Universidad Rafael Landívar
Facultad de Ingeniería
Ingeniería en Informática y Sistemas
Análisis y Diseño ll

Documentación de proyecto

Estudiantes:
Carlos Enrique Menchú Sapón 1538715
Ismar Alexander Morales Miranda 1637115

CASOS DE USO PROGRAMA “VFIX”

Como sabemos un caso de uso es la descripción de una acción o actividad. Un diagrama de caso de uso es una descripción de las actividades que deberá realizar alguien o algo para llevar a cabo algún proceso. Los personajes o entidades que participarán en un diagrama de caso de uso se denominan actores. En el contexto de ingeniería del software, un diagrama de caso de uso representa a un sistema o subsistema como un conjunto de interacciones que se desarrollarán entre casos de uso y entre estos y sus actores en respuesta a un evento que inicia un actor principal. Los diagramas de casos de uso sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas.

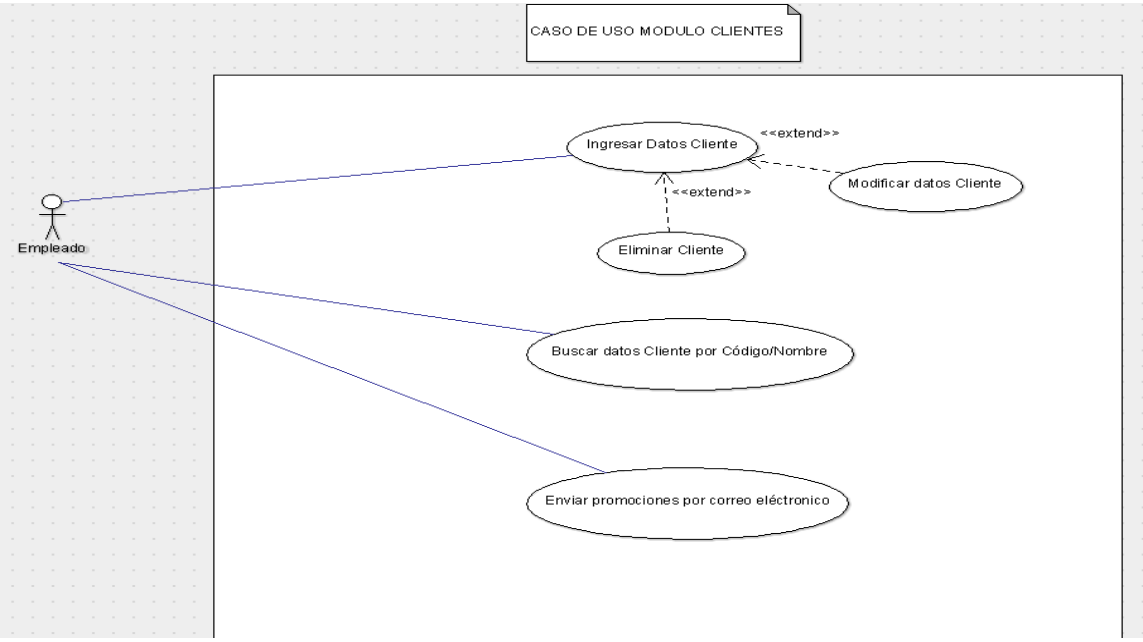
En la aplicación de escritorio que se desarrolló se usaron distintos casos de uso para tener una idea clara de las tareas que debe realizar cierto modulo y que no pasen por alto ya que tenemos estos para saber si se está cumpliendo con la tarea específica.

Caso de uso “Clientes”

Como caso de uso se tiene el modulo de clientes y el actor principal que es el empleado que manejará la aplicación de escritorio. Se tienen cosas específicas como lo es:

- Ingresar datos del cliente.
- Buscar datos del cliente por Código/Nombre.
- Enviar correo electrónico.

Se puede observar que el caso base es la inserción de los datos del cliente como su nombre, teléfono, correo electrónico, cumpleaños y estrellas. Como casos de uso extensibles se tienen el eliminar el cliente y modificar los datos. Ya que con el tiempo el cliente puede ser muy frecuente y la empresa le modificará las estrellas y así poder tenerle mas ofertas y promociones exclusivas que serán enviadas por correo electrónico.



Caso de uso “Inventario”

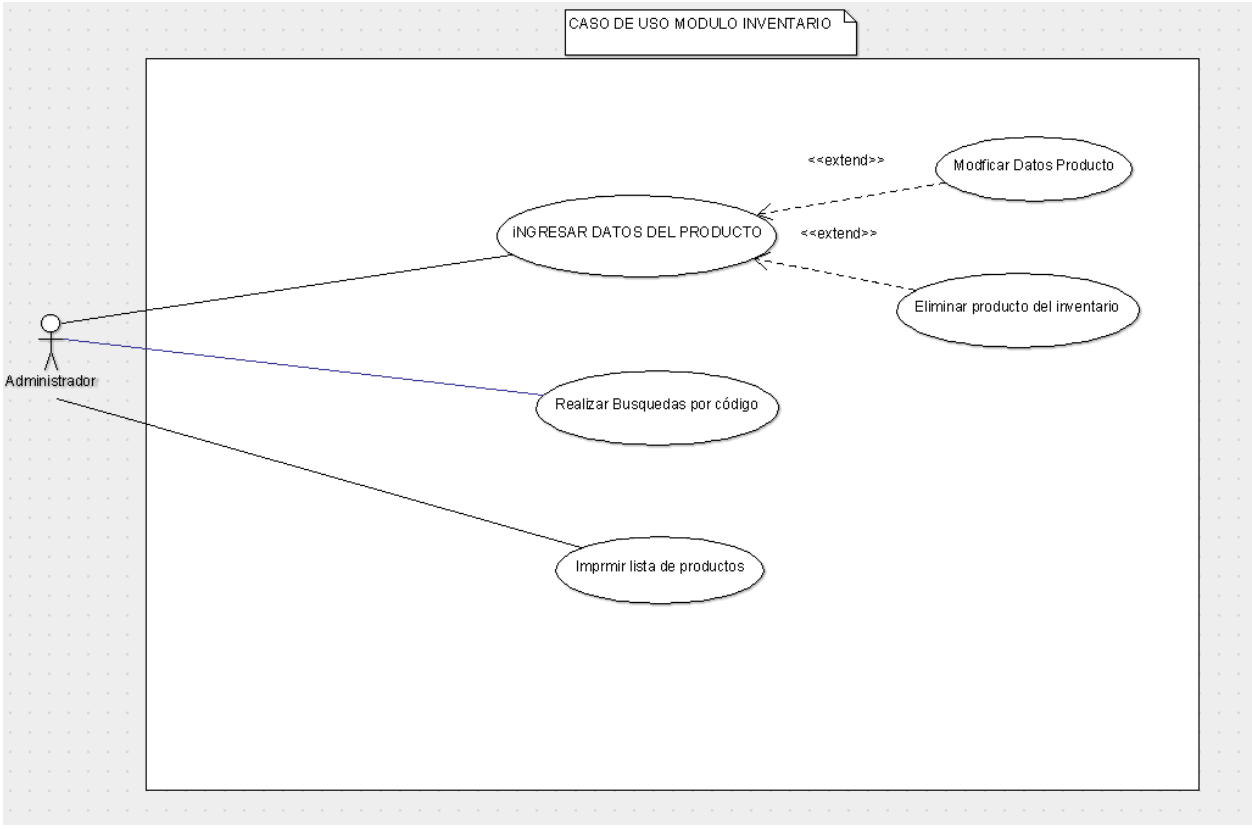
El presente caso de uso nos demuestra el funcionamiento del módulo inventario qué tiene como bases:

- Ingresar datos del producto.
- Realizar búsquedas por código.
- Imprimir lista de productos.

Se puede observar que el caso de uso controla el inventario y pueden hacerse búsquedas por código por el momento, pero en programación se dejó la aplicación extensible para nuevas modificaciones y otros tipos de búsquedas y así no tener que cambiar muchas clases para las cosas que se quieren lograr.

Se pueden modificar productos, existencias y las categorías a las que pertenece cada producto en general.

El actor que participa en el modulo inventario será el administrador del sistema, ya que él es el único que puede administrar el inventario y que los demás empleados no puedan modificar nada y así no existan anomalías en un determinado producto.



Caso de uso “Ventas”

Este es el modulo de ventas, se considera uno de los más importantes en la aplicación en conjunto con el modulo inventario, ya que estos tienen una serie de relaciones entre sí.

El actor de este modulo son los vendedores dentro de la empresa. Como casos tenemos:

- Ingresar Usuario y contraseña del vendedor.
- Ingresar datos del cliente.
- Realizar búsqueda de productos por código/nombre.
- Ingresar cantidad de producto a vender.
- Imprimir factura.

El modulo representa una serie de pasos ordenados que hay que seguir para realizar una venta, en el modulo el vendedor deberá ingresar su user y password para poder realizar una venta, ya que el sistema desea saber quien fue el vendedor que la realizó. De lo contrario no se podrá realizar ninguna acción.

Se utilizó como método de búsqueda de los productos el código ya que todos los productos manejan un código único para su venta.

Después de ingresar los productos podemos facturar la venta e imprimirla y tener una constancia de la venta para futuros reclamos de las mismas.

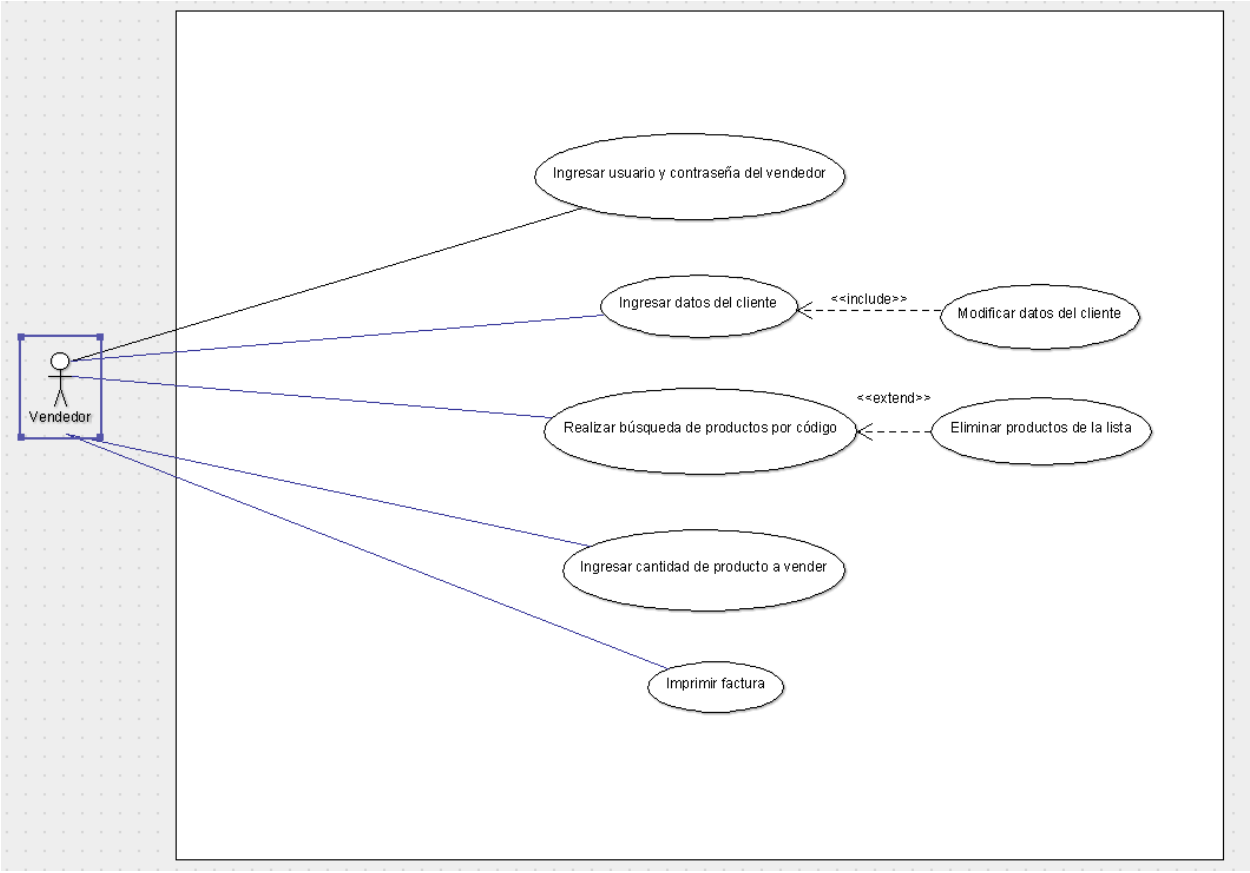


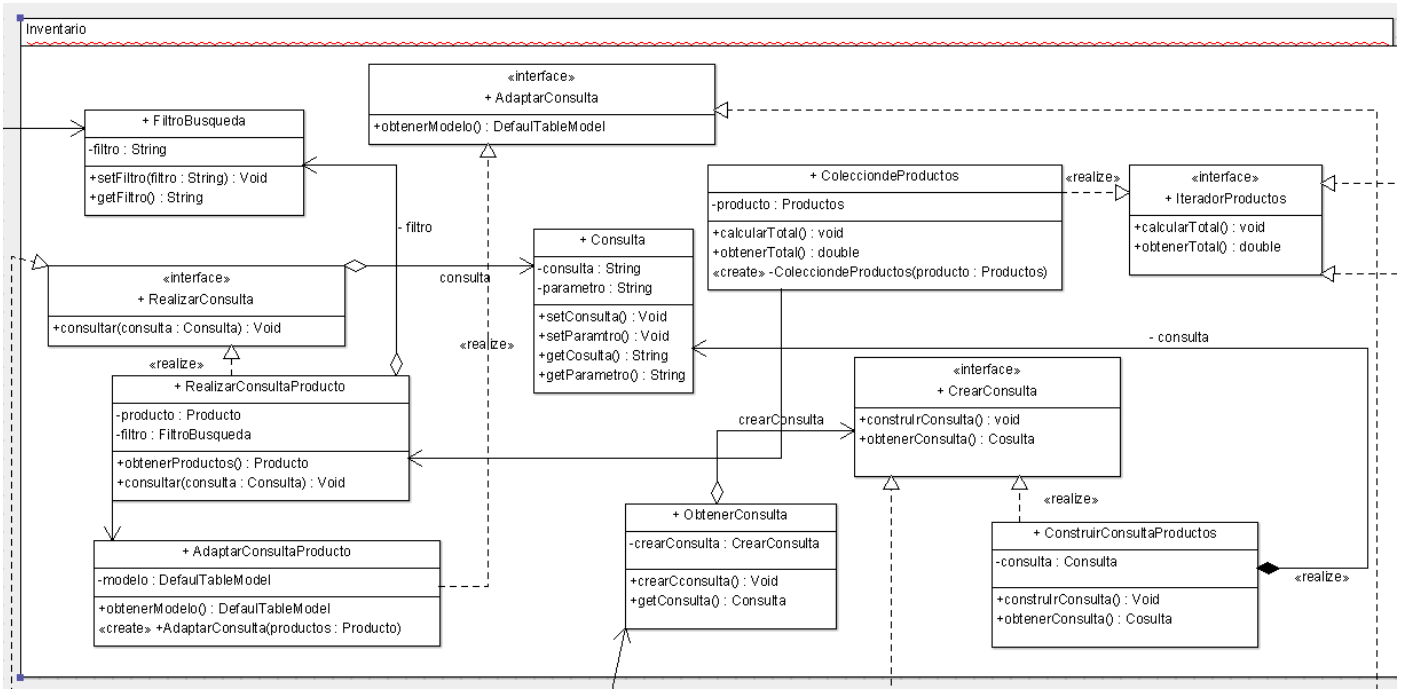
Diagrama de clases Empresa Vfix

El diseño consta de cuatro módulos que fueron diseñados los cuales fueron: Inventario, Ventas Clientes y Reparaciones. Estos módulos fueron diseñados cumpliendo con los principios de SOLID y con la implementación de patrones de diseño para lograr una funcionalidad y optima, así mismo también al cumplir estos principios se logra que le código sea más legible, eficiente y que sea posible extender la funcionalidad de la aplicación. A continuación se detallan los controladores de cada módulo diseñado explicando su funcionalidad y si se utilizó algún patrón de diseño se explicara que clases componen dicho patrón y porque se utilizó.

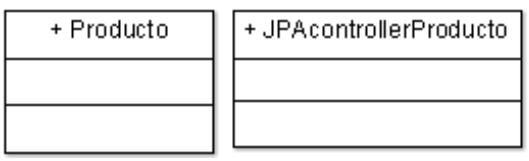
Modulo Inventario

La función del módulo inventario es que los usuarios puedan ingresar un nuevo producto, modificarlo si cometieron algún error en su ingreso y también que puedan realizar diferentes tipos de búsqueda de productos dependiendo porque atributo del producto lo quieran buscar mostrando el total en precio de los productos que sean devueltos al realizar la búsqueda.

- Diagrama de Clases

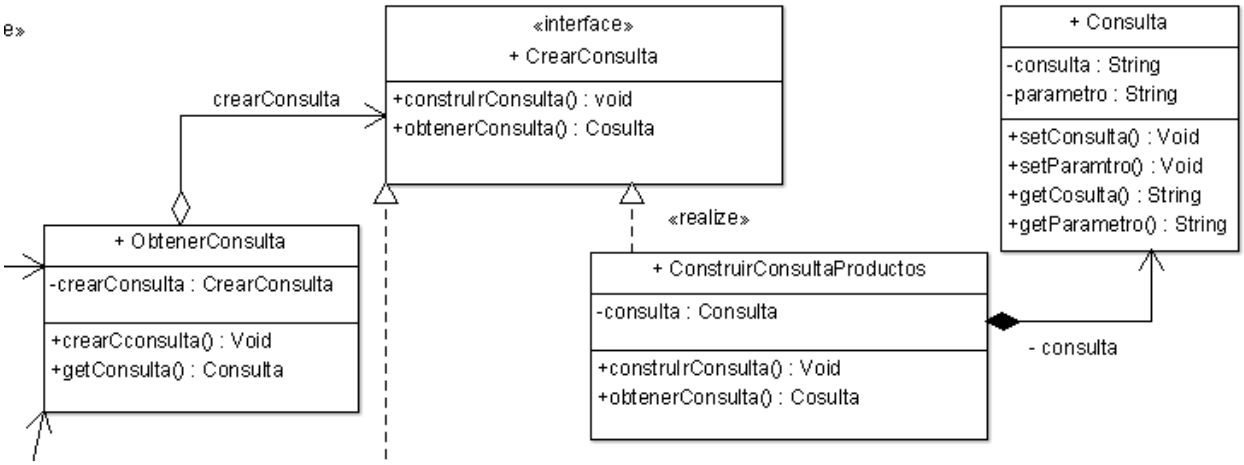


Para la inserción y modificación de un producto los encargados de guardar los datos en la base de datos son los controladores propios del modelo, en este caso son las clases Producto y ProductoJpaController. La clase Producto es la encargada de recibir los datos del producto que se va a ingresar a la BD y la clase ProductoJpaController recibe un objeto de tipo producto y es la encargada de insertar o modificar el objeto producto en la BD.



Seleccionar Tipo de Consulta para Búsqueda

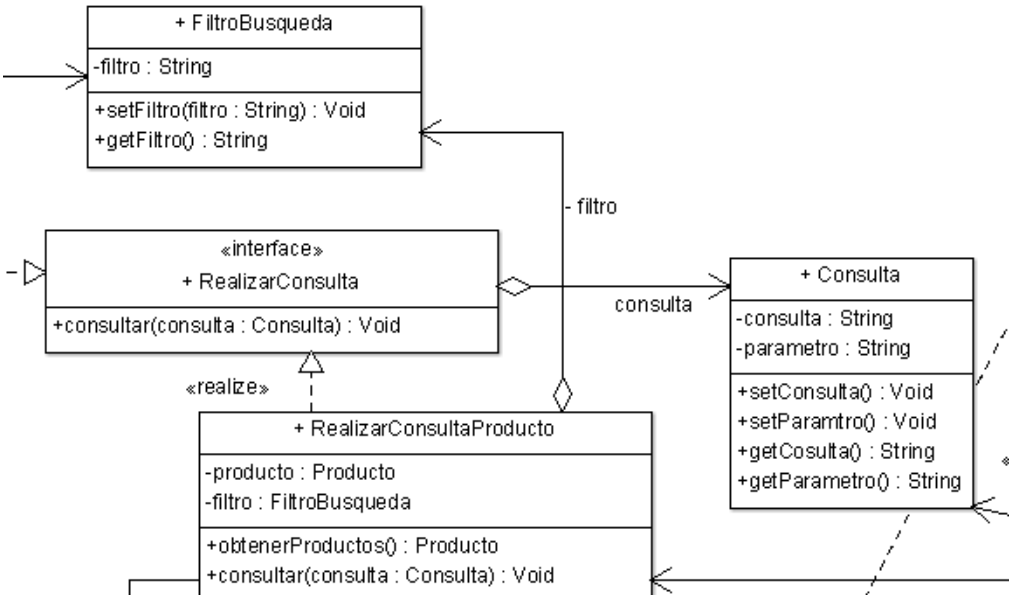
Para la realización de las búsquedas los usuarios pueden realizar las búsquedas por diferentes tipos de atributos que tenga el producto, esto presento un problema debido a que dependiendo por el tipo de búsqueda que ser quiera realizar del producto así se debe hacer la consulta a BD. Para resolver dicho problema de elegir la consulta necesaria para el tipo de búsqueda se utilizó el patrón de diseño ***builder*** para construir una consulta dependiendo del tipo de búsqueda a realizar. La conformación de este patrón está dado por: Interface CrearConsulta, clase ConstruirConsultaProducto, clase ObtenerConsulta.



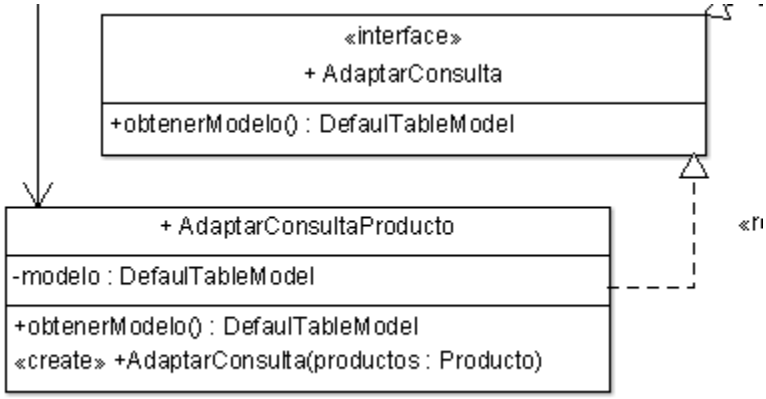
La clase **ObtenerConsulta** es la encargada de ordenar la creación de la consulta que se necesita para la búsqueda y también es la encargada de devolver un objeto de tipo Consulta con las consulta ya creada, la clase **ConstruirConsultaProducto** es la encargada de construir la consulta que se necesita dependiendo de porque atributo de vaya a realizar la búsqueda y la clase **Consulta** es la que será construida dependiendo de que consulta sea la necesaria de utilizar. La clase **ConstruirConsultaProducto** construirá a **Consulta** y le agregara la consulta y el parámetro necesario a utilizar.

Realizar Búsqueda

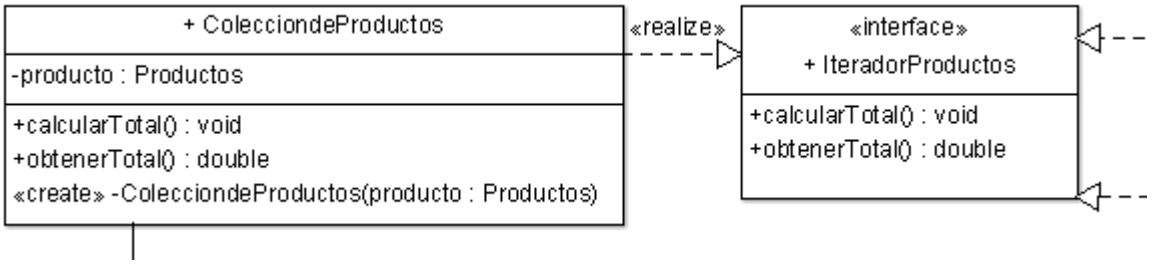
Se tiene una interfaz llamada **RealizarConsulta** cuyo método consultar recibe un parámetro de tipo **Consulta** que contiene la consulta y el campo por el que se va a realizar la búsqueda. Para la realización de la búsqueda se delega en una clase llamada **RealizarConsultaProducto** que implementa a **RealizarConsulta** el método que tiene es donde se realiza la consulta a la BD. **RealizarConsultaProducto** contiene una lista de tipo Producto que es la encargada de recibir los datos de la consulta y se tiene un método obtenerProducto que retorna una lista de productos con los productos que fueron encontrados al realizar la consulta, también contiene una variable de tipo FiltroBusqueda que es el encargado de recibir el parámetro porque el que se va a buscar (si la búsqueda es por el campo color, FiltroBusqueda recibe el color por el que se va a buscar).



Una vez realizada la búsqueda se implementó un ***proxy*** llamado **AdaptarConsultaProducto** que es el encargado de convertir la lista de Productos que retorna **RealizarConsultaProducto** a un **DefaulTableModel** para que los datos puedan ser mostrados al usuario en una tabla. **AdaptarConsultaProducto** implementa una interface llamada **AdaptarConsulta** que tiene un método obtenerModelo de tipo **DefaulTableModel** que es el encargado de convertir la lista de productos a un **DefaulTableModel** y retornarlo para que pueda ser mostrado.



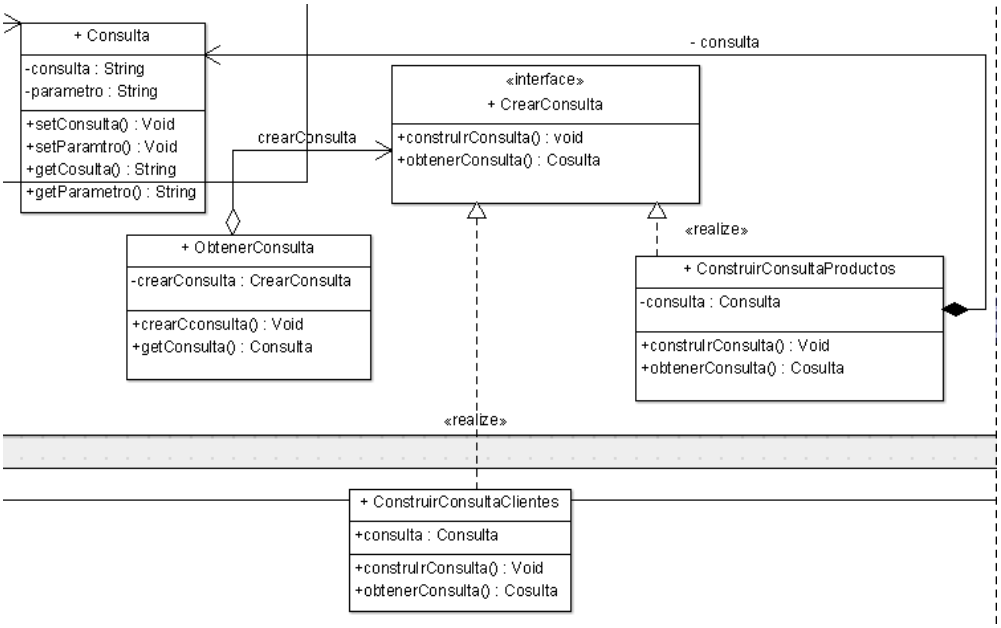
Debido a que los usuarios necesitan el valor total de los productos que han buscado se utilizó el patrón ***iterador*** para poder obtener el valor total de esos productos y mostrarlo a los usuarios. Se tiene una interface llamada **IteradorProductos** que contiene dos métodos llamados calcularTotal Y obtenerTotal que son los encargados de calcular el valor total de los productos y retornar el valor total de los mismos. La clase **ColeccciondeProductos** implementa dicha interface y también recibe una lista de tipo Producto sobre la cual va a actuar los métodos que implementa.



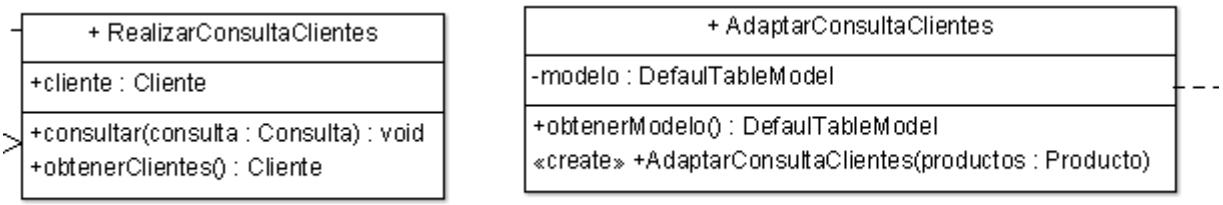
Módulo de Clientes

Para la inserción y modificación de un cliente los encargados de guardar los datos en la base de datos son los controladores propios del modelo, en este caso son las clases Cliente y ClienteJpaController. La clase Cliente es la encargada de recibir los datos del cliente que se va a ingresar a la BD y la clase ClienteJpaController recibe un objeto de tipo cliente y es la encargada de insertar o modificar el objeto cliente en la BD.}

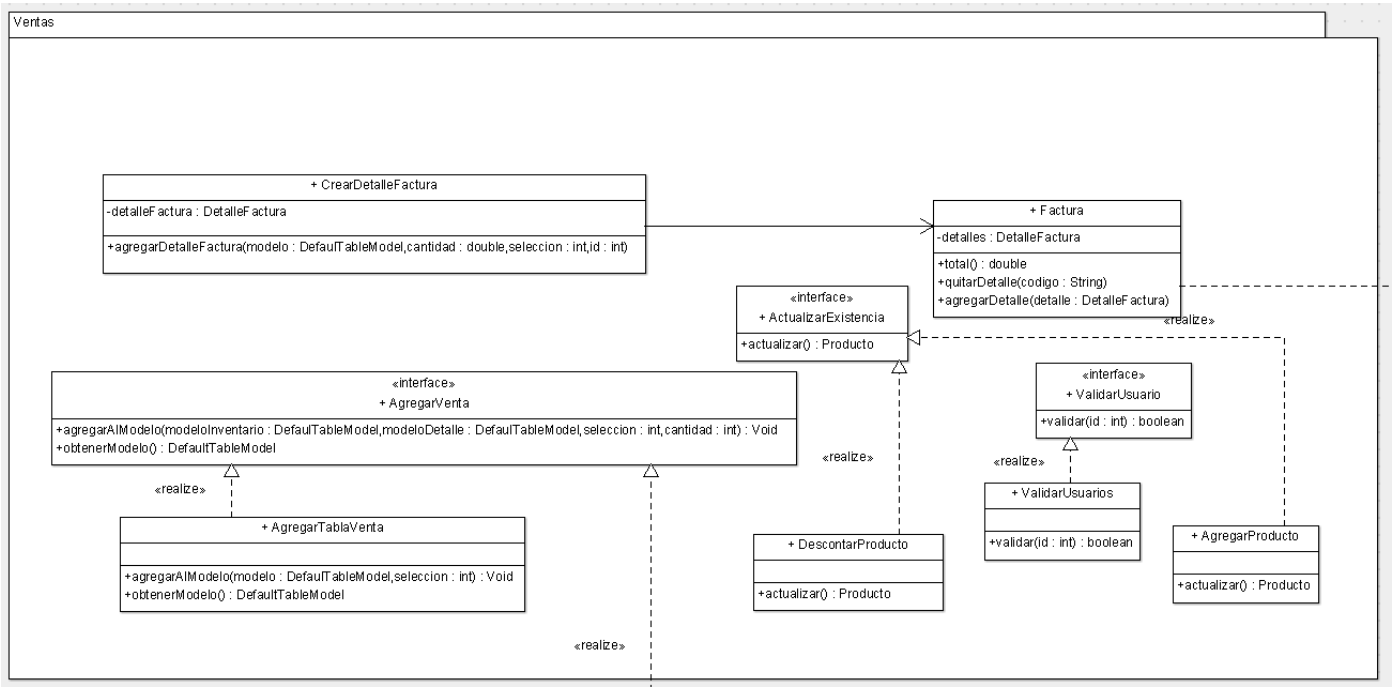
Para la búsqueda de un cliente de un cliente de igual manera que como se realiza con productos los usuarios pueden buscar a un cliente por distintos campos por ello se utilizó el mismo patrón ***builder*** ya implementado solo se le agrego otro constructor concreto que se llama **ConstruirConsultaClientes** su funcionamiento es el mismo que el de **ConstruirConsultaProducto,** construye la consulta del cliente dependiendo del campo por el que se quiere buscar al mismo.



Para realizar la consulta se le delego a una clase llamada **RealizarConsultaClientes** que es la encargada de realizar la consulta y devolver los datos encontrados. Y se utilizó un proxi llamado **AdaptarConsultaClientes** para convertir esos datos a un DefaultTableModel y poder mostrarlos en una tabla al usuario. **RealizarConsultaClientes** implementa la interfaz **RealizarConsulta** y **AdaptarConsultaClientes** implementa la interface **AdaptarConsulta.**



Modulo Ventas



La clase **AgregarTablaVenta** implementa la interface **AgregarVenta** que es la encargada de seleccionar un producto de la tabla inventario y pasarlo a la tabla de detalle factura.

Se necesita validar un usuario para realizar una venta por eso se tiene la interface **Validar Usuario** y la clase **ValidarUsuarios** que implementa la interface, si el usuario no es válido la venta no se realiza.

Para descontar un producto se dé la existencia se tiene la clase **DescontarProducto** que es la encargada de descontar de la existencia del mismo y para agregar a la existencia se tiene la clase **AgregarProcuto** ambas clases implementan la interface **ActualizarExistencia** cuyo método actualizar producto es el que utilizan las clases para realizar la actualización.

Para buscar los productos a vender se utilizan las mismas clases que se utilizan en el Modulo de ventas para realizar las Búsquedas.

Patrón Iterador Este patrón de diseño permite recorrer una estructura de datos sin que sea necesario conocer la estructura interna de la misma. Es especialmente útil cuando trabajamos con estructuras de datos complejas, ya que nos permite recorrer sus elementos mediante un Iterador, el Iterador es una interface que proporciona los métodos necesarios para recorrer los elementos de la estructura de datos, los métodos más comunes son:

- **hasNext:** Método que regresa un booleano para indicar si existen más elementos en la estructura por recorrer. True si existen más y false si hemos llegado al final y no hay más elementos por recorrer.
- **next:** Regresa el siguiente elemento de la estructura de datos.
- **Client:** Actor que utiliza al Iterator.
- **Aggregate:** Interface que define la estructura de las clases que pueden ser iteradas.
- **ConcreteAggregate:** Clase que contiene la estructura de datos que deseamos iterar.
- **IIterator:** Interface que define la estructura de los iteradores, la cual define los métodos necesarios para poder realizar la iteración sobre el ConcreteAggregator.
- **ConcreteIterator:** Implementación de un iterador concreto, el cual hereda de IIterator para implementar de forma concreta cómo iterar un ConcreteAggregate.

Este patrón se utilizó para **“manejar el detalle de la factura”** dentro del módulo ventas, la aplicación se desarrolló en “Java” y este lenguaje ya posee el patrón Iterator que no es más que una lista, solo se tiene que especificar que tipo de objetos son los que tratará, en nuestro caso serán de tipo “Producto” y métodos dentro del mismo para poder obtener el total de la factura que no es más que un ciclo que irá recorriendo cada objeto y obtener el método getPrecio(), y podremos quitar objetos si el cliente ya no desea un producto.

Se aprovecha el encapsulamiento al manejar cada objeto de distinta forma y así no tener equivocaciones al manejarlo directamente del JTable.

Con el uso de este patrón podemos tener métodos para recorrer la lista de la manera que se desee, en este caso se utilizó un recorrido de manera secuencial para ir obteniendo los precios de los productos que se iban seleccionando para la factura de la misma.