArrayList<ProductoVendido>();
        this.facturado = false;
    }

    /**
     * Obtiene el código del ticket.
     * @return el código del ticket.
     */
    public String getCodigo() {
        return codigo;
    }

    /**
     * Obtiene la fecha del ticket.
     * @return la fecha del ticket.
     */
    public Date getFecha() {
        return fecha;
    }

    /**
```

```
* Ajusta la fecha del ticket.
* @param la fecha del ticket.
*/
public void setFecha(Date fecha) {
    this.fecha = fecha;
    setChanged();
}

/**
 * Ajusta el cliente del ticket.
 * @param el cliente del ticket.
 */
public void setCliente(Cliente cliente) {
    this.cliente = cliente;
    setChanged();
}

/**
 * Obtiene el cliente del ticket.
 * @return el cliente del ticket.
 */
public Cliente getCliente() {
    return cliente;
}

/**
 * Obtiene el importe del ticket.
 * @return el importe del ticket.
 */
public double getImporte() {
    double importe = 0;
    for (ProductoVendido producto: productos)
        importe = importe + (producto.getPrecio() * producto.getUnidades())
;
    return importe;
}

/**
 * Obtiene el período fiscal del ticket.
 * @return el período fiscal del ticket.
 */
public String getPeriodoFiscal() {
    SimpleDateFormat formatoPeriodoFiscal = new SimpleDateFormat("yyyy");
    return formatoPeriodoFiscal.format(fecha);
}

/**
 * Obtiene si el ticket ha sido facturado.
 * @return si el ticket ha sido facturado.
 */
public boolean getFacturado() {
    return this.facturado;
}

/**
 * Ajusta la facturación del ticket.
```



```
* @param la facturación del ticket.
*/
public void setFacturado(boolean valor) {
    this.facturado = valor;
}

/**
 * Ajusta el ticket como facturado.
 */
public void facturar() {
    this.facturado = true;
}

/**
 * Obtiene la lista de productos del ticket.
 * @return la lista de productos del ticket.
 */
public ArrayList<ProductoVendido> getProductos() {
    return productos;
}

/**
 * Añade un producto del inventario para ser incluido en el ticket como producto
vendido.
 * @param producto el producto del inventario a incluir en el ticket.
 * @param unidades del producto a vender.
 * @throws QuantityWrongException en caso de que las unidades a vender sea
superior a las existencias.
 */
public void addProducto(ProductoInventariado producto, int unidades) throws
QuantityWrongException {

    if (producto.getStock() >= unidades) {
        ProductoVendido productoV = new ProductoVendido(
            producto.getCodigo(), producto.getDescripcion(),
producto.getPrecio(),
            producto.getIva(), unidades
        );
        productoV.setTicket(this);
        productoV.setChanged();
        this.productos.add(productoV);
        setChanged();
        producto.setStock(producto.getStock() - unidades);
        producto.setChanged();
    } else {
        throw new QuantityWrongException("Cantidad superior al stock
existente.");
    }
}

/**
 * Representación en string del ticket.
 */
public String toString() {
```

```
StringBuffer bfr = new StringBuffer();

bfr.append("ticket nº " + codigo + " fecha: " + fecha + "\n");
for(ProductoVendido articulo: this.productos) {
    bfr.append( articulo.getCodigo() + "\t" +
                articulo.getDescripcion() + "\t" +
                articulo.getUnidades() + "\t" +

    Texto.FORMATO_DECIMAL.format(articulo.getPrecio()) + "\t" +
                articulo.getIva() + "%\t" +

    Texto.FORMATO_DECIMAL.format((articulo.getUnidades() * articulo.getPrecio())) +
    "\n");
}

return bfr.toString();
}
```

## Ventas.java

```
package poouned.modelos;

import java.util.ArrayList;
import java.util.Date;

import poouned.util.PersistidorArchivo;

/**
 * Esta clase se utiliza para agrupar los tickets generados por las ventas desde caja.
 *
 * @author Francisco Javier Crespo Jiménez
 * @category modelos
 */
@SuppressWarnings("serial")
public class Ventas extends AbstractListModel {

    private Inventario inventario;
    private ArrayList<Ticket> tickets;
    private String archivo = "ventas.dat";

    /**
     * Constructor de la clase.
     */
    public Ventas(Inventario inventario) {
        this.inventario = inventario;
        this.tickets = new ArrayList<Ticket>();
    }

    /**
     * Constructor de la clase.
     */
    public Ventas(ArrayList<Ticket> tickets) {
        this.tickets = tickets;
    }

    /**
     * Añadir un ticket a las ventas
     * @param ticket el ticket a añadir.
     * @return el resultado de la operación de añadir el ticket.
     * @throws Exception se lanza cuando ya existe el ticket que se pretende
    introducir.
     */
    public boolean add(Ticket ticket) throws Exception {
        if (findByCodigo(ticket.getCodigo()) == null) {
            setChanged();
            return this.tickets.add(ticket);
        } else {
            throw new Exception("El ticket con código " + ticket.getCodigo() +
" ya existe.");
        }
    }

    /**
```

```
* Añade una lista de tickets a las ventas.
* @param tickets la lista de tickets a añadir.
* @return el resultado de la operación de añadir la lista de tickets.
* @throws Exception si el método add lanza una excepción de re-lanza para
informar que
* la operación no tuvo éxito.
*/
public boolean addAll(ArrayList<Ticket> tickets) throws Exception {
    for (Ticket t : tickets) {
        try { add(t); }
        catch (Exception ex) {
            throw new Exception(ex.getMessage());
        }
    }
    return true;
}

/**
 * Busca en la lista de ventas, un ticket por su código.
 * @param codigo el código a buscar.
 * @return si ha sido encontrado el ticket, se devuelve.
 */
public Ticket findByCodigo(String codigo) {
    for (Ticket item: tickets) {
        if (item.getCodigo().equals(codigo)) return item;
    }
    return null;
}

/**
 * Obtiene un ticket por su índice en la lista de ventas.
 * @param index el índice a buscar.
 * @return si es encontrado el ticket, se devuelve.
 */
public Ticket get(int index) {
    return this.tickets.get(index);
}

/**
 * Obtiene la lista de tickets completa.
 * @return la lista de tickets completa.
 */
public ArrayList<Ticket> getAll() {
    return tickets;
}

/**
 * Obtiene una lista de tickets que no han sido facturados previamente.
 * @param cliente el cliente al que se pretenden facturar sus ventas.
 * @param periodo el período sobre el cual se van a facturar sus ventas.
 * @return la lista de tickets para un cliente y un período de tiempo
facturables.
*/
public ArrayList<Ticket> getFacturables(Cliente cliente, String periodo) {
    ArrayList<Ticket> ticketsFact = new ArrayList<Ticket>();
    for (Ticket ticket : tickets) {
```

```
        if (ticket.getCliente().getCodigo().equals(cliente.getCodigo())
            && (ticket.getPeriodoFiscal().equals(periodo))
            && (ticket.getFacturado() == false)) {
            ticketsFact.add(ticket);
        }
    }
    return ticketsFact;
}

/**
 * Obtiene la lista de clientes que han realizado compras y por tanto tiene
 tickets en la lista de ventas.
 * @return la lista de clietnes.
 */
public ArrayList<Cliente> getClientes() {
    ArrayList<Cliente> clientes = new ArrayList<Cliente>();
    for (Ticket ticket : tickets) {
        Cliente cliente = ticket.getCliente();
        if (findById(clientes, cliente.getId()) == null) {
            clientes.add(cliente);
        }
    }
    return clientes;
}

/**
 * Obtiene un cliente en la lista de clientes.
 * @param clientes la lista de clientes.
 * @param id el id a buscar.
 * @return si es encontrado, el cliente que se buscaba.
 */
private Cliente findById(ArrayList<Cliente> clientes, String id) {
    for (Cliente item: clientes) {
        if (item.getId().equals(id)) return item;
    }
    return null;
}

/**
 * Obtiene los tickets entre un intervalo de tiempo.
 * Este método sirve de apoyo para los métodos que
 * generan listados basados en períodos de tiempo.
 * @param fechaInicial Fecha inicial de búsqueda.
 * @param fechaFinal Fecha final de búsqueda.
 * @return
 */
public ArrayList<Ticket> getIntervalo(Date fechaInicial, Date fechaFinal) {
    ArrayList<Ticket> ticketsInterval = new ArrayList<Ticket>();
    for (Ticket ticket : tickets) {
        if (ticket.getFecha().after(fechaInicial) &&
            ticket.getFecha().before(fechaFinal))
            ticketsInterval.add(ticket);
    }
    return ticketsInterval;
}
```

```
/**
 * Obtiene las ventas en un intervalo de tiempo para un cliente concreto.
 * @param fechaInicial la fecha inicial del intervalo de tiempo.
 * @param fechaFinal la fecha final del intervalo de tiempo.
 * @param cliente el cliente sobre el que se buscan sus compras.
 * @return la lista con las ventas realizadas en un intervalo de tiempo a un
cliente.
 */
public ArrayList<Ticket> getIntervaloCliente(Date fechaInicial, Date fechaFinal,
Cliente cliente) {
    ArrayList<Ticket> ticketAux = new ArrayList<Ticket>();
    ArrayList<Ticket> ticketsIntervalo = getIntervalo(fechaInicial,
fechaFinal);
    for (Ticket ticket : ticketsIntervalo) {
        if (ticket.getCliente().getCodigo().equals(cliente.getCodigo())) {
            ticketAux.add(ticket);
        }
    }
    return ticketAux;
}

/**
 * Se elimina un ticket de la lista de ventas.
 * @param el ticket a eliminar.
 * @return el resultado de la operación de eliminar un ticket.
 */
public boolean remove(Ticket ticket) {
    boolean result = this.tickets.remove(ticket);
    setChanged();
    return result;
}

/**
 * Se elimina un ticket de la lista de ventas por su número de índice.
 * @param el índice a buscar.
 * @return el resultado la operación de eliminar el ticket de la lista de ventas.
 */
public Ticket remove(int index) {
    Ticket item = this.tickets.remove(index);
    setChanged();
    return item;
}

/**
 * Realiza una devolución de una venta que consiste en eliminar el ticket de la
lista de ventas
 * y reincorporar los artículos al stock. La condición es que dicha venta no haya
sido facturada previamente.
 * @param index el índice del ticket para realizar la devolución
 * @throws IllegalArgumentException
 */
public void devolucion(int index) throws IllegalArgumentException {
    Ticket ticket = this.tickets.get(index);
    if (ticket.getFacturado() == false) {
        for (ProductoVendido pv : ticket.getProductos()) {
```

```
        ProductoInventariado pi = (ProductoInventariado)
inventario.findByCodigo(pv.getCodigo());
        pi.setStock(pi.getStock() + pv.getUnidades());
    }
    this.tickets.remove(index);
    setChanged();
} else {
    throw new IllegalArgumentException("No se puede devolver una venta
facturada.");
}
}

/**
 * Devuelve el tamaño de la lista de ventas.
 * @return el tamaño de la lista de ventas.
 */
public int size() {
    return this.tickets.size();
}

/**
 * Realiza la importación de la lista de ventas desde archivo.
 * @return el resultado de la importación.
 * @throws Exception si ha tenido éxito la operación de importar.
 */
@SuppressWarnings({ "rawtypes", "unchecked" })
public boolean importar() throws Exception {
    PersistidorArchivo p = new PersistidorArchivo();
    try {
        addAll((ArrayList) p.importar(this.archivo));
        setChanged();
        return true;
    } catch (Exception e) {
        throw new Exception("Error en la importación.");
    }
}

/**
 * Realiza la exportación de la lista de ventas hacia archivo.
 * @return el resultado de la exportación.
 */
public boolean exportar() {
    return super.exportar(tickets, archivo);
}

/**
 * Representación como string de la lista de ventas.
 */
public String toString() {
    StringBuffer bfr = new StringBuffer();
    for (Ticket item: tickets) bfr.append(bfr + " " + item.toString());
    return bfr.toString();
}

}
```

## CientesTM.java

```
package poouned.modelos.tablemodel;

import java.util.Date;

import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;

import poouned.modelos.Directorío;
import poouned.modelos.Cliente;
import poouned.modelos.Particular;

/**
 * Clase que implementa y concreta DefaultTableModel para ser utilizado como
 * modelo por los objetos JTable que mantienen los datos de los clientes.
 *
 * @author Francisco Javier Crespo Jiménez
 * @category modelos.tablemodel
 */
@SuppressWarnings("serial")
public class CientesTM extends DefaultTableModel {

    private Directorío clientes;

    /**
     * Constructor de la clase
     * @param clientes el directorío de clientes
     */
    public CientesTM(Directorío clientes) {
        this.clientes = clientes;
    }

    String[] columnas = { "Código", "Identificación", "Denominación", "Domicilio",
"Fecha Alta" };

    /**
     * Obtiene el número de columnas
     */
    @Override
    public int getColumnCount() {
        return columnas.length;
    }

    /**
     * Obtiene el número de filas.
     */
    @Override
    public int getRowCount() {
        if (clientes == null) {
            return 0;
        }

        return clientes.size();
    }
}
```



```
/**
 * Obtiene el nombre de las columnas.
 */
@Override
public String getColumnName(int col) {
    return columnas[col];
}

/**
 * Obtiene el valor de la combinación fila y columna.
 */
@Override
public Object getValueAt(int fila, int columna) {
    Cliente cliente = (Cliente) clientes.get(fila);
    switch (columna) {
        case 0: return cliente.getCodigo();
        case 1: return cliente.getId();
        case 2: return cliente.getDenominacion();
        case 3: return cliente.getDomicilio();
        case 4: return cliente.getFechaAlta();
        default: return null;
    }
}

/**
 * Devuelve si la combinación de fila y columna es editable.
 */
@Override
public boolean isCellEditable(int fila, int columna) {
    if (columna == 0 || columna == 4) {
        return false;
    } else {
        return true;
    }
}

/**
 * Ajusta el valor del objeto en la fila y columna indicada.
 */
@Override
public void setValueAt(Object valor, int fila, int columna) {
    Cliente cliente = (Cliente) clientes.get(fila);
    switch (columna) {
        case 1: cliente.setId(valor.toString()); break;
        case 2: {
            if (cliente.getClass() == pooured.modelos.Particular.class) {
                String nombreNuevo = JOptionPane.showInputDialog("Introduce
el nombre");
                String apellidosNuevos =
JOptionPane.showInputDialog("Introduce los apellidos.");
                ((Particular) cliente).setNombre(nombreNuevo);
                ((Particular) cliente).setApellidos(apellidosNuevos);
                break;
            } else {
                cliente.setDenominacion(valor.toString()); break;
            }
        }
    }
}
```

```
        }  
        case 3: cliente.setDomicilio(valor.toString()); break;  
    }  
}  
  
/**  
 * Devuelve la clase de la columna indicada.  
 */  
@Override  
public Class<?> getColumnClass(int columnIndex) {  
    switch (columnIndex) {  
        case 4: return Date.class;  
        default: return String.class;  
    }  
}  
}
```

## FacturasTM.java

```
package pooned.modelos.tablemodel;

import javax.swing.table.DefaultTableModel;

import pooned.modelos.Factura;
import pooned.modelos.Facturas;
import pooned.util.Numericas;

/**
 * Clase que implementa y concreta DefaultTableModel para ser utilizado como
 * modelo por los objetos JTable que mantienen los datos de las facturas.
 *
 * @author Francisco Javier Crespo Jiménez
 * @category modelos.tablemodel
 */
@SuppressWarnings("serial")
public class FacturasTM extends DefaultTableModel {

    private Facturas facturas;

    /**
     * Constructor de clase
     * @param facturas
     */
    public FacturasTM(Facturas facturas) {
        this.facturas = facturas;
    }

    String[] columnas = { "Factura Nº", "Cliente", "Importe" };

    /**
     * Obtiene el número de columnas
     */
    @Override
    public int getColumnCount() {
        return columnas.length;
    }

    /**
     * Obtiene el número de filas.
     */
    @Override
    public int getRowCount() {
        if (facturas == null) {
            return 0;
        }
        return facturas.size();
    }

    /**
     * Obtiene el nombre de las columnas.
     */
    @Override
```

```
public String getColumnName(int col) {
    return columnas[col];
}

/**
 * Obtiene el valor de la combinación fila y columna.
 */
@Override
public Object getValueAt(int fila, int columna) {
    Factura factura = (Factura) facturas.get(fila);
    switch (columna) {
        case 0: return factura.getNumeroFactura();
        case 1: return factura.getCliente().toString();
        case 2: return Numericas.redondeo(factura.getImporte());
        default: return null;
    }
}

/**
 * Devuelve si la combinación de fila y columna es editable.
 */
@Override
public boolean isCellEditable(int fila, int columna) {
    return false;
}

/**
 * Devuelve la clase de la columna indicada.
 */
@Override
public Class<?> getColumnClass(int columnIndex) {
    switch (columnIndex) {
        case 2: return Double.class;
        default: return String.class;
    }
}
}
```

## ProductosROTM.java

```
package pooured.modelos.tablemodel;

import pooured.modelos.Inventario;

/**
 * Clase que implementa y concreta DefaultTableModel para ser utilizado como
 * modelo por los objetos JTable que muestran los datos de los productos pero
 * no pueden editarlos.
 * Esta clase hereda de ProductosTM para hacer no editables las celdas de la tabla.
 *
 * @author Francisco Javier Crespo Jiménez
 * @category modelos.tablemodel
 * @see ProductosTM
 */
@SuppressWarnings("serial")
public class ProductosROTM extends ProductosTM {

    /**
     * Constructor de la clase
     * @param el inventario de productos
     */
    public ProductosROTM(Inventario productos) {
        super(productos);
    }

    /**
     * Devuelve si la combinación de fila y columna es editable.
     */
    @Override
    public boolean isCellEditable(int fila, int columna) {
        return false;
    }
}
```

## ProductosTM.java

```
package poouned.modelos.tablemodel;

import javax.swing.table.DefaultTableModel;

import poouned.modelos.Inventario;
import poouned.modelos.ProductoInventariado;
import poouned.util.Numericas;

/**
 * Clase que implementa y concreta DefaultTableModel para ser utilizado como
 * modelo por los objetos JTable que mantienen los datos de los productos.
 *
 * @author Francisco Javier Crespo Jiménez
 * @category modelos.tablemodel
 */
@SuppressWarnings("serial")
public class ProductosTM extends DefaultTableModel {

    private Inventario productos;

    /**
     * Constructor de la clase
     * @param productos
     */
    public ProductosTM(Inventario productos) {
        this.productos = productos;
    }

    String[] columnas = { "Código", "Descripción", "Precio Bruto", "IVA",
        "Precio IVA", "Stock" };

    /**
     * Obtiene el número de columnas
     */
    @Override
    public int getColumnCount() {
        return columnas.length;
    }

    /**
     * Obtiene el número de filas.
     */
    @Override
    public int getRowCount() {

        if (productos == null) {
            return 0;
        }

        return productos.size();
    }

    /**
     * Obtiene el nombre de las columnas.
     */
}
```

```
    */
    @Override
    public String getColumnName(int col) {
        return columnas[col];
    }

    /**
     * Obtiene el valor de la combinación fila y columna.
     */
    @Override
    public Object getValueAt(int fila, int columna) {
        ProductoInventariado producto = (ProductoInventariado)
productos.get(fila);
        switch (columna) {
            case 0: return producto.getCodigo();
            case 1: return producto.getDescripcion();
            case 2: return Numericas.redondeo(producto.getPrecio0());
            case 3: return producto.getIva();
            case 4: return Numericas.redondeo(producto.getPrecio());
            case 5: return producto.getStock();
            default: return null;
        }
    }

    /**
     * Devuelve si la combinación de fila y columna es editable.
     */
    @Override
    public boolean isCellEditable(int fila, int columna) {
        if (columna == 0 || columna == 4) {
            return false;
        } else {
            return true;
        }
    }

    /**
     * Ajusta el valor del objeto en la fila y columna indicada.
     */
    @Override
    public void setValueAt(Object valor, int fila, int columna) {
        ProductoInventariado producto = (ProductoInventariado)
productos.get(fila);
        switch (columna) {
            case 0: producto.setCodigo(valor.toString()); break;
            case 1: producto.setDescripcion(valor.toString()); break;
            case 2: producto.setPrecio0(Double.parseDouble(valor.toString())); break;
            case 3: producto.setIva(Integer.parseInt(valor.toString())); break;
            case 5: producto.setStock(Integer.parseInt(valor.toString())); break;
        }
    }

    /**
     * Devuelve la clase de la columna indicada.
     */
    @Override
```

```
public Class<?> getColumnClass(int columnIndex) {  
    switch (columnIndex) {  
        case 2: return Double.class;  
        case 3: return Integer.class;  
        case 4: return Double.class;  
        case 5: return Integer.class;  
        default: return String.class;  
    }  
}  
}
```



## ProductosVendidosTM.java

```
package poouned.modelos.tablemodel;

import java.util.ArrayList;

import javax.swing.table.DefaultTableModel;

import poouned.modelos.ProductoVendido;
import poouned.util.Numericas;

/**
 * Clase que implementa y concreta DefaultTableModel para ser utilizado como
 * modelo por los objetos JTable que mantienen los datos de los productos vendidos.
 *
 * @author Francisco Javier Crespo Jiménez.
 * @category modelos.tablemodel
 */
@SuppressWarnings("serial")
public class ProductosVendidosTM extends DefaultTableModel {

    private ArrayList<ProductoVendido> ventas;

    /**
     * Constructor de la clase.
     * @param lista de ventas.
     */
    public ProductosVendidosTM(ArrayList<ProductoVendido> ventas) {
        this.ventas = ventas;
    }

    String[] columnas = { "Código", "Descripción", "unidades", "Precio Unitario",
        "IVA", "Importe" };

    /**
     * Obtiene el número de columnas.
     */
    @Override
    public int getColumnCount() {
        return columnas.length;
    }

    /**
     * Obtiene el número de filas.
     */
    @Override
    public int getRowCount() {
        if (ventas == null) {
            return 0;
        }
        return ventas.size();
    }

    /**
     * Obtiene el nombre de las columnas.
     */
}
```

```
    */
    @Override
    public String getColumnName(int col) {
        return columnas[col];
    }

    /**
     * Ajusta el valor del objeto en la fila y columna indicada.
     */
    @Override
    public Object getValueAt(int fila, int columna) {
        ProductoVendido producto = (ProductoVendido) ventas.get(fila);
        switch (columna) {
            case 0: return producto.getCodigo();
            case 1: return producto.getDescripcion();
            case 2: return producto.getUnidades();
            case 3: return Numericas.redondeo(producto.getPrecio());
            case 4: return producto.getIva();
            case 5: return Numericas.redondeo(producto.getPrecio() *
producto.getUnidades());
            default: return null;
        }
    }

    /**
     * Devuelve si la combinación de fila y columna es editable.
     */
    @Override
    public boolean isCellEditable(int fila, int columna) {
        return false;
    }

    /**
     * Devuelve la clase de la columna indicada.
     */
    @Override
    public Class<?> getColumnClass(int columnIndex)
    {
        switch (columnIndex)
        {
            case 2: return Integer.class;
            case 3: return Double.class;
            case 4: return Integer.class;
            case 5: return Double.class;
            default: return String.class;
        }
    }
}
```

## TicketTM.java

```
package poouned.modelos.tablemodel;

import java.util.Date;

import javax.swing.table.DefaultTableModel;

import poouned.modelos.Ticket;
import poouned.modelos.Ventas;
import poouned.util.Numericas;

/**
 * Clase que implementa y concreta DefaultTableModel para ser utilizado como
 * modelo por los objetos JTable que mantienen los datos de los tickets.
 *
 * @author Francisco Javier Crespo Jiménez.
 */
@SuppressWarnings("serial")
public class TicketTM extends DefaultTableModel {

    private Ventas ventas;

    /**
     * Constructor de la clase.
     * @param lista de ventas.
     */
    public TicketTM(Ventas ventas) {
        this.ventas = ventas;
    }

    String[] columnas = { "Código", "Fecha", "Cliente", "Importe", "Facturado" };

    /**
     * Obtiene el número de columnas.
     */
    @Override
    public int getColumnCount() {
        return columnas.length;
    }

    /**
     * Obtiene el número de filas.
     */
    @Override
    public int getRowCount() {
        if (ventas == null) {
            return 0;
        }

        return ventas.size();
    }

    /**
     * Obtiene el nombre de las columnas.
     */
}
```

```
    */
    @Override
    public String getColumnName(int col) {
        return columnas[col];
    }

    /**
     * Obtiene el valor de la combinación fila y columna.
     */
    @Override
    public Object getValueAt(int fila, int columna) {
        Ticket venta = (Ticket) ventas.get(fila);
        switch (columna) {
            case 0: return venta.getCodigo();
            case 1: return venta.getFecha();
            case 2: return venta.getCliente();
            case 3: return Numericas.redondeo(venta.getImporte());
            case 4: return venta.getFacturado();
            default: return null;
        }
    }

    /**
     * Devuelve si la combinación de fila y columna es editable.
     */
    @Override
    public boolean isCellEditable(int fila, int columna) {
        return false;
    }

    /**
     * Ajusta el valor del objeto en la fila y columna indicada.
     */
    @Override
    public Class<?> getColumnClass(int columnIndex) {
        switch (columnIndex) {
            case 1: return Date.class;
            case 3: return Double.class;
            case 4: return Boolean.class;
            default: return String.class;
        }
    }
}
}
```

## Listados.java

```
package poouned.util;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.Date;

import poouned.modelos.Cliente;
import poouned.modelos.ProductoVendido;
import poouned.modelos.Ticket;
import poouned.modelos.Ventas;

/**
 * Clase con métodos para implementar los listados requeridos en el enunciado.
 *
 * @author Francisco Javier Crespo Jiménez.
 * @category util
 */
public class Listados {

    private Ventas ventas;

    /**
     * Constructor de la clase
     * @param gestor de ventas
     */
    public Listados(Ventas ventas) {
        this.ventas = ventas;
    }

    /**
     * Listado 1
     *
     * Ventas realizadas en un intervalo de tiempo determinado agrupadas estas
     * por clientes.
     *
     * @param fechaInicial la fecha inicial de búsqueda.
     * @param fechaFinal la fecha final de búsqueda.
     * @return el String con el cuerpo del listado.
     */
    public String getIntervaloGroupByClientes(Date fechaInicial, Date fechaFinal) {
        StringBuffer bfr = new StringBuffer();
        for (Cliente cliente : ventas.getClientes()) {
            bfr.append("Cliente: " + cliente + "\n");
            for (Ticket ticket : ventas.getIntervaloCliente(fechaInicial,
                fechaFinal, cliente)) {
                for (ProductoVendido pv : ticket.getProductos()) {
                    bfr.append(pv.getCodigo());
                    bfr.append(" - " + pv.getDescripcion());
                    bfr.append(" - " + pv.getUnidades() + " unidades");
                    bfr.append(" - "
                        +
```

```
Texto.FORMATO_DECIMAL.format(pv.getPrecio())
                                + "\n");
        }
    }
    bfr.append("\n");
}
return bfr.toString();
}

/**
 * Listado 2
 *
 * @param fechaInicial la fecha inicial de búsqueda.
 * @param fechaFinal la fecha final de búsqueda.
 * @param cliente el cliente del cual se buscan las ventas realizadas.
 * @return el String con el cuerpo del listado.
 */
public String getIntervaloCliente(Date fechaInicial, Date fechaFinal, Cliente
cliente) {
    StringBuffer bfr = new StringBuffer();
    ArrayList<Ticket> tickets = ventas.getIntervaloCliente(fechaInicial,
        fechaFinal, cliente);
    bfr.append("Cliente: " + cliente + "\n");
    for (Ticket ticket : tickets) {
        for (ProductoVendido pv : ticket.getProductos()) {
            bfr.append(pv.getCodigo());
            bfr.append(" - " + pv.getDescripcion());
            bfr.append(" - " + pv.getUnidades() + " unidades");
            bfr.append(" - "
                +
                Texto.FORMATO_DECIMAL.format(pv.getPrecio())
                + "\n");
        }
    }
    return bfr.toString();
}

/**
 * Listado 3
 *
 * @param fechaInicial la fecha inicial de búsqueda.
 * @param fechaFinal la fecha final de búsqueda.
 * @return el String con el cuerpo del listado.
 */
public String rankingProductosIntervalo(Date fechaInicial, Date fechaFinal) {
    ArrayList<ProductoVendido> ranking = new ArrayList<ProductoVendido>();
    ArrayList<Ticket> ventasIntervalo = ventas.getIntervalo(fechaInicial,
fechaFinal);
    for (Ticket ticket : ventasIntervalo) {
        for (ProductoVendido pv : ticket.getProductos()) {
            for (ProductoVendido pr : ranking) {
                if (pr.getCodigo().equals(pv.getCodigo())) {
```

```
        pr.setUnidades(pr.getUnidades() + pv.getUnidades());
    }
    ranking.add(pv);
}

Collections.sort(ranking, new Comparator<ProductoVendido>() {
    @Override
    public int compare(ProductoVendido p1, ProductoVendido p2) {
        return (new Integer(p2.getUnidades()).compareTo(new
Integer(p1.getUnidades())));
    }
});

StringBuffer bfr = new StringBuffer();
for (ProductoVendido pv : ranking) {
    bfr.append(pv.getCodigo());
    bfr.append(" - " + pv.getDescripcion());
    bfr.append(" - " + pv.getUnidades() + " unidades");
    bfr.append(" - " + Texto.FORMATO_DECIMAL.format(pv.getPrecio()) +
"\n");
}
return bfr.toString();
}
}
```

## Numericas.java

```
package pooned.util;

/**
 * Clase con utilidad para el redondeo de los precios de los productos
 *
 * @author Francisco Javier Crespo Jiménez
 * @category util
 */
public class Numericas {

    /**
     * Efectúa el redondeo de un double con 2 cifras decimales
     *
     * @param val cifra a redondear
     * @return la cifra redondeada con 2 cifras decimales
     */
    public static double redondeo(double val) {
        return Math.round(val * 100.0) / 100.0;
    }
}
```



## Persistible.java

```
package pooned.util;

import java.util.ArrayList;

/**
 * Interfaz que debe implementar la utilidad que haga la persistencia de los
 * datos indicados en el enunciado.
 *
 * @author Francisco Javier Crespo Jiménez
 * @category util
 */
public interface Persistible {

    /**
     * Método para realizar la exportación de un Object genérico a un destino
     *
     * @param objeto de la clase Object a exportar
     * @param destino destino de la exportación
     * @throws Exception Lanza una excepción genérica en caso de problema en la
    exportación
     */
    public void exportar(Object objeto, String destino) throws Exception;

    /**
     * Método para realizar la importación desde un origen a un ArrayList de objetos
     *
     * @param origen
     * @return ArrayList<Object> ArrayList de objetos importados
     * @throws Exception Lanza una excepción genérica si falla la importación.
     */
    public ArrayList<Object> importar(String origen) throws Exception;

}
```

## PersistidorArchivo.java

```
package poowned.util;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;

/**
 * Esta clase es una utilidad que puede ser usada por los objetos que vayan a ser
 * persistidos a archivos en la aplicación.
 * @author Francisco Javier Crespo Jiménez.
 */
public class PersistidorArchivo implements Persistible {
    /**
     * Realiza la exportación a archivo del objeto pasado como parámetro.
     * @param objeto El objeto a exportar.
     * @param archivo El archivo de destino.
     * @throws Exception lanzada si el proceso falla.
     */
    public void exportar(Object objeto, String archivo) throws Exception {
        try {
            FileOutputStream fos= new FileOutputStream(archivo);
            ObjectOutputStream oos= new ObjectOutputStream(fos);
            oos.writeObject(objeto);
            oos.close();
            fos.close();
        } catch (IOException e) {
            throw new Exception("No se pudo exportar.");
        }
    }

    /**
     * Realiza la importación hacia archivo de un array de objetos.
     * @param archivo el archivo a importar.
     * @throws Exception lanzada si el proceso de importación no fuese exitoso.
     */
    @SuppressWarnings("unchecked")
    public ArrayList<Object> importar(String archivo) throws Exception {
        ArrayList<Object> tmplist = new ArrayList<Object>();
        try {
            FileInputStream fis = new FileInputStream(archivo);
            ObjectInputStream ois = new ObjectInputStream(fis);
            tmplist = (ArrayList<Object>) ois.readObject();
            ois.close();
            fis.close();
        } catch (IOException | ClassNotFoundException e) {
            throw new Exception("No se pudo importar. " + e.getMessage() + "\n" +
e.getStackTrace());
        }
        return tmplist;
    }
}
```

## QuantityWrongException.java

```
package poouned.util;

/**
 * Clase que define una excepción personalizada en caso de incluir una cantidad
 * de artículos en una venta superior a lo existente en el stock. Se implementa
 * a modo de demostración de esta técnica ya que se podría haber hecho uso de
 * algunas excepciones ya existentes en Java como IllegalArgumentException o
 * InvalidParameterException.
 *
 * @author Francisco Javier Crespo Jiménez.
 * @category util
 */
@SuppressWarnings("serial")
public class QuantityWrongException extends Exception {
    /**
     * Excepción lanzada cuando la cantidad de productos es superior a la existente
     * en el stock de la tienda.
     * @param msg el mensaje de respuesta a devolver por la excepción.
     */
    public QuantityWrongException(String msg) {
        super(msg);
    }
}
```

## Texto.java

```
package poowned.util;

import java.text.DecimalFormat;

/**
 * Clase con utilidad para dar formato a algunas salidas de texto
 *
 * @author Francisco Javier Crespo Jiménez
 * @category util
 */
public class Texto {

    public static final DecimalFormat FORMATO_DECIMAL = new
DecimalFormat("####0.##");

    private static int TAMANIO_MAX = 24;

    /**
     * Método para truncar los textos superiores a TAMANIO_MAX truncando el String y
     * añadiendo los puntos suspensivos que indican que el texto ha sido truncado.
     *
     * @param str el string a truncar.
     * @return es string truncado a TAMANIO_MAX número de caracteres.
     */
    public static String truncar(String str) {
        if (str.length() > TAMANIO_MAX)
            return str.substring(0, TAMANIO_MAX - 3) + "...";
        else {
            return String.format("%1$-" + TAMANIO_MAX + "s", str);
        }
    }
}
```

