Trabajo Práctico 1 — Smalltalk

[7507/9502] Algoritmos y Programación III
Curso X
Primer cuatrimestre de 2018

Alumno:	Reinaudo Dante
Número de padrón:	102848
Email:	dreinaudo@fi.uba.ar

Índice

1. Introducción

El presente informe reúne la documentación de la solución del primer trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar una aplicación de un sistema de pedidos de comida en *Pharo* utilizando los conceptos del paradigma de la orientación a objetos vistos hasta ahora en el curso.

2. Supuestos

Solo se aplica un cupón de descuento al pedido especificado, es decir, en principio no hay restricción a la cantidad de cupones que se pueden añadir a un pedido, pero estos no son acumulativos. Cada nuevo cupón que se agregue al mismo pedido reemplazara al anterior, ya sea un descuento por porcentaje o un descuento por valor fijo.

Al remover un cupón de un pedido, no se realizara un descuento en el precio final no importa la cantidad de cupones que hayan sido agregados hasta ese momento.

Si el valor del cupón de descuento es mayor al precio del pedido, este se consumirá totalmente y el pedido tendrá un precio nulo, es decir, será gratis.

No existe restricción a la cantidad de menús que pueden agregarse a un mismo pedido. Con que haya por lo menos un menú en este, ya es suficiente para anular cualquier tipo de descuento en el precio total de dicho pedido.

En caso de eliminar el único menú de cierto pedido, cualquier cupón de descuento que se haya aplicado o se aplique en un futuro tendrá validez.

Asumo que todos los clientes tendrán nombres distinguibles entre sí, con el objetivo de evitar casos bordes y confusiones al crear nuevos pedidos y/o agregar nuevos productos. A su vez asumo que todos los productos y menús, también, tendrán nombres distinguibles entre si.

3. Diagramas de clase

A continuación se muestra el diagrama de clases de la solución propuesta para abordar el presente trabajo práctico. En él se muestran todos las clases del sistema, sus atributos, sus métodos y la relación que mantienen entre si dichas clases. Cabe aclarar que en el diagrama no aparecen absolutamente todos los métodos y atributos, sino los más representativos para esclarecer la solución planteada.

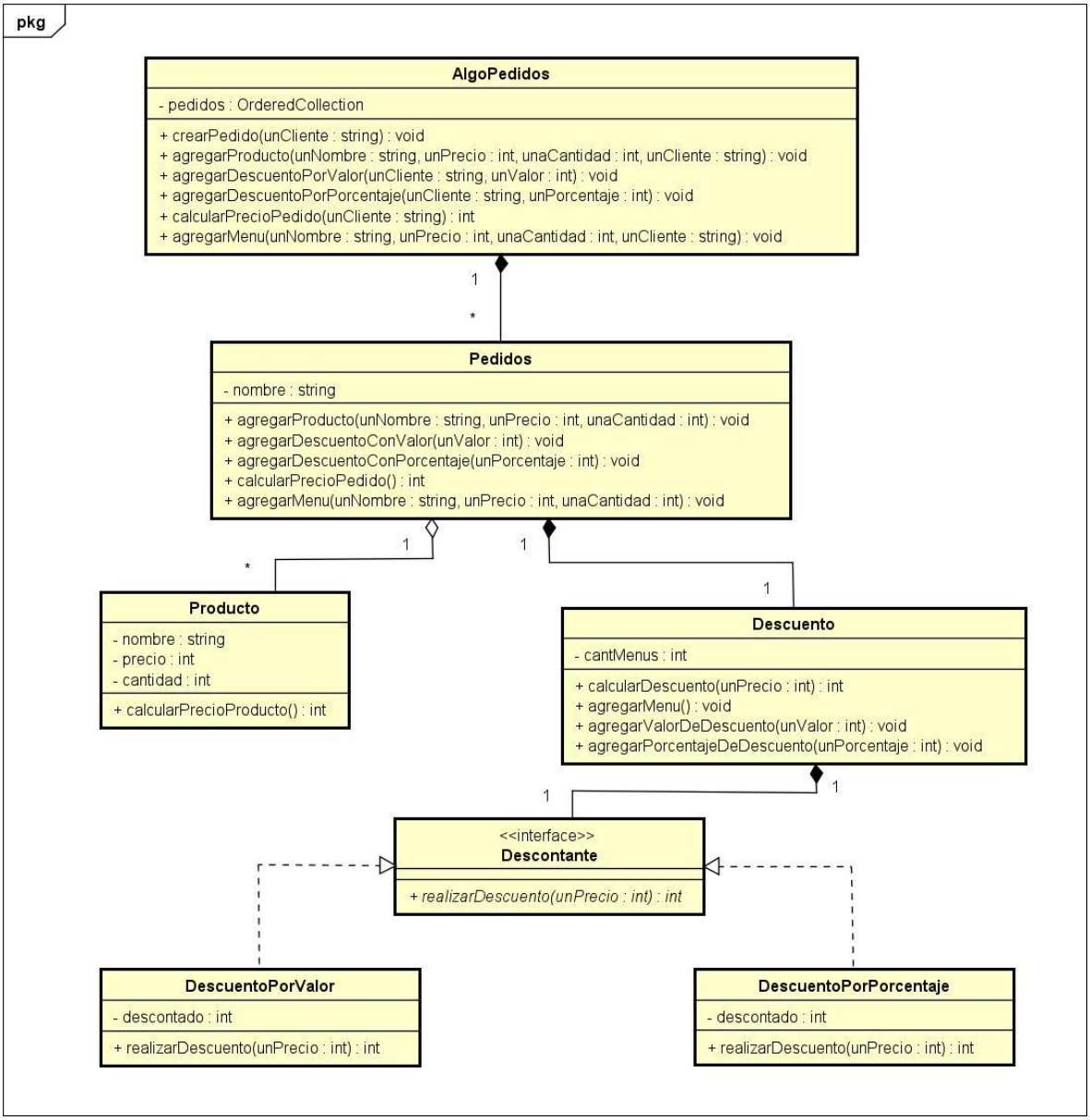

Figura 1: Diagrama de clases de AlgoPedidos.

4. Detalles de implementación

4.1. Técnicas de diseño y planteo de la solución

Para resolver este trabajo use la metodología TDD (Test-Driven Depelopment) vista en clase, esta técnica de diseño consiste en ir realizando pruebas unitarias primero, haciendo lo mínimo y suficiente para poder cumplir con las pruebas proporcionadas por la cátedra. Una vez logrado esto se refactoriza el código escrito para obtener una mayor prolijidad, y así, cumplir con las llamadas buenas prácticas de programación. Este proceso se repite una y otra vez hasta poder abarcar todos los campos y llegar a una solución muy optimizada. Esta metodología presenta muchas ventajas, las cuales no pasare a detallar dado que las vimos en clase y no son el objetivo de este informe

4.2. Explicación de las Clases utilizadas y sus métodos

Como puede verse claramente en el diagrama de clases mostrado en la sección anterior, el modelo propuesto consta de seis clases (más siete excepciones), las cuales se detallaran a continuación:

- **AlgoPedidos:** esta es la clase especificada en las pruebas de las cátedras. Cuenta con un único atributo el cual es una OrderedCollection de objetos que son instancias de la clase Pedido. Esta clase delega todas sus responsabilidades hacia la clase Pedido, ya que la mayoría de los métodos aluden a un pedido particular. Sus métodos son los mismos que se muestran en las pruebas. Además, con el fin de evitar la repetición de código, se agregó un método que permite buscar un pedido específico en la colección, con su respectiva excepción.

- **Pedido**: es la clase con más responsabilidades, cuenta con dieciséis métodos de instancia más dos métodos de clase. Presenta cuatro atributos: *nombre*, es el nombre del cliente que ordeno el pedido; *productos*, es una OrderedCollection de objetos que son instancias de la clase Producto; *entrega*, es un numero entero el cual hace alusión al formato de entrega del pedido y su recargo correspondiente, vale 1 si el pedido se retira por el local o 1.2 si se entrega con delivery, en un momento opte por resolver esto de manera polimórfica creando dos clases PedidoConEntrega y PedidoConDelivery, donde ambas entendiesen el mismo mensaje realizarRecargo, pero dado que, en este caso, ambas clases serian simplemente setters que contendrían un unico valor numérico, deseche esta alternativa; por ultimo *descuento* es una instancia de la clase Descuento, la cual describiremos detalladamente unos párrafos más adelante, pero si es importante resaltar que todo pedido tiene asociado un descuento, el cual puede ser nulo o no dependiendo el caso. A su vez, esta clase delega varias responsabilidades en las clases Producto y Descuento.

- **Producto**: es importante destacar que para el modelo planteado tanto los productos, como los menús solicitados son instancias de la clase Producto, esto se debe a que la única diferencia sustancial entre estos se haya en el descuento al precio final del pedido. Por esta razón, consideré que a la clase Producto no le corresponde saber si es del tipo producto o menú, pero a la clase Descuento si le interesa saber si hay menús en el pedido. Esta clase cuenta con los tres atributos mostrados en el diagrama, de los cuales la *cantidad* es modificable mediante un método. A su vez, la clase Pedido le delega la responsabilidad de calcularse su propio precio (*cantidad* * *precio*) y validar las excepciones correspondientes para el *precio* y la *cantidad*.

- **Descuento**: en mi opinión el cálculo de los descuentos y sus derivaciones, fue la parte más complicada del trabajo práctico, estuve varios días pensando distintas soluciones hasta que se me ocurrió crear esta clase. Esta cuenta con dos atributos, el primero es *cantMenus*, ni más menos que un contador de la cantidad de menús que hay en el pedido hasta el momento. Como se explicó anteriormente, tanto el menú como el producto son tratados como instancias de la clase Producto, pero cada vez que se añade un menú al objeto instancia de la clase Pedido, este se encarga de enviarle un mensaje a su atributo descuento, instancia de la clase Descuento, comunicadole que hay un nuevo menú, lo que aumenta en una unidad el contador. De forma análoga, si se retira un menú, el contador disminuye una unidad. De esta manera, cuando se solicita conocer el precio de un pedido que contiene menus, la instancia de la clase Descuento, que recibe el mensaje calcularDescuento, simplemente retorna el mismo parámetro recibido. Por otro lado se eligió trabajar con un contador en lugar de un booleano hayMenu, dado que esto permite al modelo comportarse de acuerdo a los supuestos en casos bordes, como por ejemplo si se agregan varios menús y luego se retiran de a uno, o si se retira un menú de un pedido que ya tenía un cupón de descuento. Su segundo atributo es *tipoDeDescuento,* este es una instancia de la clase DescuentoPorValor o de la clase DescuentoPorPorcenaje dependiendo el caso, donde ambas clases comprenden el mismo mensaje realizarDescuento. De esta forma, las distintas maneras de calcular el descuento se resuelven polimórficamente y la clase Descuento delega esta responsabilidad enviándole un mensaje a su atributo *tipoDeDescuento*. Por otro lado, la clase Pedido le delega la responsabilidad de calcular el descuento al precio del pedido, de tener una noción de la cantidad de menús que hay en el pedido y la validación de las excepciones correspondientes para los valores y porcentajes.

- **DescuentoPorValor:** cuenta con un atributo *descontado*, el cual utiliza para realizar el descuento a un precio recibido (precio - *descontado*). Por otro lado, como vimos unos párrafos más arriba, al crear una instancia de la clase Pedido también se crea una instancia de la clase Descuento asociada a dicho pedido. A su vez, toda instancia de la clase Descuento tiene asociado un atributo que puede ser instancia de dos clases diferentes. Dado que al crear cualquier objeto debo dejarlo en un estado valido, por elección arbitraria opte que al crear una instancia de la clase de Descuento o al remover un cupón del pedido, su atributo *tipoDeDescuento* sea una instancia de la clase DescuentoPorValor, cuyo atributo *descontado* sea igual a cero. De esta manera, para calcular el precio de un pedido al cual no tiene un descuento explicito, dado que nunca se agregó uno o este fue removido, el atributo tipoDeDescuento, que es una instancia de la clase DescuentoPorValor, al recibir el mensaje realizarDescuento, retorna el precio recibido menos el descontado que es nulo, resultando así un precio igual al recibido. Esto es lógico dado que no se especificó que haya ningún tipo de descuento. Por otro lado, la clase Descuento le delega la responsabilidad validación de parámetro de valor recibido el lanzamiento de su excepción correspondiente.

- **DescuentoPorPorcentaje:** cuenta con un atributo *descontado*, el cual utiliza para realizar el descuento a un precio recibido (precio – (1 – (*descontado*/100)). Por otro lado, la clase Descuento le delega la responsabilidad de la validación del parámetro de porcentaje recibido y el lanzamiento de su excepción correspondiente.

5. Excepciones

Excepción PedidoNoEncontradoError. En una gran parte de los métodos, debo trabajar con los pedidos a nombre de un cliente, se lanza cuando no existe ningún pedido con el nombre especificado.

Excepción ProductoNoEncontradoError. Es lanzada cuando se busca un producto de un determinado pedido con cierto nombre y este no es encontrado o no existe. Esta misma excepción también es válida para los menús no encontrados.

Excepción ParametroNoEsUnNumeroError. Dado que se pasa por parámetro diversas variables que debieran ser numéricas, las cuales se utilizan para realizar varias operaciones, es necesario validar que efectivamente estas variables sean numéricas. Se utiliza para validar precio, cantidad, porcentaje y valor.

Excepción CantidadInvalidaError. Aparece si la cantidad recibida es un número fraccionario o si es un número menor a 1.

Excepción PrecioInvalidoError. Se lanza en caso de que el precio recibido sea un número menor o igual a 0.

Excepción PorcentajeInvalidoError. Es lanzada cuando el porcentaje recibido es menor a 0 o es mayor a 100.

Excepción ValorInvalidoError. Se manifiesta si el valor del cupón de descuento recibido por parámetro es un número menor a 0.

6. Diagramas de secuencia

Con el fin de comunicar y esclarecer un poco más la solución planteada para el sistema, en esta sección del informe, se presentaran algunos diagramas de secuencia donde se muestran distintas situaciones posibles que comprende el modelo. En ellos se podrá observar los mensajes enviados entre los distintos objetos del sistema a lo largo del tiempo. Es importante resaltar que en dichos diagramas no aparecen absolutamente todos los mensajes que llevan a cabo los objetos si no los más representativos, facilitando su lectura y comprensión dado que una de las funciones principales de estos diagramas es comunicar. Al mismo tiempo, serán de gran ayuda para complementar la información que aparece en el diagrama de clases.

En ambos diagrama podremos observar varios de los aspectos descriptos en la sección detalles de implementación, como, por ejemplo, que al crear una instancia de la clase Pedido se crea una instancia de la clase Descuento asociada a esta, o como que tanto los productos y los menús son tratados como instancias de la clase Producto, entre otros .

En el siguiente diagrama, compuesto tanto por la figura 2 como por la figura 3, se muestra una situación donde un actor, el cual podría ser un usuario, realiza un pedido al nombre de unCliente, luego agrega un producto con su respectivo nombre, precio y cantidad para este mismo cliente, posteriormente, agrega un cupón de descuento porcentual a dicho pedido y, por último, solicita el precio total del pedido. Podemos observar claramente la interacción entre los objetos del modelo. Por otro lado, dado que no existe un menú en el pedido, al calcular el precio del pedido se devuelve un valor precioConDescuento.

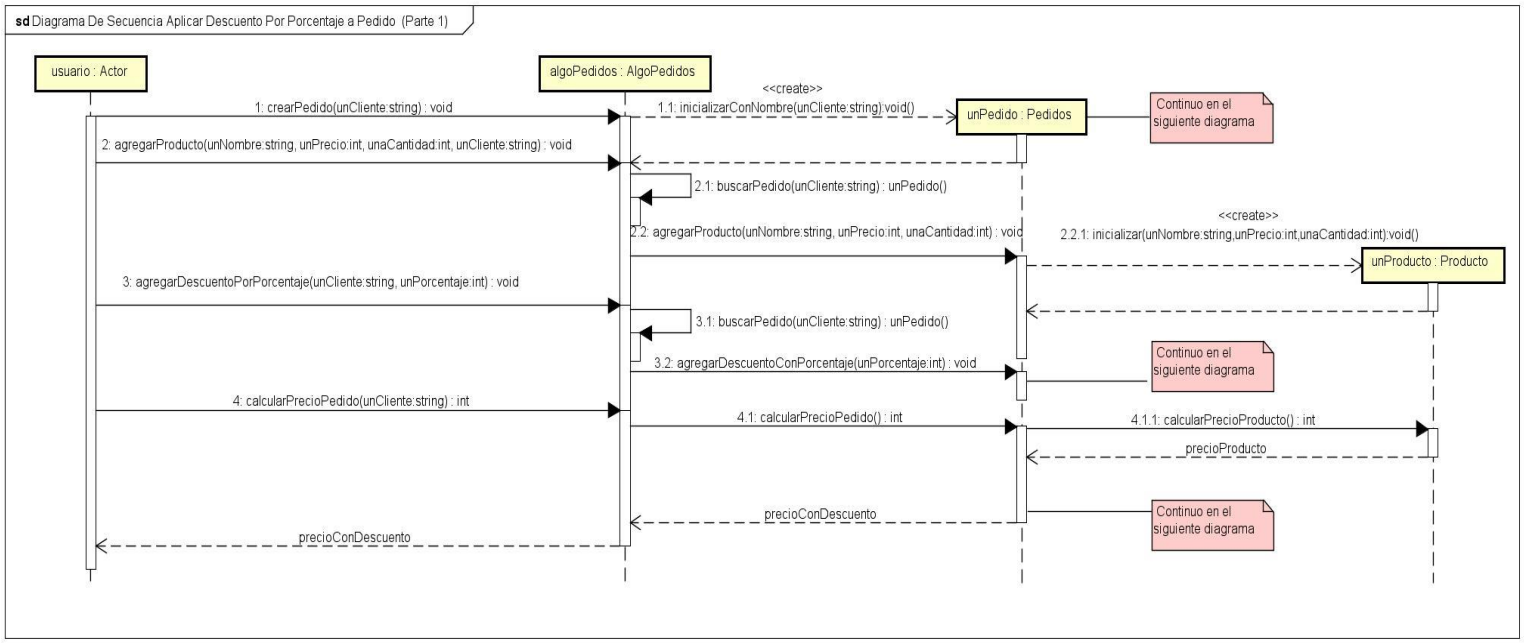

Figura 2: Diagrama de secuencia aplicar descuento por porcentaje (Primera Parte).

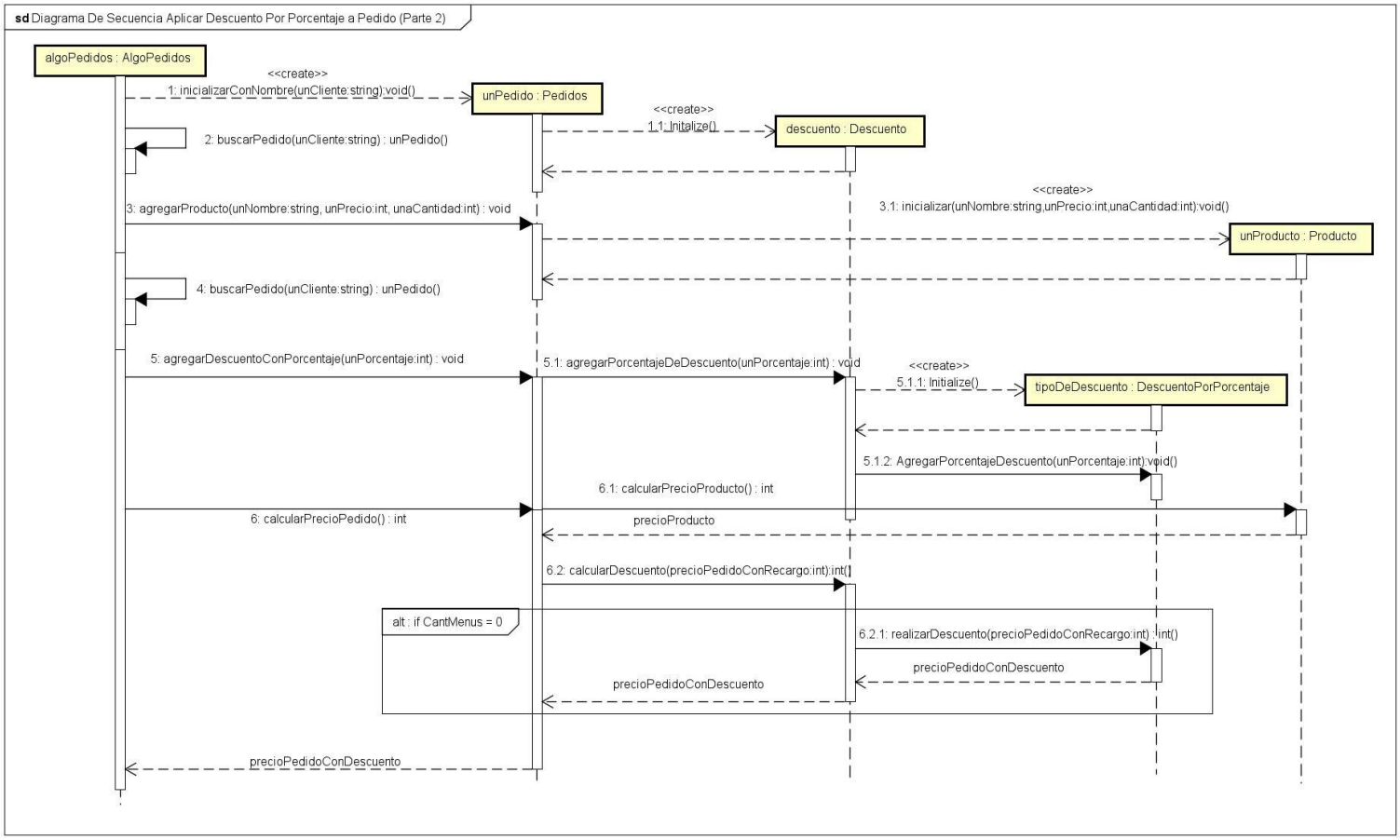

Figura 3: Diagrama de secuencia aplicar descuento por porcentaje (Segunda Parte).

En el siguiente diagrama, conformado por las figuras 4 y 5, se muestra una situación similar a la anterior, un actor realiza un pedido al nombre de unCliente, pero esta vez agrega un menú en lugar de un producto, también, con su respectivo nombre, precio y cantidad. Posteriormente, agrega un cupón de descuento por valor a dicho pedido y, por último, solicita el precio total del pedido. Dado que en este caso, el pedido contiene menús al calcular el precio del pedido, se devuelve un precioPedidoSinDescuento, y la instancia de la clase Descuento ni siquiera le envía el mensaje realizarDescuento a su atributo tipoDeDescuento.

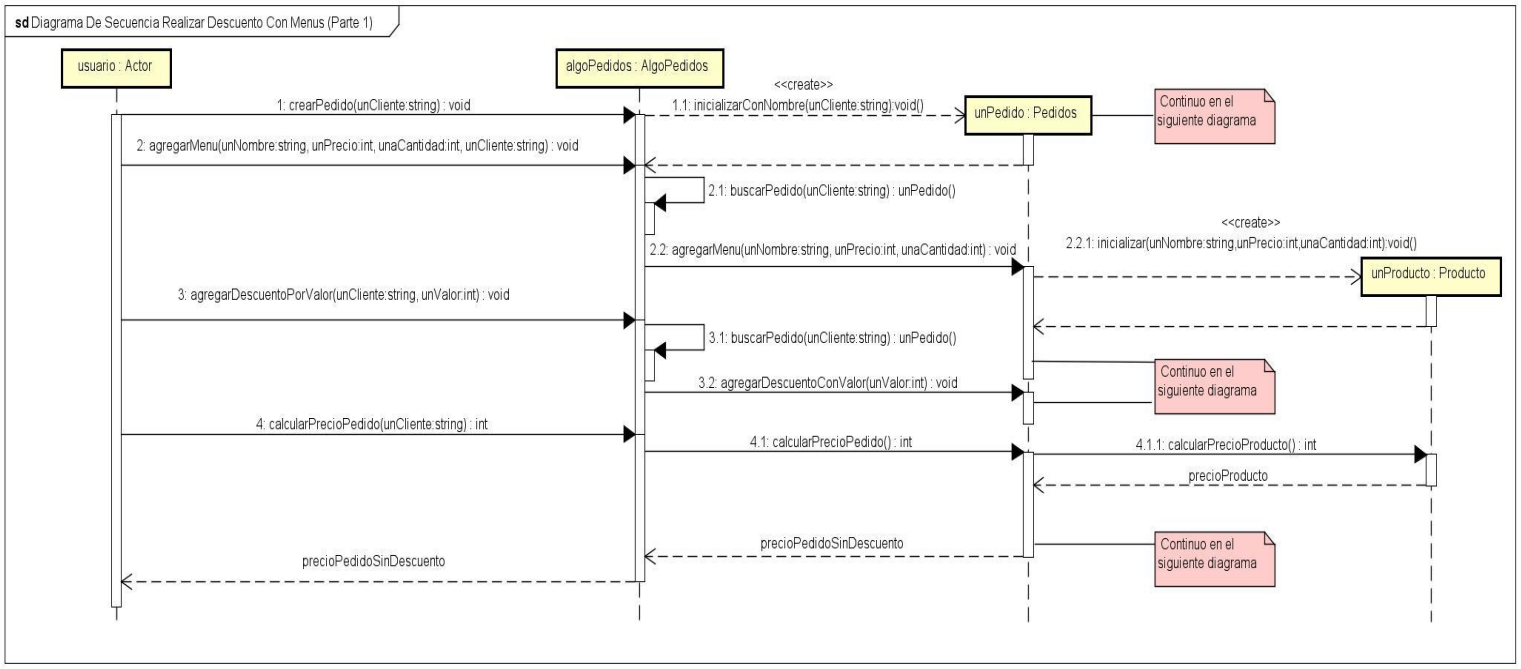

Figura 4: Diagrama de secuencia realizar descuento con menú (Primera Parte).

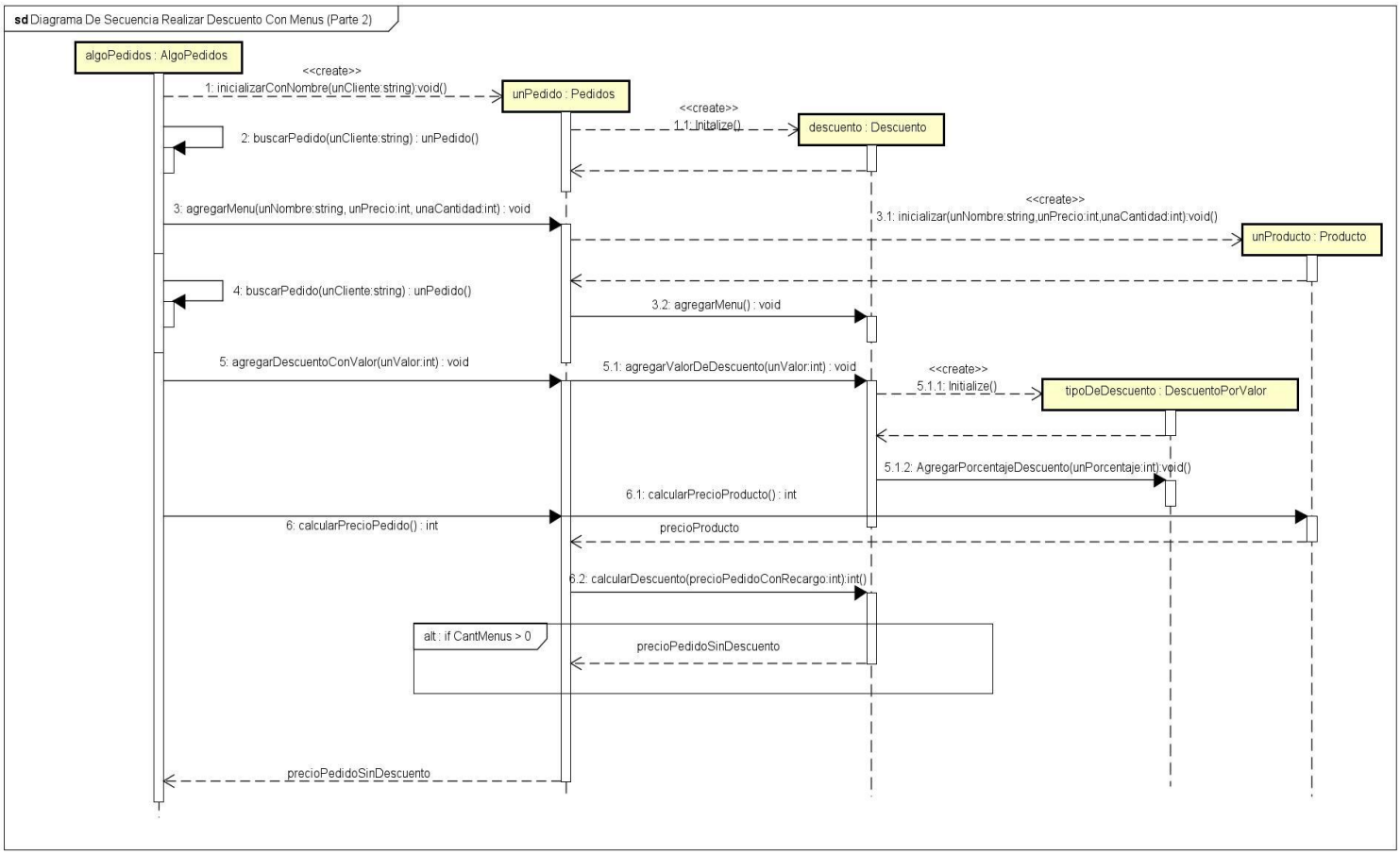

Figura 5: Diagrama de secuencia realizar descuento con menú (Segunda Parte).