Práctica de Programación Orientada a Objetos – Junio 2015

Misrraim Suárez Pérez
e-Mail: misrraimsp@gmail.com
Tlf: 626743976

ÍNDICE

1. Actores principales, relaciones y jerarquías

La arquitectura general del sistema gira alrededor de cuatro clases básicas:

- **Productos**
- **Clientes**
- **Ventas**
- **Facturas**

Todas ellas se caracterizan por poseer identidad unívoca (un código de identificación) y una fecha de creación. Se modelan como clases que heredan de la superclase **Elemento**, siendo en esta donde se gestiona el código de identificación y la fecha de alta. En el siguiente diagrama de clases se especifican las variables de instancia de cada clase, así como las relaciones de herencia y uso:

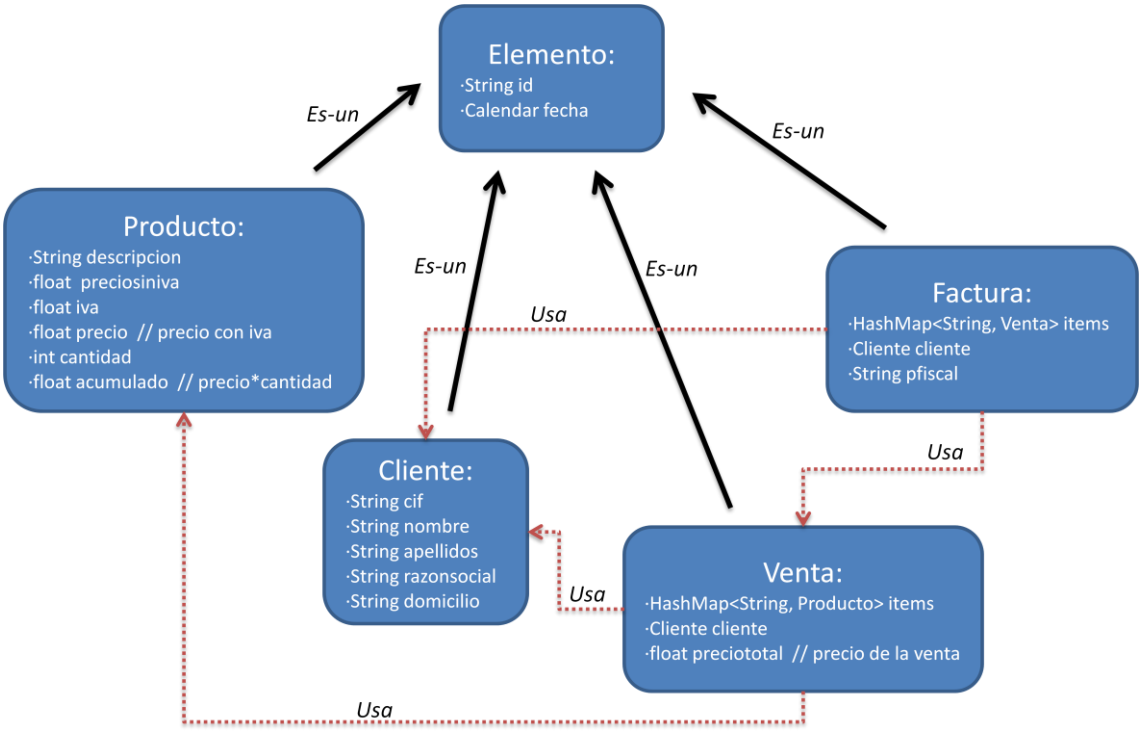

Figura 1. Jerarquía y usos a partir de la clase Elemento

Por otro lado, dichos elementos deben ser almacenados, de forma a poder acumularlos, aumentarlos, disminuirlos, exportarlos, y en general contenerlos para gestionarlos. Esta función se implementa con otras cuatro clases:

- **RegProducto**
- **RegCliente**
- **RegVenta**
- **RegFactura**

Éstas, a su vez, son extensiones de la superclase **Registro**, donde se implementa toda la funcionalidad común.

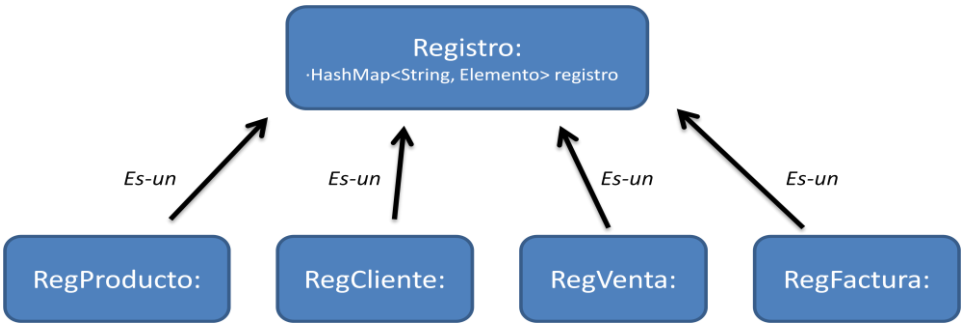

Figura 2. Jerarquía y usos a partir de la clase Registro

Además, el sistema cuenta con otras dos clases:

- **TPV.** Clase principal del sistema. Realiza la función de interfaz con el usuario (textual), canalizando todas las acciones hacia las correspondientes clases que las implementen.
- **IO.** Clase donde se ubica la funcionalidad de lectura y escritura de archivos.

Por lo tanto, toda la información del sistema se organiza y se sustenta su gestión en una arquitectura del tipo:

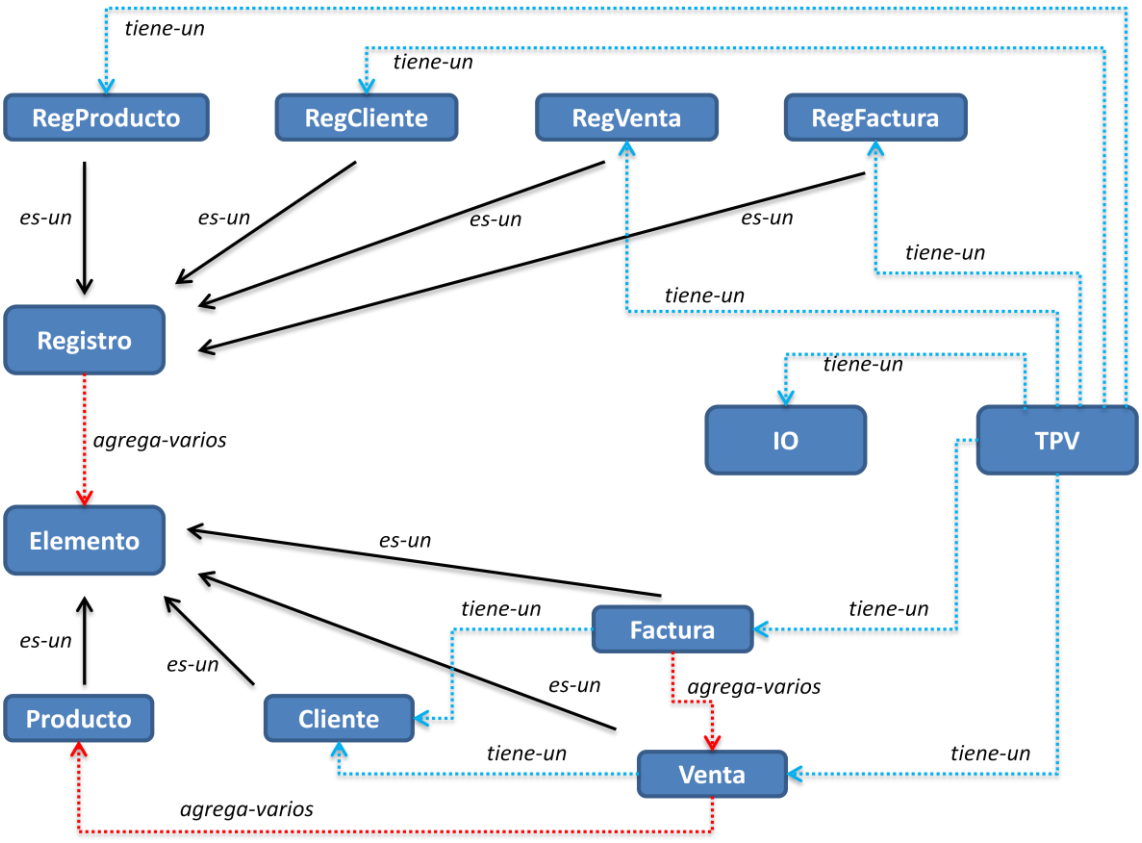

Figura 3. Esquema de la aplicación

2. Descripción de las clases

En este apartado se ofrece una descripción relativamente pormenorizada de los campos y métodos de cada una de las clases que integran el sistema. Se hace uso de la herramienta javaDoc para generar la documentación de los métodos.

2.1. Clase **TPV**

Es la clase principal de la aplicación.

- Campos

private RegProducto **almacen**;
Este campo tiene como misión dar soporte a los diferentes productos del establecimiento. Los almacena y ofrece métodos para su gestión.

private RegCliente **regcliente**;
Este campo tiene como misión dar soporte a los diferentes clientes. Los almacena y ofrece los métodos necesarios para su alta, baja, y demás gestiones necesarias.

private Venta **venta**;
Este campo representa una venta en curso. Cuando no hay venta abierta apuntará a null. Representa una suerte de "carrito de la compra", donde se van introduciendo todos los productos, secuencialmente, asociados a una venta en concreto.

private RegVenta **regventa**;
Este campo tiene como misión dar soporte a las diferentes ventas realizadas. Las almacena y ofrece los métodos necesarios para introducir nuevas ventas o hacer consultas sobre ventas realizadas.

private Factura **factura**;
Este campo representa una factura en curso. Cuando no hay factura abierta apuntará a null. Representa una especie de "buffer de ventas", donde se van introduciendo todas las ventas, secuencialmente, asociadas a una factura en concreto.

private RegFactura **regfactura**;
Este campo tiene como misión dar soporte a las diferentes facturas emitidas. Las almacena y ofrece métodos de acceso.

private IO **io**;
Proporciona los métodos necesarios para la lectura/escritura de archivos.

- Métodos públicos. Acciones disponibles al usuario

Los métodos públicos de esta clase son los que el usuario invoca, es decir, esta clase hace de interfaz (textual) con el usuario.

Constructor Summary

TPV()	Constructor de objetos de la clase TPV

Method Summary

void	**altaCliente**(java.lang.String cif, java.lang.String nombre, java.lang.String apellidos, java.lang.String razonsocial, java.lang.String domicilio) Da de alta un nuevo cliente
void	**altaProducto**(java.lang.String descripcion, float preciosiniva, float iva, int cantidad) Da de alta un nuevo producto
void	**aumentarFactura**(java.lang.String idventa) Adiciona una venta a la factura
void	**aumentarVenta**(int cantidad, java.lang.String descripcion) Adiciona un nuevo producto a la venta, buscándolo por descripción
void	**aumentarVenta**(java.lang.String idproducto, int cantidad) Adiciona un nuevo producto a la venta
void	**bajaCliente**(java.lang.String idcliente) Da de baja un cliente
void	**bajaProducto**(java.lang.String idproducto) Da de baja un producto
void	**cambiarApellidos**(java.lang.String idcliente, java.lang.String apellidos) Modifica los apellidos de un cliente.
void	**cambiarCantidad**(java.lang.String idproducto, int incremento) Modifica la cantidad de un producto
void	**cambiarCif**(java.lang.String idcliente, java.lang.String cif) Modifica el CIF de un cliente.
void	**cambiarDescripcion**(java.lang.String idproducto, java.lang.String nuevadescripcion) Metodo cambiarDescripcion(): Modifica la descripcion de un producto.
void	**cambiarDomicilio**(java.lang.String idcliente,

	java.lang.String domicilio) Modifica el domicilio de un cliente.
void	**cambiarIva**(java.lang.String idproducto, float nuevoiva) Modifica el IVA de un producto.
void	**cambiarNombre**(java.lang.String idcliente, java.lang.String nombre) Modifica el nombre de un cliente.
void	**cambiarPreciosiniva**(java.lang.String idproducto, float nuevoprecio) Modifica el precio sin IVA de un producto.
void	**cambiarRazonsocial**(java.lang.String idcliente, java.lang.String razonsocial) Modifica la razón social de un cliente.
void	**cancelarFactura**() Cancela la factura en curso
void	**cancelarVenta**() Cierra el proceso de venta.
void	**concluirFactura**() Cierra el proceso de facturar.
void	**concluirVenta**() Cierra el proceso de venta.
void	**disminuirFactura**(java.lang.String idventa) Elimina una venta de la factura
void	**disminuirVenta**(java.lang.String idproducto) Elimina el producto de la venta
void	**exportarClientes**(java.lang.String nombrearchivo) Escribe en un fichero los datos de los clientes
void	**exportarFacturas**(java.lang.String nombrearchivo) Escribe en un fichero los datos de las facturas
void	**exportarProductos**(java.lang.String nombrearchivo) Escribe en fichero todos los productos del almacén
void	**exportarVentas**(java.lang.String nombrearchivo)

	Escribe en un fichero los datos de las ventas
void	**importarClientes**(java.lang.String nombrearchivo) Da de alta en el sistema los clientes leidos de un archivo.
void	**importarFacturas**(java.lang.String nombrearchivo) Da de alta en el sistema las facturas leidas de un archivo.
void	**importarProductos**(java.lang.String nombrearchivo) Da de alta en el sistema los productos leidos de un archivo.
void	**importarVentas**(java.lang.String nombrearchivo) Da de alta en el sistema las ventas leidas de un archivo.
void	**imprimirFactura**(java.lang.String idfactura) Imprime una factura
void	**imprimirVenta**(java.lang.String idventa) Imprime el ticket de una venta
void	**nuevaFactura**(java.lang.String idcliente, java.lang.String pfiscal) Abre una nueva factura
void	**nuevaVenta**(java.lang.String idcliente) Abre una nueva venta

2.2. Clase **IO**

Es la encargada de gestionar la lectura y escritura en ficheros. Fundamental en las tareas de importación y exportación de objetos *elemento*.

- Variables de instancia

No tiene.

- Métodos públicos

Constructor Summary

IO() Constructor for objects of class IO

Method Summary	
void	**escribir**(java.lang.String nombrearchivo, java.lang.String texto) Escribe en el fichero especificado el texto dado.
java.lang.String	**leer**(java.lang.String nombrearchivo) Lee un fichero y devuelve el texto leído

2.3. Clase **Elemento**

Es la superclase que representa todas las entidades que se gestionan en el sistema.

- Variables de instancia

private String **id**;
Es el identificador unívoco que cada objeto *Elemento* posee.

private Calendar **fecha**;
Juega el papel de la fecha de alta en el sistema del objeto *Elemento*.

- Métodos públicos

Constructor Summary
Elemento() Constructor de la clase Elemento.
Elemento(java.util.Calendar fecha) Constructor de la clase Elemento.

Method Summary	
java.util.Calendar	**extraerFecha**(java.lang.String texto) Detecta, y devuelve en su caso, una fecha válida en el texto de entrada.
java.lang.String	**extraerPatron**(java.lang.String texto, java.lang.String marcainicial, java.lang.String marcafinal, java.lang.String regex) Detecta, y devuelve en su caso, un patrón en el texto de

	entrada.
java.lang.String	**fecha2String**() Devuelve la fecha en formato texto
java.util.Calendar	**getFecha**() Devuelve la fecha de alta del elemento
java.lang.String	**getId**() Devuelve el código del elemento
void	**setId**() Crea el ID del elemento a partir del campo fecha
void	**setId**(java.lang.String id) Establece como ID el pasado en parámetro.

2.4. Clase **Producto**

Esta clase genera los objetos producto que son vendidos en el establecimiento. Posee todos los campos exigidos así como métodos de acceso y modificación.

- Variables de instancia

private String **descripcion**;
Contiene la descripción del producto.
private float **preciosiniva**;
Precio sin iva del producto.

private float **iva**;
I.V.A. aplicable al producto

private float **precio**;
Es el precio de venta del producto, con I.V.A. incluido.

private int **cantidad**;
Número de unidades disponibles en stock del producto.

private float **acumulado**;
Es el valor total del producto teniendo en cuenta el número de unidades de que se dispone. Se obtiene como el producto del precio de venta por la cantidad disponible.

- Métodos públicos

Constructor Summary

Producto(java.util.Calendar fecha, java.lang.String descripcion, float preciosiniva, float iva, int cantidad) Constructor de objetos de la clase Producto, especificando la fecha de alta del producto
Producto(java.lang.String descripcion, float preciosiniva, float iva, int cantidad) Constructor de objetos de la clase Producto, sin especificar la fecha de creacion del producto

Method Summary

float	**getAcumulado**() Devuelve el valor acumulado de un producto
int	**getCantidad**() Devuelve la cantidad de un producto
java.lang.String	**getDescripcion**() Devuelve la descripcion de un producto
java.lang.String	**getFormatDescripcion**() Devuelve la descripcion de un producto fijando el tamaño a 14 caracteres con alineación a izquierda
float	**getIva**() Devuelve el IVA asociado a un producto
float	**getPrecio**() Devuelve el precio de un producto, con IVA
float	**getPreciosiniva**() Devuelve el precio de un producto sin IVA
void	**setCantidad**(int cantidad) Modifica la cantidad de un producto
void	**setDescripcion**(java.lang.String descripcion) Modifica la descripcion de un producto
void	**setIva**(float iva) Modifica el IVA aplicable a un producto
void	**setPreciosiniva**(float preciosiniva)

	Modifica el precio sin IVA de un producto
java.lang.String	**toString**() Redefinición de toString(): devuelve un String con toda la información ordenada del producto

Methods inherited from class Elemento
extraerFecha, extraerPatron, fecha2String, getFecha, getId, setId, setId

2.5. Clase **Cliente**

Esta clase genera los objetos que simulan los clientes que compran en el establecimiento. Posee todos los campos exigidos así como métodos de acceso y modificación.

- Variables de instancia

private String **cif**;
CIF del cliente.

private String **nombre**;
Nombre del cliente.

private String **apellidos**;
Apellidos del cliente.

private String **razonsocial**;
Razón social del cliente.

private String **domicilio**;
Domicilio del cliente.

- Métodos públicos

Constructor Summary
Cliente(java.util.Calendar fecha, java.lang.String cif, java.lang.String nombre, java.lang.String apellidos, java.lang.String razonsocial, java.lang.String domicilio) Constructor de objetos de la clase Cliente, especificando fecha de alta
Cliente(java.lang.String cif, java.lang.String nombre, java.lang.String apellidos, java.lang.String razonsocial, java.lang.String domicilio)

Constructor de objetos de la clase Cliente, sin especificar la fecha de alta

Method Summary

java.lang.String	**getApellidos**() Devuelve los apellidos del cliente
java.lang.String	**getCif**() Devuelve el CIF del cliente
java.lang.String	**getDomicilio**() Devuelve el domicilio del cliente
java.lang.String	**getNombre**() Devuelve el nombre del cliente
java.lang.String	**getRazonsocial**() Devuelve la razón social del cliente
void	**setApellidos**(java.lang.String apellidos) Modifica los apellidos del cliente
void	**setCif**(java.lang.String cif) Modifica el CIF del cliente
void	**setDomicilio**(java.lang.String domicilio) Modifica el domicilio del cliente
void	**setNombre**(java.lang.String nombre) Modifica el nombre del cliente
void	**setRazonsocial**(java.lang.String razonsocial) Modifica la razón social del cliente
java.lang.String	**toString**() Redefinición de toString(): devuelve un String con toda la información ordenada del cliente

Methods inherited from class Elemento
extraerFecha, extraerPatron, fecha2String, getFecha, getId, setId,

setId

2.6. Clase **Venta**

La clase Venta cumple el papel de representar una venta determinada de un conjunto de productos. Una instancia de esta clase posee toda la información relacionada con una venta, y los métodos necesarios para gestionarla.

- Variables de instancia

private HashMap<String, Producto> **items**;
Conjunto de productos que conforman la venta.

private Cliente **cliente**;
Cliente que realiza la compra

private float **preciototal**;
Precio total de la venta.

- Métodos públicos

Constructor Summary

Venta(java.util.Calendar fecha, Cliente cliente) Constructor de objetos Venta.
Venta(Cliente cliente) Constructor de objetos Venta.

Method Summary

Producto	**excluir**(java.lang.String id) Elimina el producto de la venta y lo devuelve como parametro
Cliente	**getCliente**() Devuelve el cliente de la venta
float	**getPreciototal**() Devuelve el precio total de la venta

java.util.HashMap<java.lang.String,Producto>	**getProductos**() Devuelve la coleccion completa de productos de la venta
void	**incluir**(Producto nuevoproducto) Introduce, si no estaba ya, un producto a la venta.
java.lang.String	**printTicket**() Devuelve un String con toda la información de la venta en formato de ticket
java.lang.String	**toString**() Redefinición de toString(): Devuelve un String con la información que define a la venta
void	**updatePrecio**() Establece el precio total de la venta

Methods inherited from class Elemento
extraerFecha, extraerPatron, fecha2String, getFecha, getId, setId, setId

2.7. Clase **Factura**

Esta clase representa las facturas generadas por el sistema. Cuenta con dos campos estáticos finales, es decir dos constantes de clase, que representan el CIF del vendedor y la razón social del vendedor.

- Variables de instancia

private Cliente **cliente**;
Cliente para el cual se pretende generar la factura.

private String **pfiscal**;
Periodo fiscal al que deben pertenecer las ventas de la factura.

private HashMap<String, Venta> **items**;
Conjunto de ventas incluidas en la factura.

- Métodos públicos

Constructor Summary

Factura(java.util.Calendar fecha, Cliente cliente, java.lang.String pfiscal) Constructor de objetos de la clase Factura.
Factura(Cliente cliente, java.lang.String pfiscal) Constructor de objetos de la clase Factura.

Method Summary

void	**excluir**(java.lang.String idventa) Elimina la venta de la factura
Cliente	**getCliente**() Devuelve el cliente de la factura
java.lang.String	**getPfiscal**() Devuelve el periodo fiscal de la factura
void	**incluir**(Venta venta) Introduce una venta a la factura
java.lang.String	**printFactura**() Devuelve un String con formato de factura
java.lang.String	**toString**() Redefinición de toString(): Devuelve un String con la información que define a la factura

Methods inherited from class Elemento
extraerFecha, extraerPatron, fecha2String, getFecha, getId, setId, setId

2.8. Clase **Registro**

Se trata de la superclase de todos los registros de elementos del sistema. Contiene los métodos de acceso y modificación necesarios para almacenar elementos.

- Variables de instancia

private HashMap<String, Elemento> **registro**;
Cumple la función de almacenar los diferentes elementos que el sistema gestiona.

- Métodos públicos

Constructor Summary

Registro() Constructor de objetos de la clase Registro

Method Summary

boolean	**containsKey**(java.lang.String llave) Devuelve true si el registro contiene la llave, false en caso contrario
Elemento	**getElemento**(java.lang.String llave) Devuelve el elemento correspondiente a la llave dada como parámetro
java.util.Set<java.lang.String>	**getLlaves**() Devuelve un conjunto con las llaves del registro
void	**putElemento**(Elemento elemento) Introduce un elemento al registro
Elemento	**removeElemento**(java.lang.String llave) Elimina del registro el elemento correspondiente a la llave dada como parámetro.
java.util.ArrayList<java.lang.Str	**splitElementos**(java.lang.String texto, java.lang.String regex)

ing>	Retorna un ArrayList resultado de cortar la cadena de entrada según la expresión de separación proporcionada
java.lang.String	**toString**() Redefinición de toString(): devuelve un String con toda la información ordenada del elemento correspondiente

2.9. Clase **RegProducto**

Esta clase es la encargada de ofrecer los métodos necesarios para realizar todas las operaciones que tengan que ver con la gestión de productos. Cumple el papel de almacén.

- Variables de instancia

No tiene.

- Métodos públicos

Constructor Summary

RegProducto() Constructor de objetos de la clase RegProducto.

Method Summary

void	**alta**(java.lang.String descripcion, float preciosiniva, float iva, int cantidad) Da de alta un nuevo producto.
java.lang.String	**altaAutomatica**(java.lang.String texto) Introduce en el sistema los productos válidos encontrados en el texto de entrada.
void	**baja**(java.lang.String idproducto) Da de baja un producto
void	**cambiarCantidad**(java.lang.String idproducto, int incremento) Modifica la cantidad de un producto.

void	**cambiarDescripcion**(java.lang.String idproducto, java.lang.String nuevadescripcion) Modifica la descripción de un producto.
void	**cambiarIva**(java.lang.String idproducto, float nuevoiva) Modifica el IVA de un producto.
void	**cambiarPreciosiniva**(java.lang.String idproducto, float nuevoprecio) Modifica el precio sin IVA de un producto.
void	**devolverProducto**(Producto producto) Introduce el producto devuelto en el registro
Producto	**solicitarProducto**(int cantidad, java.lang.String descripcion) Devuelve el objeto Producto buscado (por la descripción) en la cantidad pedida.
Producto	**solicitarProducto**(java.lang.String idproducto, int cantidad) Devuelve el objeto Producto buscado en la cantidad pedida.

Methods inherited from class Registro
containsKey, getElemento, getLlaves, putElemento, removeElemento, splitElementos, toString

2.10. Clase **RegCliente**

Proporciona los métodos necesarios para gestionar el almacenamiento de los clientes en el sistema.

- Variables de instancia

No tiene.

- Métodos públicos

Constructor Summary
RegCliente()

Constructor de objetos de la clase RegCliente

Method Summary

void	**alta**(java.lang.String cif, java.lang.String nombre, java.lang.String apellidos, java.lang.String razonsocial, java.lang.String domicilio) Da de alta un nuevo cliente
java.lang.String	**altaAutomatica**(java.lang.String texto) Introduce en el sistema los clientes válidos encontrados en el texto de entrada.
void	**baja**(java.lang.String idcliente) Da de baja un cliente
void	**cambiarApellidos**(java.lang.String idcliente, java.lang.String apellidos) Modifica los apellidos de un cliente.
void	**cambiarCif**(java.lang.String idcliente, java.lang.String cif) Modifica el CIF de un cliente.
void	**cambiarDomicilio**(java.lang.String idcliente, java.lang.String domicilio) Modifica el domicilio de un cliente.
void	**cambiarNombre**(java.lang.String idcliente, java.lang.String nombre) Modifica el nombre de un cliente.
void	**cambiarRazonsocial**(java.lang.String idcliente, java.lang.String razonsocial) Modifica la razón social de un cliente.

Methods inherited from class Registro

containsKey, getElemento, getLlaves, putElemento, removeElemento, splitElementos, toString

2.11. Clase **RegVenta**

Proporciona los métodos necesarios para gestionar el almacenamiento de las ventas en el sistema.

- Variables de instancia

No tiene.

- Métodos públicos

Constructor Summary
RegVenta() Constructor de objetos de la clase RegVenta

Method Summary	
java.lang.String	**altaAutomatica**(java.lang.String texto, Registro regcliente, Registro regproducto) Introduce en el sistema las ventas válidas encontradas en el texto de entrada.
void	**registrarVenta**(Venta venta) Introduce una nueva venta al registro

Methods inherited from class Registro
containsKey, getElemento, getLlaves, putElemento, removeElemento, splitElementos, toString

2.12. Clase **RegFactura**

Proporciona los métodos necesarios para gestionar el almacenamiento de las facturas en el sistema.

- Variables de instancia

No tiene.

- Métodos públicos

Constructor Summary

RegFactura() Constructor de objetos de la clase RegFactura

Method Summary

java.lang.String	**altaAutomatica**(java.lang.String texto, Registro regcliente, Registro regventa) Introduce en el sistema las facturas válidas encontradas en el texto de entrada.
void	**registrarFactura**(Factura factura) Introduce una nueva factura al registro

Methods inherited from class Registro
containsKey, getElemento, getLlaves, putElemento, removeElemento, splitElementos, toString

3. Procesos de venta y facturación. Solución implementada

Tanto las ventas como las facturas siguen el ciclo de vida:

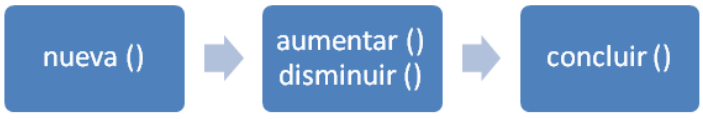

Figura 4. Creación de ventas y facturas

3.1. Proceso de venta

Una venta se crea, se añaden y/o se quitan productos, y por último se concluye. Veamos algo más detalladamente cada fase:

- Apertura de una nueva venta:
  El usuario invoca al método ***nuevaVenta****(String idcliente)*. El método busca en el registro de clientes el que corresponda al ID pasado (aborta el proceso y avisa si no encuentra nada) y llama al constructor de la clase Venta para

instanciarla. El constructor usado es el que establece la fecha actual como la fecha de la venta, ***Venta****(Cliente cliente)*. Se asigna la venta creada al campo *venta* de TPV.

- Añadir un producto a la venta:
  El usuario invoca al método ***aumentarVenta****(String idproducto, int cantidad)*. El método, después de comprobar que efectivamente existe una venta abierta, ejecuta la siguiente instrucción:

  venta.incluir(almacen.solicitarProducto(idproducto, cantidad));

  En este punto conviene explicar cómo se mantiene la coherencia entre la venta y el inventario. El método ***incluir****(Producto producto)* de la clase *Venta* añade un producto a la venta en curso. El producto que se añade es precisamente el devuelto por el método ***solicitarProducto****(String idproducto, int cantidad)* de la clase *RegProducto*. La ejecución de este último método simula la acción de físicamente ir al mostrador del ficticio almacén y pedirle al encargado que nos dé *cantidad* unidades del producto *idproducto*. Entonces, el método *solicitarProducto* devuelve el producto pedido en la cantidad estipulada, actualizando al mismo tiempo el contenido del inventario. En caso de algún problema (no existe el producto, cantidad disponible inferior a la pedida, etc.) informa de la situación y detiene el proceso.

  El proceso descrito también es posible realizarlo a través de una llamada por parte del usuario al método ***aumentarVenta****(int cantidad, String descripcion)*, que permite introducir un producto en la venta especificándolo por su descripción. La clave es el método *private String* ***descrip2id****(String descripcion)* de la clase *RegProducto*, que retorna el ID del primer producto encontrado en el inventario con la cadena *descripcion* contenida en su campo descripcion.

- Quitar un producto de la venta:
  El usuario invoca al método ***disminuirVenta****(String idproducto)*. El método, después de comprobar que efectivamente existe una venta abierta, ejecuta la siguiente instrucción:
  almacen.devolverProducto(venta.excluir(idproducto));

  Este es el proceso inverso al anterior. El método ***excluir****(String idproducto)* de la clase *Venta* elimina un producto de la venta en curso, devolviéndolo como parámetro. El producto eliminado es precisamente el suministrado al método ***devolverProducto****(Producto producto)* de la clase *RegProducto*, que lo vuelve a añadir al inventario.

- Concluir el proceso de venta
  El usuario invoca al método ***concluirVenta()*** que, después de comprobar que efectivamente existe una venta abierta, ejecuta:
  venta.updatePrecio(); // se establece el valor total de la venta
  regventa.registrarVenta(venta); // se almacena la venta en su registro
  System.out.println(venta.printTicket()); // se genera el ticket de la venta
  venta = null; // se cierra la venta

- Cancelar el proceso de venta
  Este proceso es importante porque cuando se cancela una venta se debe devolver todos los productos del carro de la venta al inventario. El usuario invoca al método ***cancelarVenta()***, que va devolviendo uno a uno todos los productos contenidos en la venta que se cancela de nuevo al almacén.

En síntesis, la solución adoptada para garantizar la integridad de los datos de los productos, en especial la cantidad de los mismos, durante el proceso de venta ha sido concebir que:

1. El campo *venta* de la clase **TPV** actúa a modo de "carrito de la compra". Así, cumple la misma función que el campo *almacén* en tanto en cuanto ambos campos almacenan en una colección HashMap<Idproducto, Producto> una serie de productos.
2. El acto de aumentar una venta es introducir en la colección de *venta* un nuevo producto, en cuyo campo *cantidad* se tiene la cantidad añadida a la venta de dicho producto.
3. Al tiempo que se añade el nuevo producto al carro de la compra se disminuye en *cantidad* unidades la cantidad del producto en cuestión en *almacén*.
4. El acto de disminuir una venta ó cancelarla sigue el mismo proceso descrito pero en sentido inverso: se elimina el producto de la colección de *venta* y se aumenta la cantidad en *almacén*.

3.2. Proceso de facturación

Como se ha indicado, el proceso de facturación sigue las mismas etapas que el proceso de venta. Una factura se crea, se añaden y/o se quitan ventas, y por último se concluye. El proceso es más simple que en el caso de las ventas porque no hay que actualizar cantidades de nada durante el proceso de facturar, es decir, cuando una venta se añade a la factura no tiene sentido hablar de actualizar el número de ventas en el registro de ventas. Sólo hay que tener en cuenta que la venta que se pretenda añadir a la factura pertenezca al cliente y año fiscal para el cual se está emitiendo la factura. Veamos cada fase:

- Apertura de una nueva factura:
  El usuario invoca al método ***nuevaFactura****(String idcliente, String pfiscal)*. Este método asigna al campo *factura* de TPV una nueva factura, con periodo fiscal *pfiscal* y cliente el correspondiente a *idcliente* (si no lo encuentra detiene el proceso e informa de la situación).

- Añadir una venta a la factura:
  El usuario invoca al método ***aumentarFactura****(String idventa)*. El método, después de comprobar que efectivamente existe una factura abierta y de encontrar la venta en el registro de ventas correspondiente a *idventa,* comprueba las condiciones de periodo fiscal y cliente. Si todo está en orden, añade la venta a la colección del campo *factura*.

- Quitar una venta de la factura:
  El usuario invoca al método ***disminuirFactura****(String idventa)*. El método, después de comprobar que efectivamente existe una factura abierta, elimina de la colección de la misma la venta especificada.

- Concluir el proceso de facturación
  El usuario invoca al método ***concluirFactura****()* que, después de comprobar que efectivamente existe una factura abierta, almacena la factura en el registro de facturas, la imprime por pantalla y establece el campo *factura* de TPV como null.

- Cancelar el proceso de facturación
  El usuario invoca al método ***cancelarFactura****()* que, después de comprobar que efectivamente existe una factura abierta, establece el campo *factura* de TPV como null.