Programación Orientada a Objetos

Práctica 3: Clases de asociación, y algoritmos de la STL
Implementación del caso de uso 3 «Facturación e histórico de compras» del S. G. L.

Curso 2016–17

Índice

1. Introducción

En esta práctica se va a llevar a cabo la implementación y prueba del caso de uso 3, «Facturación e histórico de compras», del Sistema de Gestión de Librería (SGL) descrito de manera general en la sección de prácticas del campus virtual de la asignatura. Además, se revisará la implementación de algunas operaciones de la práctica anterior para que se utilicen los algoritmos de la STL.

Se partirá de la solución de los casos de uso 1 y 2, desarrollada en la práctica anterior, junto con los requisitos correspondientes al caso de uso 3, y a partir del diagrama de clases se realizará la implementación en el lenguaje de programación C++. Después se deberá realizar una serie de pruebas para asegurarse de que la aplicación, para los mencionados casos de uso, funciona correctamente y satisface los requisitos especificados.

2. Objetivos

Implementar en C++ el modelo de clases para el caso de uso 3 del SGL, cumpliendo los principios de la programación orientada a objetos, y todas las pruebas necesarias para que estemos seguros en un alto grado de que la implementación tiene el comportamiento esperado.

Revisar la implementación de algunas operaciones de los casos de uso 1 y 2 del SGL para que utilicen algoritmos de la STL.

3. Descripción de requisitos del caso de uso 3

Continuaremos el desarrollo de nuestro SGL con la gestión de los pedidos que recibe de sus usuarios, partiendo de la gestión de usuarios, tarjetas y carritos de compra ya implementada. La gestión de pedidos incluye la facturación del contenido del carrito y la consulta del histórico de pedidos según distintos criterios.

- Facturación de la compra a partir del contenido del carrito.

  Cuando el usuario pasa a facturar la compra (*checkout*), el carrito se vacía y se genera un pedido de los artículos que contiene. El usuario deberá elegir la tarjeta de crédito con la que pagará el pedido, y de este se guardará además la fecha en la que se realiza. En este momento se asocia el pedido con la tarjeta de crédito y el usuario, y se actualizan las existencias disponibles de cada artículo. Esto quiere decir que se puede ir añadiendo al carrito lo que se quiera, pero hasta finalizar la compra no se comprueban y se ajustan las existencias, y en caso de que falte algún producto se vaciará el carrito y se anulará todo el pedido.

- Visualizar el histórico de pedidos.

  Se permitirán visualizar dos tipos de listados:

1. Pedidos realizados hasta la fecha incluyendo el detalle y el importe total de cada pedido, así como el importe de todas las ventas.
2. Pedidos realizados hasta la fecha agrupados por nombre de artículo e incluyendo para cada uno el número de pedidos, el importe total de las ventas y la suma de ejemplares vendidos.

Nótese que los precios de venta de los artículos pueden ser diferentes de sus precios actuales.

4. Diagrama de clases para el caso de uso 3

En la figura 1 aparece el diagrama de clases correspondiente al caso de uso 3.

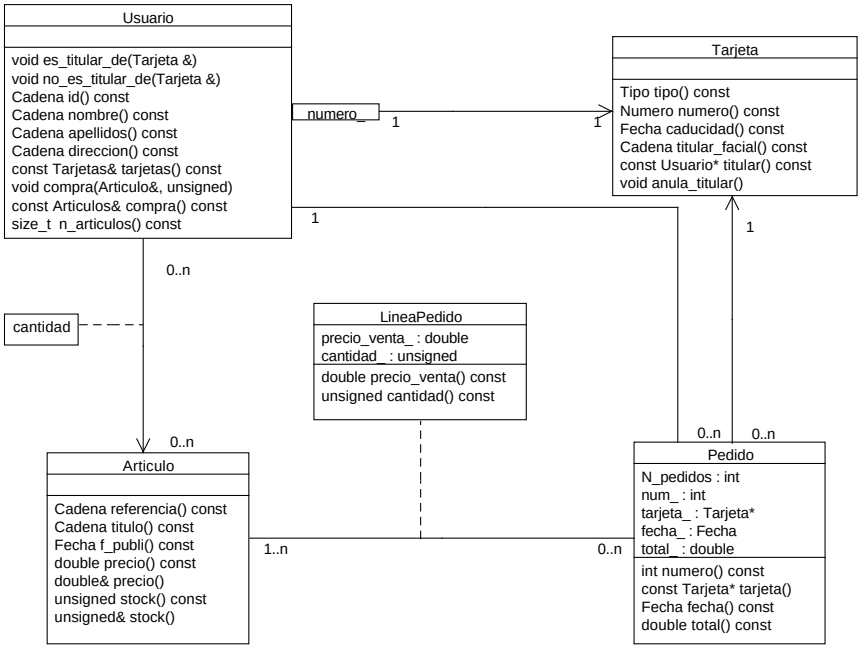

Figura 1: Diagrama de clases del caso de uso 3

En el diagrama aparecen las relaciones de asociación entre las clases ya implementadas, *Usuario*, *Tarjeta* y *Articulo*, y las clases nuevas, *Pedido* y *LineaPedido*. Se han omitido otras clases relacionadas con las anteriores, como *Clave* y *Numero*, por no estar directamente implicadas en el caso de uso que nos ocupa.

5. Implementación de las clases

5.1. La clase Numero

- Reimplemente la retirada de espaciado en blanco y la detección de caracteres no numéricos en el constructor de la clase *Numero* utilizando los algoritmos de la STL *remove_if* y *find_if*.
- Cada uno de los algoritmos anteriores recibe un predicado (clase de objetos función cuyo operador `()` devuelve un valor *booleano*). Necesitará dos, uno para cada algoritmo:
  - Para *remove_if* defina uno llamado *EsBlanco* para detectar espaciado en blanco, o bien use una función anónima (λ) de C++11.
  - Para *find_if* defina otro llamado *EsDigito* para detectar dígitos, y combínelo con el adaptador de negación unaria disponible en `<functional>` para detectar caracteres que no sean dígitos.

5.2. La clase Pedido

- Los cinco atributos de la clase *Pedido* son: el número del pedido (número correlativo empezando en 1), tarjeta de pago (puntero a *Tarjeta* constante), fecha del pedido, importe total de la venta y número del último pedido realizado, o cantidad de pedidos hechos.
- El constructor realiza la compra de los artículos, para lo cual utiliza cinco parámetros:
  - La asociación entre usuarios y pedidos, representada por la clase *Usuario_Pedido* (V. § 5.5),
  - la asociación entre pedidos y artículos, representada por la clase *Pedido_Articulo* (V. § 5.4),
  - el *Usuario* que realiza la compra,
  - la *Tarjeta* con la que se realiza el pago, y
  - la *Fecha* del pedido, por omisión será la actual.

  Cuando se genera un pedido de los artículos que contiene el carrito, este quedará vacío y las existencias de los artículos vendidos actualizadas. El nuevo pedido quedará asociado al usuario que lo realiza, a la tarjeta de pago y a los artículos incluidos en la compra. Se llamará a los métodos apropiados de la clases de asociación para crear los enlaces (véanse § 5.4 y § 5.5).
- Si el carrito no tiene ningún artículo, para evitar crear un pedido vacío, el constructor terminará con la excepción *Pedido::Vacio*. Esta clase tiene un atributo que es un puntero al usuario que realiza el pedido, un constructor que recibe el usuario y un método observador *usuario* que lo devuelve.

Se comprobará también que el titular de la tarjeta es quien hace el pedido. De no ser así se lanzará la excepción *Pedido::Impostor*. Al igual que en la clase de excepción anterior, el atributo almacena un puntero al usuario, el constructor recibe el usuario y el observador lo devuelve.

En el caso de que la cantidad de alguno de los artículos del carrito supere las existencias en almacén (*stock*), se vacía el carrito y se anula el pedido lanzando la excepción *Pedido::SinStock*. Esta clase de excepción consta de un atributo de tipo puntero a *Articulo*, un constructor que recibe el primer artículo del carrito sin existencias suficientes y un método observador *articulo* que lo devuelve.

Además, el constructor lanzará la excepción *Tarjeta::Caducada* si la fecha de caducidad de la tarjeta de pago es anterior a la fecha del pedido.

- Proporcionará métodos observadores para sus atributos, cuyos nombres serán *numero*, *tarjeta*, *fecha*, *total* y *n_total_pedidos*.
- Se sobrecargará el operador de inserción en flujo (<<) para mostrar o imprimir un *Pedido* en un flujo de salida con el formato del ejemplo:

```
Núm. pedido: 1
Fecha:       jueves 10 de marzo de 2016
Pagado con:  VISA n.º: 4539451203987356
Importe:     149,95 €
```

5.3. La clase LineaPedido

- Se utilizará para almacenar los atributos de enlace de la asociación bidireccional *varios-a-varios* entre *Pedido* y *Articulo*. Por tanto, constará de dos atributos: precio de venta y cantidad vendida. Este precio puede ser diferente del precio actual, ya que los precios pueden cambiar a lo largo del tiempo, de ahí la necesidad de almacenarlo por separado.
- Una *LineaPedido* se construye con dos parámetros que corresponden a los valores de sus atributos. La cantidad, que es el segundo, tomará por omisión el valor 1. No se permitirá la conversión implícita de un precio a *LineaPedido*.
- Tendrá dos métodos observadores llamados *precio_venta* y *cantidad*, que devolverán sus atributos.
- Se definirá el operador de inserción << para mostrar o imprimir un objeto *LineaPedido* en un flujo de salida. Se imprimirá el precio de venta con dos decimales, seguido de un espacio, el símbolo de moneda €, un tabulador y la cantidad. Ejemplo: `35,20␣€` TAB `3`

5.4. La clase de asociación Pedido_Articulo

- Representa la asociación bidireccional *varios-a-varios* con atributos de enlace entre *Pedido* y *Articulo*.
- Los atributos de la clase serán dos diccionarios, uno por cada sentido de la relación:

  - El que representa la asociación desde *Pedido* a *Articulo* será del tipo `std::map<Pedido*, ItemsPedido, OrdenaPedidos>` donde *ItemsPedido* será un tipo público de la clase *Pedido_Articulo* definido como un diccionario `std::map<Articulo*, LineaPedido, OrdenaArticulos>`.

    OrdenaPedidos es una clase de objetos función para ordenar los pedidos ascendentemente por número, y *OrdenaArticulos* es otra clase de objetos función para ordenar los artículos ascendentemente por referencia.
  - La asociación en el sentido inverso (desde *Articulo* a *Pedido*) se representará por un diccionario del tipo `std::map<Articulo*, Pedidos, OrdenaArticulos>`, donde *Pedidos* será un tipo público de la clase *Pedido_Articulo* definido como `std::map<Pedido*, LineaPedido, OrdenaPedidos>`.

- No es necesario definir constructores.
- Para la creación de enlaces bidireccionales entre pedidos y artículos la clase proporcionará el método *pedir*, que tendrá cuatro parámetros en el orden: pedido, articulo, precio, cantidad (por omisión, 1). Este método se sobrecargará invirtiendo los dos primeros parámetros. No devolverá nada.
- El método *detalle* devolverá la colección de artículos de un pedido (que se le pasa, por referencia) junto a su precio de venta y cantidad comprada; o sea, una referencia constante a *ItemsPedido*.
- Para obtener los enlaces en el sentido contrario se proporcionará un método *ventas*, que devolverá todos los pedidos de un artículo (que se le pasa, por referencia) con precio de venta y cantidad; o sea, un *Pedidos*.
- Se sobrecargará el operador de inserción en flujo de salida << para los tipos *Pedido_Articulo::ItemsPedido* y *Pedido_Articulo::Pedidos*. El primero, además de los detalles del pedido, mostrará el importe total y el número de ejemplares del pedido. El segundo mostrará precio, cantidad y fecha de cada venta y el importe total de las ventas del artículo. Ejemplo de salida de los detalles de un pedido:

```
  PVP   Cantidad        Artículo
================================================================
35,20 €  2              [100] "Programación Orientada a Objetos"
29,95 €  1              [110] "Fundamentos de C++"
================================================================
Total   100,35 €
```

Ejemplo de salida de pedidos:

```
[Pedidos: 3]
================================================================
  PVP   Cantidad           Fecha de venta
================================================================
39,95 €  2          miércoles 19 de abril de 2017
34,90 €  1          jueves 20 de abril de 2017
```

```
34,90 € 1          lunes  5 de abril de 2010
================================================================
149,70 €        4
```

- Un método llamado *mostrarDetallePedidos* imprimirá en el flujo de salida proporcionado el detalle de todos los pedidos realizados hasta la fecha, así como el importe de todas las ventas. Ejemplo:

  ```
  Pedido núm. 1
  Cliente: lucas          Fecha: viernes 10 de marzo de 2017
  ```

  detalle de ese pedido, como en ejemplo anterior
  Resto de los pedidos ...

  ```
  TOTAL VENTAS        981,60 €
  ```

- El método *mostrarVentasArticulos* visualizará en el flujo de salida proporcionado todas las ventas agrupadas por artículos. Ejemplo:

  ```
  Ventas de [110] "Fundamentos de C++"
  ```

  pedidos de ese artículo, como en ejemplo anterior
  Resto de los artículos vendidos ...

5.5. La clase de asociación Usuario_Pedido

Representa la asociación bidireccional *uno-a-varios* entre *Usuario* y *Pedido*. Es una clase típica de asociación. Tiene dos atributos, uno por cada una de ellas:

- Un diccionario de punteros a *Usuario* y conjunto de punteros a *Pedido*. El tipo del conjunto de punteros a *Pedido* será *Pedidos*, un tipo miembro de *Usuario_Pedido*.
- Un diccionario de punteros a *Pedido* y punteros a *Usuario*.

En la parte pública tendremos un método sobrecargado llamado *asocia*, que es el encargado de realizar las asociaciones:

- uno recibe un *Usuario* y un *Pedido*, y
- otro recibe un *Pedido* y un *Usuario*.

Además tenemos otros dos métodos:

- El método *pedidos* recibe un *Usuario* y devuelve los pedidos que haya hecho ese usuario.
- El método *cliente* recibe un *Pedido* y devuelve la dirección de memoria del usuario que ha hecho el pedido.

6. El Makefile

Se proporciona el fichero *Makefile*, donde deberá, como en la práctica 2, modificar la variable NOMBREALUMNO. Vea el enunciado correspondiente para más detalles.