Sistema gestión almacén

SISTEMA INTEGRADO DE GESTIÓN DE UNA FABRICA ARTESANA DE MUEBLES. PRÁCTICA DE PROGRAMACION ORIENTADA A OBJETOS CURSO 2019-2020

Iván Portillo Pérez,
correo: ipp_1981@hotmail.com , telf.: 682177179

ÍNDICE

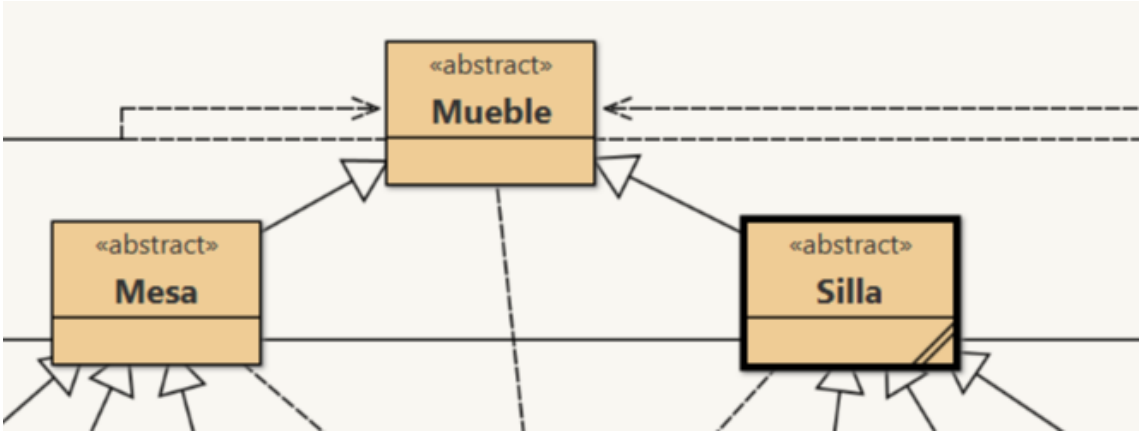
Clases

1.1 - Clases usadas / Relaciones

Para la realización de está practica se han utilizado un total de 28 clases, en las que hay dos jerarquías principales Mueble y Personas.

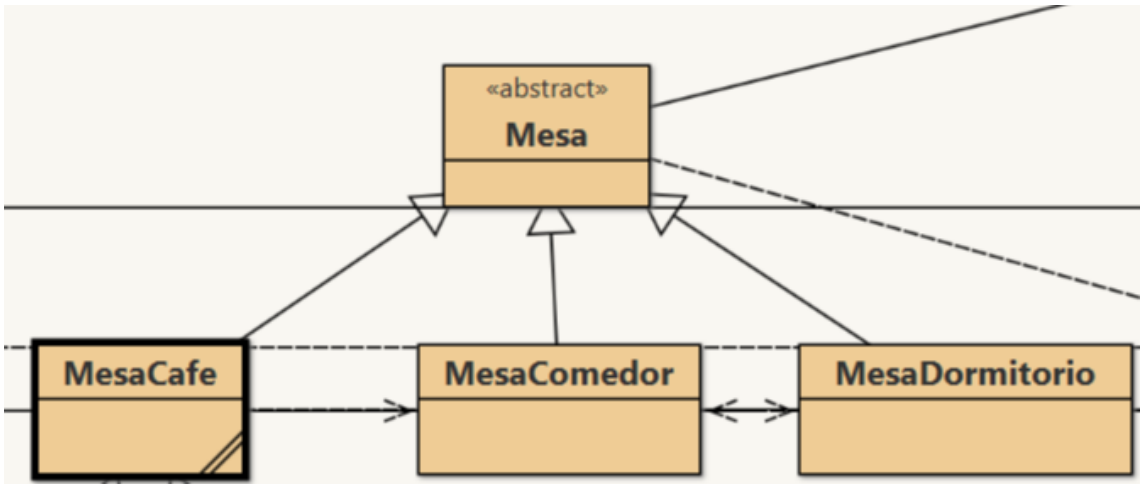
(Jerarquía Mueble)

La clase Mueble representa a la clase padre que abriga a todos los tipos de muebles que se fabrican en el almacén. Su distribución y relación sería la siguiente:

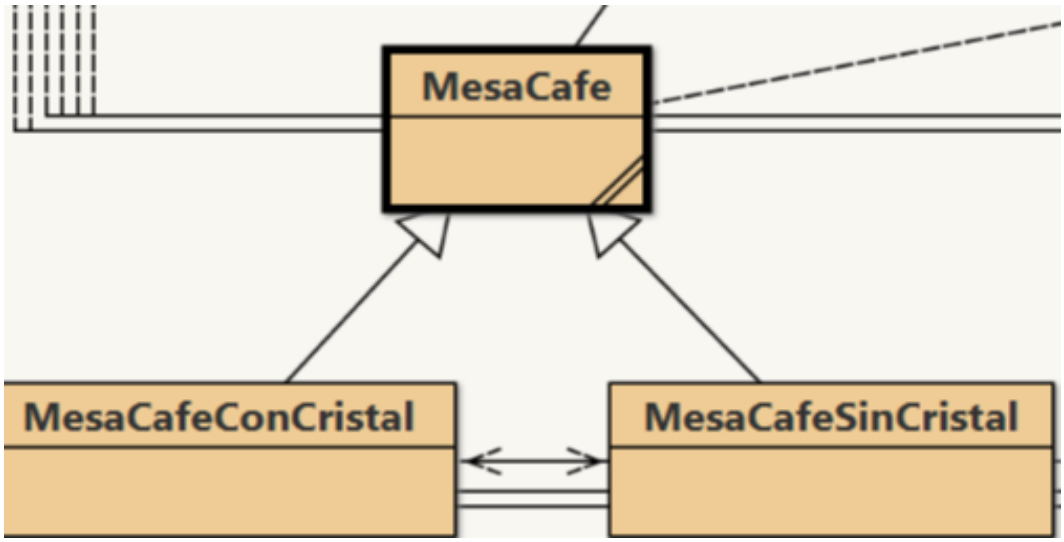


Como se puede observar de la clase principal Mueble, se extienden dos clases más, mobiliario de tipo Mesa y otro de Tipo Silla. Las tres son clases abstractas.

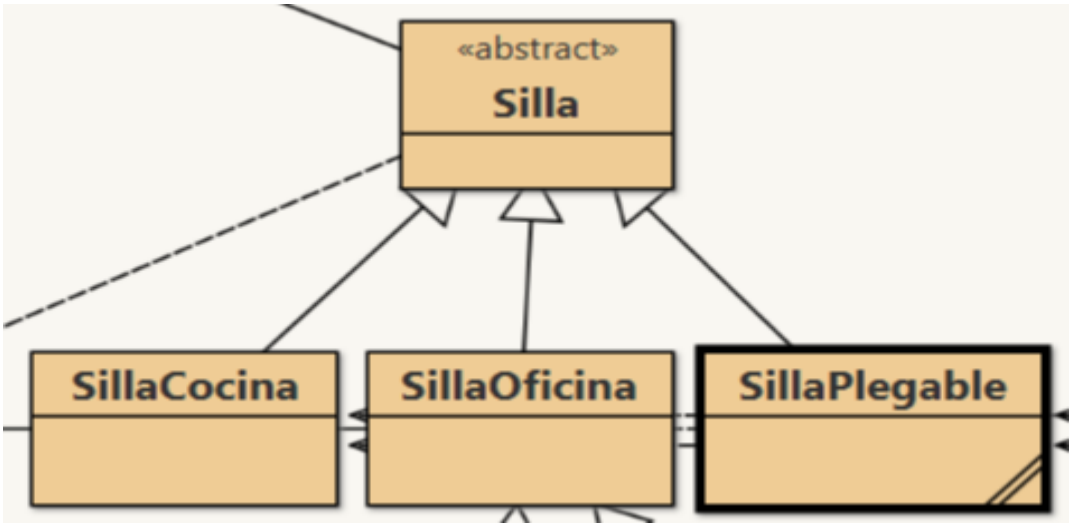
De la clase Mesa, se extienden otras tres clases; Mesa Café, Mesa Dormitorio y Mesa Comedor.



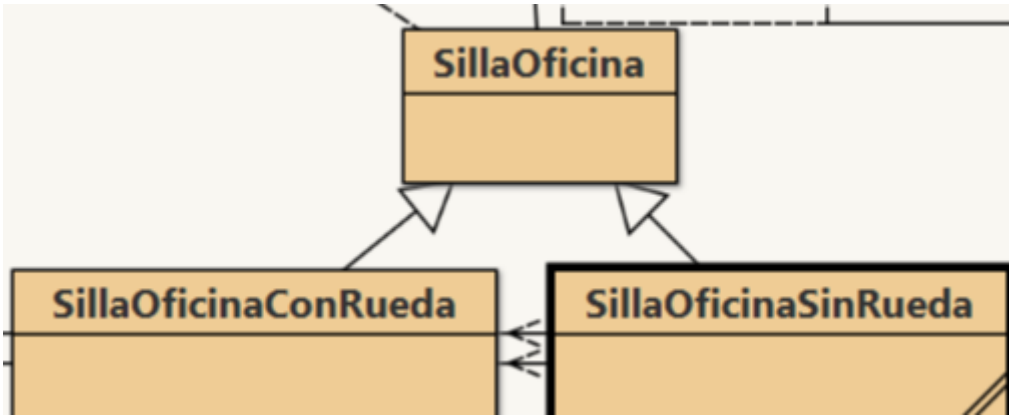
A su vez la clase Mesa Café, también actúa como clase padre de dos clases más Mesa Café con cristal y Mesa Café sin cristal.



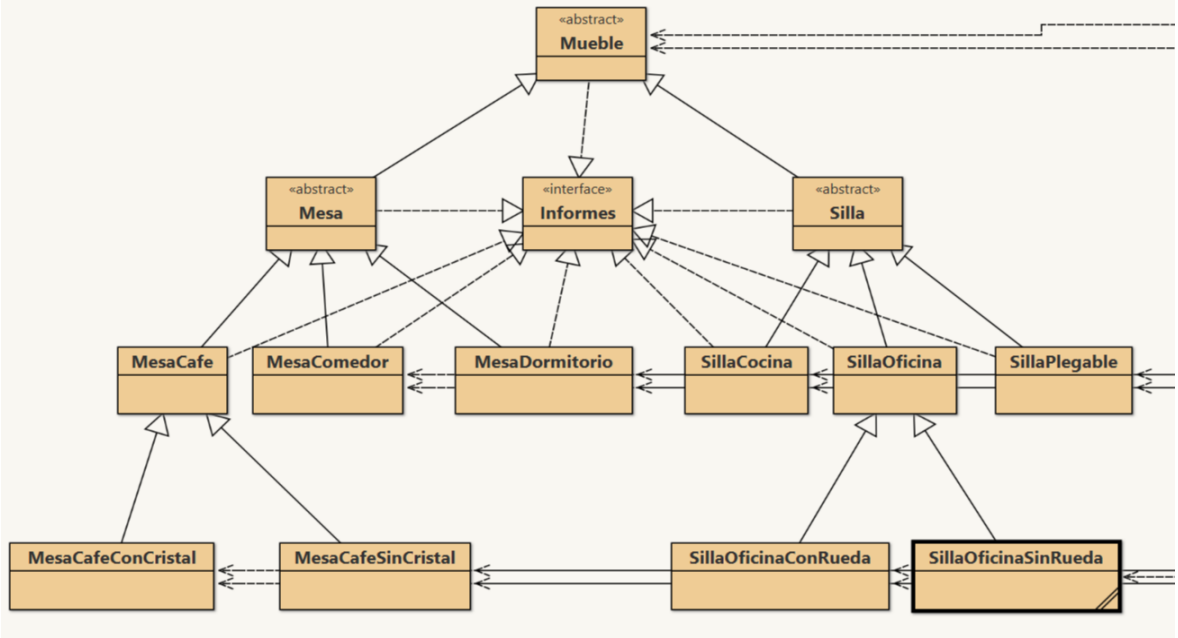
Si observamos la clase silla, podemos observar como de ella extienden tres clases más; Silla Cocina, Silla Oficina y Silla Plegable.



Como ocurriera con la clase Mesa Café, de la Silla de Oficina también extienden dos clases más; Silla Oficina con ruedas y Silla Oficina sin ruedas.



La visión global de estas relaciones sería la siguiente:



Como se puede observar todas estas clases implementan una interface que se llama Informes.

En el interface Informes se encuentran dos métodos:

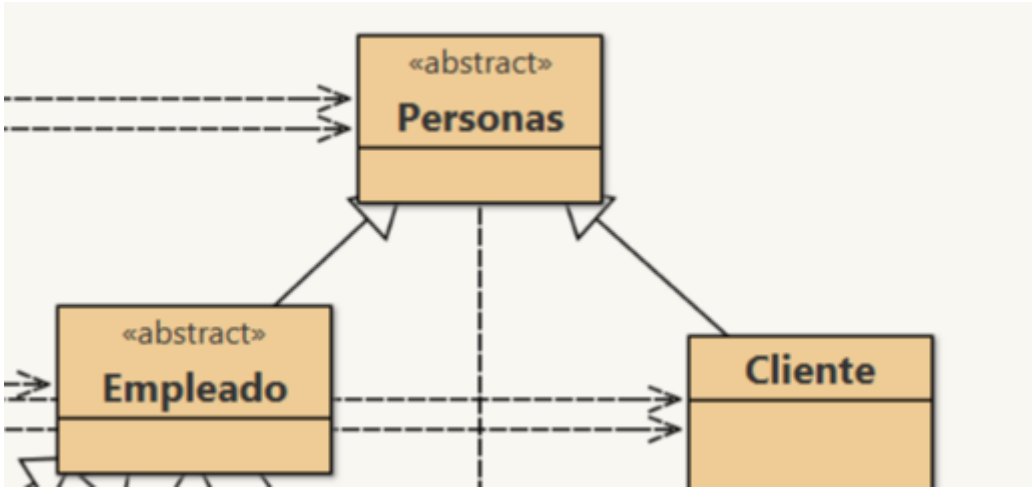
- public void SetDatos();
- public void SetActualizarDatos();

El primer método es llamado cuando creamos las clases desde el menú, y solicita el valor de cada atributo pantalla.

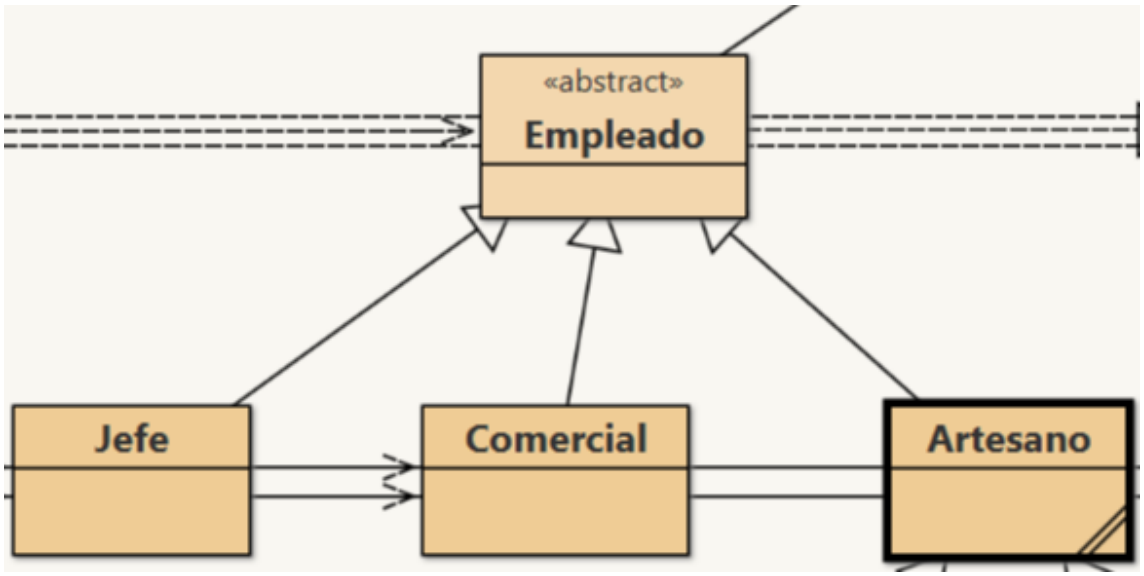
El segundo método es llamado cuando queremos actualizar algún atributo de las clases muebles que se han creado para ser fabricadas.

(Jerarquía Personas)

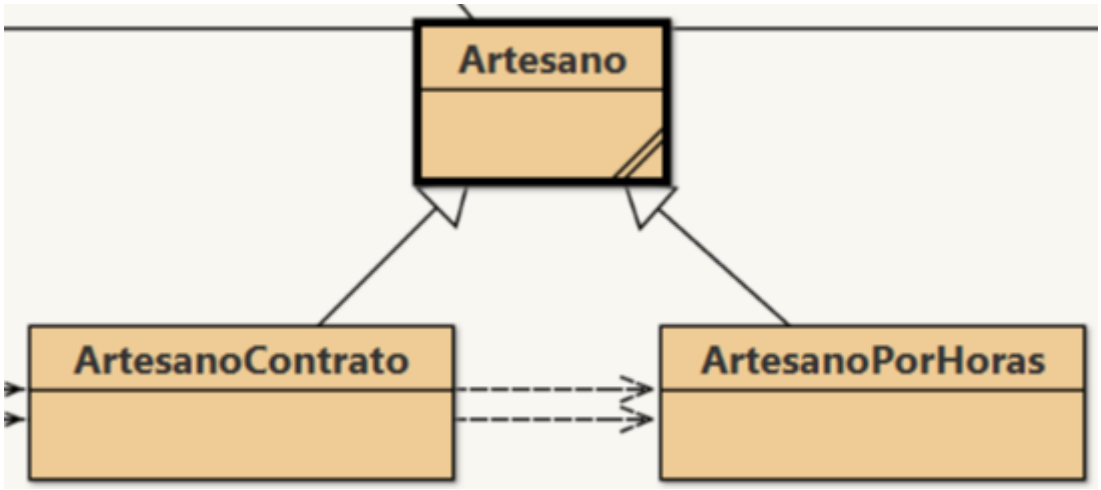
La clase Personas es una clase abstracta que se usa como base para la creación de las subclases relacionadas con las personas que forman la plantilla de la fábrica (clase Empleado) y también son la base de las clases que dan forma a los clientes (clase Cliente).



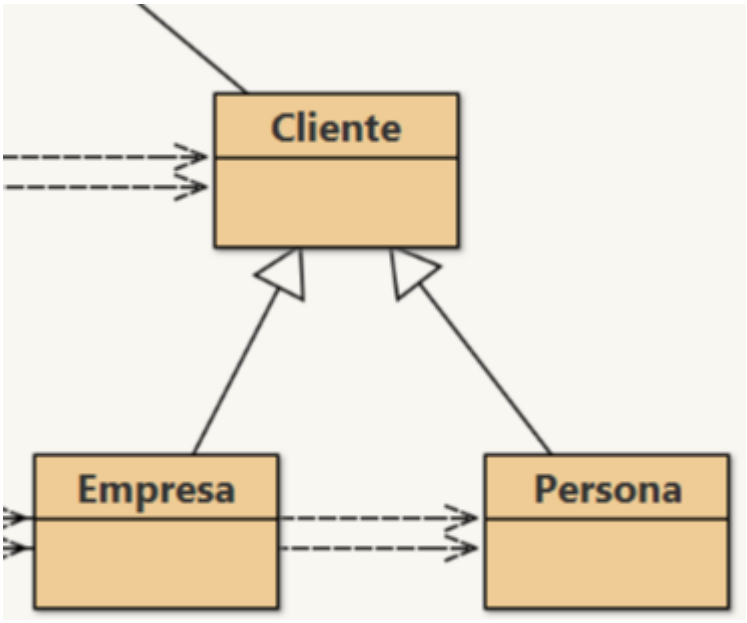
Como he comentado, la clase Empleado va a relacionar a las personas que trabajan dentro de la fábrica y de la que van a extender tres clases más del tipo empleados; Jefe, Comercial y Artesano.



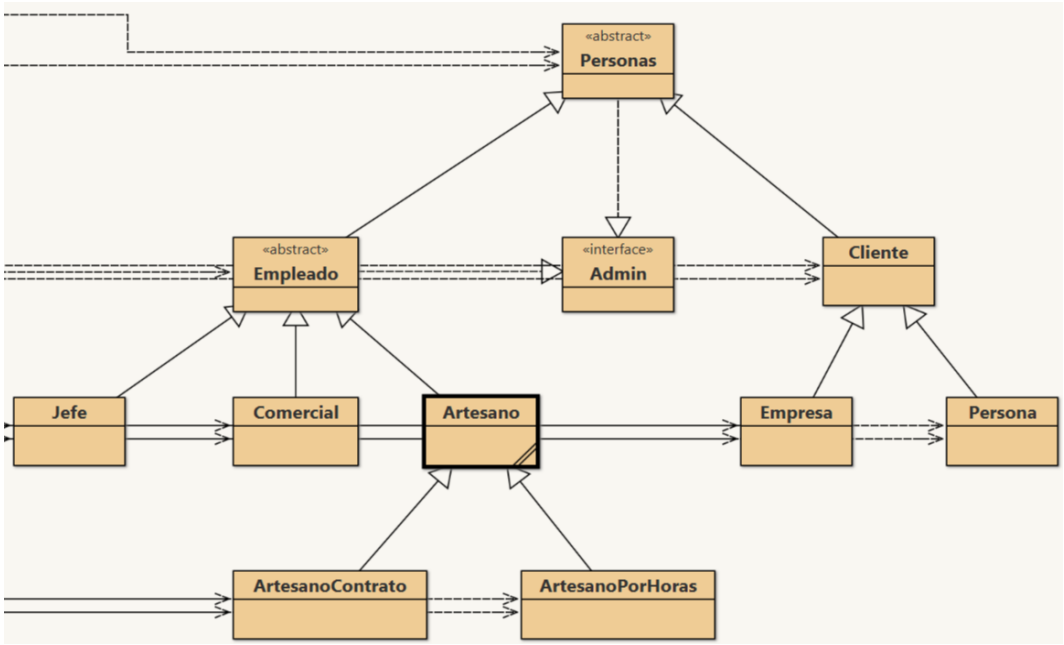
De estas tres clases, la clase Artesano también tiene su relación de clase padre con otras dos clases más; Artesano por contrato y Artesano por horas.



Por último tenemos la relación de la clase clientes. En este caso da estructura al tipo de personas que van a realizar pedidos en nuestro almacén. De esta clase extienden dos tipos de clientes; cliente tipo empresa (clase Empresa) y de tipo persona física (clase Persona).



La visión global de la relación de la clase Personas, sería la siguiente:



Para la jerarquía de Personas, también se ha implementado una interface llamado Admin.

En el interface Admin se encuentran dos métodos:

- public void SetDarDeAlta();
- public void SetActualizadorDatos();

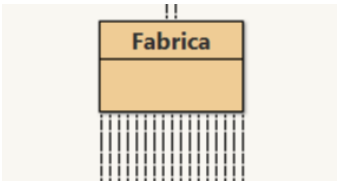
El primer método es llamado cuando creamos las clases desde el menú principal al crear indistintamente una clase tipo personal o cliente.

El segundo método es llamado cuando queremos actualizar algún atributo de las clases personas que se ya están creadas.

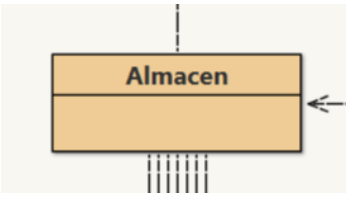
(Resto de clases)

Para gestionar el proceso de gestión y fabricación del almacén he procedido a la creación de 5 clases más.

La clase principal que da nombre al proyecto clase Fábrica:

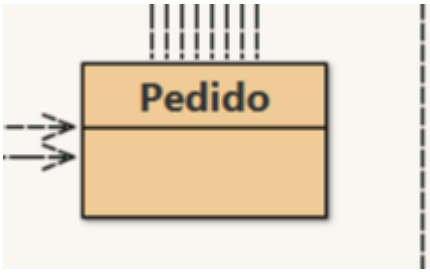


La clase Almacén, gestiona el CRUD del almacén (Creación, actualización, búsqueda y eliminación)

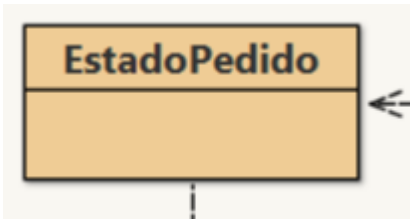


La clase Administración, gestiona el CRUD de las personas (Creación, actualización, búsqueda y eliminación)

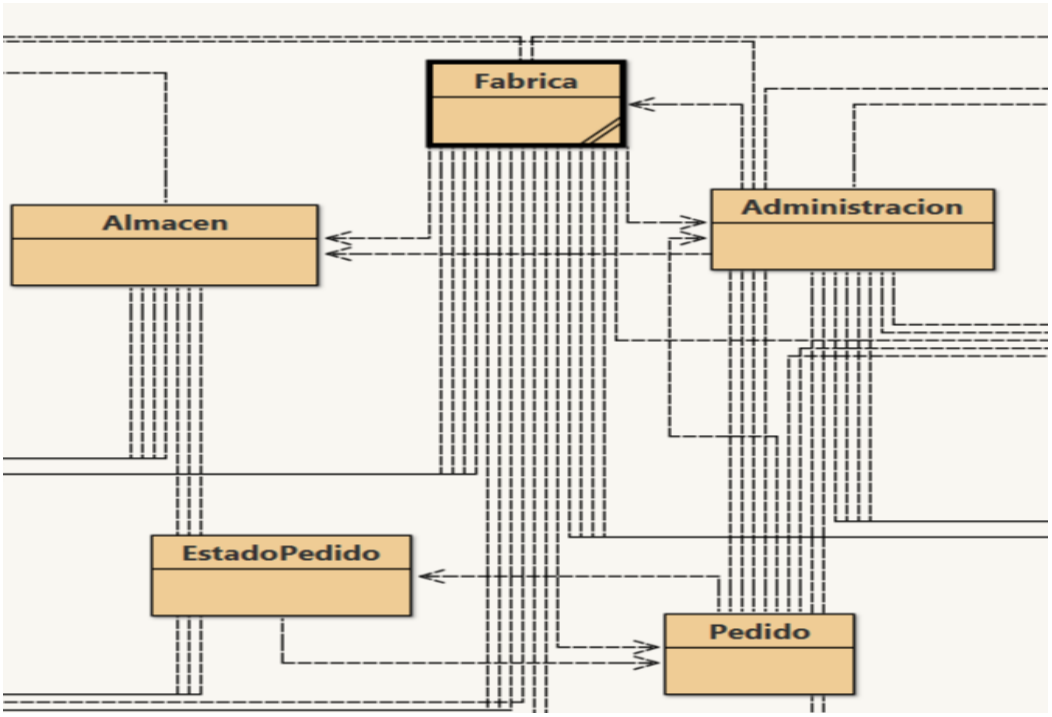
La clase Pedido, en la que se realiza todo el proceso de relación entre muebles y clientes. Además también tiene los métodos de asignación y gestión del proceso de fabricación de los pedidos.



Por último la clase estado pedidos, creada para para poder utilizarla como lista o tabla de los pedidos generados, donde se relaciona el mueble, el nº de pedido, cliente, artesano que lo está fabricando y un estado que representa el estado de la fabricación.



Relación completa:



1.2 - Funcionalidades

1 – Jerarquías Mueble y Personas, las funcionalidades implementadas han sido las mismas en todos los casos:

- métodos **set** y **get** de cada atributo
- el método **toString** para mostrar la descripción e información de cada artículo.
- Llamada al **metodo public void SetDarDeAlta()** para la jerarquía Personas y **public void SetDatos()** para la jerarquía Mueble. Su funcionalidad indicada más arriba.
- Llamada al **public void SetActualizadorDatos**() para la jerarquía Personas y **public void SetActualizarDatos()** para la jerarquía Mueble. funcionalidad indicada más arriba.

2 – Clase Almacén. La clase almacén contiene los siguientes métodos:

- **public void insertarMueble**-> Este método crea los muebles desde el menú añadiendo los atributos por pantalla.
- **public void MostrarMueble** -> Lista todos los muebles creados en el almacén
- **public void ConsultarMueble** -> Consulta un mueble en específico, buscado por el atributo "nombre".
- **ActualizarAlmacen** -> Actualiza los atributos de cada mueble ya creado en el almacén. La búsqueda se hace por el nombre del mueble
- **public void EliminarMueble** -> Elimina cualquier mueble creado en el almacén. La búsqueda se hace por el nombre del mueble.
- **public static Mueble getDevolverMueble**->este método retorna el mueble que se va a insertar en el pedido.
- **public void ActualizarAlmacen,** actualiza los muebles que ya están creados en el almacén. La búsqueda se hace por el nombre del mueble

3 – Clase Administración. Para este caso he decidido crear dos ArrayList para gestionar por separado las clases de empleados y clientes. La clase administración contiene los siguientes métodos:

- **public void ListadoPersonal / public void ListadoCliente,** carga los primeros registros al iniciarse la aplicación, para poder trabajar con datos desde el primer momento.
- **public void AñadirPersonal ->** Llamando a este método se añade los distintos tipos de personal que existe. Primero solicita indicar que tipo de personal es el que deseas crear.
- **public void AñadirCliente ->** Llamando a este método se añade los distintos tipos de cliente que existe. Primero solicita indicar que tipo de cliente es el que deseas crear.
- **public void DarBajaPersonal ->**Elimina el registro que se indique por nombre en relación al personal. Búsqueda por nombre
- **public void DarBajaCliente ->** Elimina el registro que se indique por nombre en relación al cliente. Búsqueda por nombre
- **public void BuscarInfoPersonal ->**Muestra todo el personal que hay registrado
- **public void BuscarInfoCliente ->** Muestra todo los clientes que hay registrado
- **public void ConsultarPersona ->**Realiza una búsqueda por nombre, en este caso del personal. Búsqueda por nombre
- **public void ConsultarCliente ->** Realiza una búsqueda por nombre, en este caso del cliente. Búsqueda por nombre
- **public void ActualizarDatosCliente ->** Actualiza cada atributo de cada clase cliente. La búsqueda del artículo se hace por nombre.
- **public void ActualizarDatosPersonal->** Actualiza cada atributo de cada clase personal. La búsqueda del artículo se hace por nombre.
- **public Cliente getDevolverCliente ->** Este método devuelve el cliente que se añadirá a cada pedido. Para acceder a él, hay que buscarlo por el nombre con el que se creó.
- **public static ArrayList<Cliente> getClientes(),** retorna el arraylist para gestionar los pedidos
- **public static ArrayList<Empleado> getPersonal()** retorna los empleados para poder gestionar la asignación de pedidos

4 – Clase fabrica. Es la parte main de la aplicación. En esta clase se muestran los menús por donde iterar con la aplicación y además también gestiona la creación y actualización de pedidos.

Dentro del método main, además de estar incluido la llamada al menú principal, también se encuentra la creación de una clase de ejemplo relacionado con los muebles y personas.

Los métodos que incluye son los que detallo a continuación:

- **public static void mostrarMenu()->** gestiona el menú principal de la aplicación con todas sus opciones.
- **public static void MenuInformes(),** menú que gestiona la llamada a los métodos relacionados con los distintos informes que incluye la aplicación.
- **public void crearMueble()**, contiene un menú donde seleccionar el tipo de mueble que se desea crear y la llamada al método crear de cada clase.
- **public static void añadirMueble** -> dependiendo de la opción que se haya seleccionado en el método anterior, creara un mueble u otro.
- **public void EliminarMueble()** ->Realiza la llamada al método que elimina el registro del mueble indicado por su nombre.
- **public void BuscarMueble()** -> //Realiza la llamada al método que busca el registro del mueble indicado por su nombre.
- **public void ActualizarNombre()->**Realiza la llamada al método que actualiza los atributos, según el registro del mueble indicado por su nombre.
- **public void listarMuebles-**> Muestra el catalogo detallando de todos los muebles que se fabrican en el almacén
- **public void CrearPersonas** -> contiene un menú donde dependiendo de la opción que se elija, puede añadir un empleado o un cliente y la llamada al método que crea cada clase.
- **public void BajaPersonas**-> Realiza la llamada al método que elimina el registro del personal indicado por su nombre, ya sea personal o cliente.
- **public void BuscarPersonas**-> Realiza la llamada al método que busca un registro concreto de persona indicado por su nombre, ya sea personal o cliente.

- **public void ListarPersonal** -> lista todo el personal o cliente registrado indicando de que tipo es (Personal o cliente).
- **public void ActualizarPersonal**->Actualiza los atributos del personal o cliente existente. Para ello se debe Indicar de que tipo es y el nombre de la persona o empresa que se desea actualizar.

5 – Clase pedido.

- **public Cliente ObtnerCliente(),** método que obtiene el cliente que se va a integrar en el pedido. Solicita el nombre del cliente con el que se registró para buscarlo.
- **public Mueble ObtnerMueble(),** método que obtiene el mueble que se va a integrar en el pedido. Pide que indique el nombre del mueble con el que se creó.
- **public Empleado ObtnerEmpleado(),** obtiene el array de empleado
- **public void CrearPedido()** -> crea los pedidos. Primero pregunta si existe el cliente o el mueble que se desea añadir al pedido. En cada caso si no existe, le indica que vaya a la opción requerida para crearlo y en caso contrario pide el nombre (cliente o mueble), para buscarlo y agregarlo al pedido. En última instancia Informa al usuario del nº con el que ha sido creado el pedido.
- **public void MostrarPedido()** – Muestra todos los pedidos creados. Detallando por nº pedido, cliente y artículos que lo componen.
- **public void LlenarPedido,** este método alimenta un arraylist que es invocado en el momento de crear el pedido. Sirve para llevar un control de los pedidos que se van creando y en qué situación están.
- **public void AsignarPedido, Metodo** asignar los pedidos a los artesanos. Primero muestra si hay pedidos sin asignar, una vez mostrados, luego pedirá cual es el nº del pedido asignar y posteriormente pedirá el nombre del artesano al que quiere asignar el pedido. En el caso de que no existiera pedidos pendiente de asignar o deseara modificarlo, dará la posibilidad introduciendo otra vez el nº de pedido a modificar.
- **public void MetodoAsignarPedido,** método llamado en AsignarPedido, y donde se realiza el proceso de asignación.

- **public void EstadoAsignacionPedidos(),**Método para listar todos los pedidos e indica si están asignados a un artesano o no
- **public static String getCliente**, método invocado en la clase EstadoPedido, cuando se inicia el proceso de fabricación del producto. Lo busca en el HashMap del pedido. Obtiene solo el nombre del cliente. Se pasa por parámetro el nº de pedido.
- **public String[] getArticulos,** método invocado en la clase EstadoPedido, cuando se inicia el proceso de fabricación del producto. Obtiene cada uno de los muebles creados en cada pedido. Se pasa por parámetro el nº de pedido.
- **public static String getArtesano,** método invocado en la clase EstadoPedido, cuando se inicia el proceso de fabricación del producto. Obtiene el artesano asociado al pedido
- **public void EstadoFabricacion()** método para asignar los estados a cada artículo incluido en un pedido. Para comenzar el proceso indicar el nº de pedido y la aplicación le indicara que estado desea asignar a cada articulo
- **public void ComenzarPedidos,** método llamado en EstadoFabricacion(), para la asignación de los estados.
- **public void MostrarEstadoFabricacionPedidos,** muestra todos los muebles a los que se les ha asignado un estado de fabricación. Detallando, pedido, articulo, estado, motivo de parada si existiera, artesano asignado y nombre del cliente.

5 – Clase EstadoPedidos. En esta clase tengo los siguientes métodos:

- métodos **set** y **get** de cada atributo
- **public String toString(),** para visualizar los atributos de la clase cuando es invocada.
- **public void SetDatosEstadoPedido,** Crea el estado de cada artículo. Necesita recibir dos parámetros cuando es llamado. El nº de pedido y el nombre del mueble.
- **public String getEstados,** muestra las opciones de los distintos estados que se desea asignar.
- **public String getMotivos(),** muestra y da la opción de señalar, los dos motivos que hay en el caso de que se seleccione el estado de “Parado”.

Descripción practica

2.1 - Detalles de la práctica

Este proyecto se ha desarrollado, teniendo en cuenta en todo momento, las pautas en las que se basaba la práctica, llegando a todos los puntos solicitados y bajo el paradigma que engloba está asignatura de la programación orientado a objetos.

Durante el proceso, he implementado algunas de las partes vistas durante el curso, como es la **herencia**, sobrescritura entre clases, la implementación de **interfaces** y **colecciones** (**ArrayList**, **HashMap**, etc.). También se han añadido los comentarios necesarios en cada parte del código. Y he realizado test unitarios para poder probar la funcionalidad de la aplicación.

El modelo de negocio pensado para esta aplicación, es la de una fábrica artesanal que fabrica muebles a medida y a petición del cliente. Tal y como se solicitaba en la descripción de la prueba, desde la aplicación se puede realizar las siguientes funciones:

- Crea Mueble / Personas
- Actualizar o eliminar muebles / Personas
- Búsqueda por puebles / Personas
- Listar todos los muebles fabricados / Personas creadas
- Crear pedidos y relacionar el cliente o el articulo
- Asignar el pedido a un artesano
- Gestionar la fabricación de cada mueble mediante, mediante la asignación de estados.
- Posibilidad de visualizar por medio de informes los pedidos creados, asignados y el estado actual e histórico.

A continuación, voy a proceder a detallar el uso de determinadas funciones realizadas en la práctica.

En primer lugar, en el caso de la relación de pedidos-cliente, he decidido usar un HashMap anidado. De esta manera he podido implementar una buena relación, porque de esta forma podía asociar en primera instancia un pedido con un cliente, y luego con el segundo HashMap, podía asociar el cliente con los muebles pedidos. Para solventar el problema de capacidad de almacenamiento de los muebles, he creado un ArrayList de la clase mueble, para poder añadir tantos muebles como yo quisiera.

En segundo lugar, para la asignación de pedidos a los artesanos, también opte por usar un HashMap, por su facilidad y comodidad a la hora de crear las relaciones necesarias entre pedidos y artesanos.

Y finalmente, para tratar el proceso de la gestión de los productos en su fase de gestión productiva (gestión de los estados), opte por crear una nueva clase, que constara de un atributo por cada clase que intervenía en la relación pedido-personas-mueble. De esta manera quedaba todo trazado y visible. Por poner un ejemplo, en el método **public void MostrarEstadoFabricacionPedidos** de la clase pedido, se muestra toda la información del proceso de creación y gestión del pedido.

2.2 – A tener en cuenta

A nivel global he usado el atributo nombre de las distintas clases a modo de ‘id’, como referencia para su gestión durante su proceso de vida dentro de la aplicación. Por ese motivo, para poder realizar cualquiera de las funciones de eliminación, búsqueda o actualización, se solicitara el nombre de los muebles o las personas para poder realizar las gestiones que sean requeridas.

Por último, cuando la aplicación se inicia, crea una serie de clases con las que poder trabajar desde el comienzo. Además, desde los métodos Listar Muebles o Listar Personal se pueden obtener toda la información de cada clase ya creadas, y las que se vayan creando en cada momento, incluso comprobar la actualización que se vayan realizando sobre ellos. Obviamente, entre esos datos, también está el nombre que será útil y necesario para poder moverse por la aplicación.

No obstante, les indico los nombres y la clase a la que hace referencia, para facilitarles el acceso y que puedan tener los nombres más a mano.

A continuación, adjunto la tabla con las clases y sus nombres.

Clase	***Nombre***
MesaCafeConCristal	MesadeCafeCristal
MesaCafeSinCristal	MesadeCafeSinCristal1
MesaComedor	MesaComedor1
MesaDormitorio	MesaDormitorio1
SillaCocina	SillaCocina1
SillaPlegable	SillaPlegable1
SillaOficinaConRueda	SillaOficinaConRueda1
SillaOficinaSinRueda	SillaOficinaSinRueda1
Jefe	Manuel
Comercial	Jose Luis
ArtesanoContrato	Maria
ArtesanoPorHoras	Mario
Empresa	Tiendas Rodriguez
Persona	Sonia