Trabajo Práctico 1 — Smalltalk

[7507/9502] Algoritmos y Programación III
Curso 1
Segundo cuatrimestre de 2020

Alumno:	VARGAS, Marcos
Número de padrón:	104992
Email:	mlvargas@fi.uba.ar

Índice

1. Introducción

El presente informe reune 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 un local gastronómico en Pharo utilizando los conceptos del paradigma de la orientación a objetos vistos hasta ahora en el curso.

2. Supuestos

Debido a que se presentaron situaciones que quedaron a elección del alumnado se consideraron las siguientes suposiciones:

- Solo se puede ingresar un cupón por pedido. Esto es debido a que generalmente uno no puede entregar una cantidad extendida de cupones para solamente pagar 1 % del monto final.
- Un cliente solo puede tener un pedido, ya sea para retirar o con delivery, si se desea crear otro pedido al mismo se informará el error.
- No se pueden agregar productos con el mismo nombre.
- Al solo poder agregar un cupón, si se agrega uno a un pedido al que ya se le fue asignado, se sustituirá al anterior.

3. Detalles de implementación

3.1. Sobre los pilares de POO

Para el programa se utilizó la abstracción para crear todas las clases ya que de cada objeto necesitaba ciertas características particulares, como por ejemplo, en el caso del objeto Producto las tres características que necesité fueron precio, cantidad y nombre aún sabiendo que un producto tiene muchas más características. La ventaja de haber utilizado la abstracción fue que el objetivo quedó más centrado.

Se utilizó el encapsulamiento para todo el programa ya que cada clase tiene sus responsabilidades propias, como por ejemplo almacenar los productos es responsabilidad de la clase Pedido y no de la clase Cliente ni de AlgoPedidos, por ende el acceso a los atributos de las clases es para ellas mismas. La ventaja de hacer esto es que se puede delegar mejor las responsabilidades. Un ejemplo de su uso en el programa es en el objeto Descuento, en AlgoPedidos no se puede usar un descuento si hay un menú en el pedido, por ende es Descuento quien sabe y debe saber que no se puede aplicar y no Pedido.

Se utilizó delegación, por ejemplo, en AlgoPedidos. Delega la responsabilidad de procesos a los diferentes objetos, como por ejemplo agregar un producto a un pedido, la orden la recibe el objeto Pedido y el mismo se comunica con el objeto Producto. La ventaja de hacer esto es que AlgoPedidos no es modificado a lo largo del tiempo si un proceso varia, los únicos objetos que sufririan variaciones son los involucrados.

Sobre polimorfismo, se utilizó en dos objetos, Descuento y Entrega. Un Descuento, en este programa, puede tener hasta tres versiones diferentes, las cuales son: DescuentoValorFijo, DescuentoConPorcentaje y DescuentoNulo. ¿Por qué utilicé polimorfismo de esta manera? Porque las tres versiones de Descuento son capaces de entender el mismo mensaje y cada una responder de la manera que deben; por ejemplo, un descuento con valor fijo solo se debe restar al monto total, un descuento con porcentaje primero debe saber sobre que monto aplicar el porcentaje y conocer su valor para restarse al mismo monto, y por ultimo el descuento nulo, el mismo no deberia de restar algo al monto, entonces su respuesta al mismo mensaje que reciben las tres versiones responde restando 0 al monto final.

Para Entrega sucede algo similar, para el local (y para la mayoria de locales) existen dos tipos de entregas, con delivery o sin delivery, si bien ambas son entregas su costo es diferente, por lo que decidí aplicar polimorfismo para que dos entregas que se comportan distinto entiendan el mismo mensaje de aplicar recarga al monto y respondan lo que le corresponde a cada una,

EntregaConDelivery le sumará el costo al monto total y EntregaEnLocal no le sumará nada ya que no tiene recargo ese tipo de entrega.

La ventaja de usar este pilar fue que se evitó el código repetido y que además en el caso de cambiar la clase madre, sus subclases también cambian lo que comparten, y lo que no comparten solo se tiene que cambiar en su subclase correspondiente sin afectar al resto.

3.2. Sobre el código

DescuentoNulo, si bien entiende los mensajes aplicarDescuento:, conValor: y validarValorIngresado:, sus respuestas siempre seran:

```
aplicarDescuento: unPedido.
    ^ 0.
```

Donde 0 es lo que se le resta a al monto final.

Y en el caso de querer asignarle un valor, podra responder al mensaje conValor: de la misma manera que DescuentoValorFijo y DescuentoConPorcentaje pero en esta subclase la respuesta a validarValorIngresado siempre será:

```
validarValorIngresado: unValor
    NoSePuedeAsignarValorADescuentoNuloError signal.
```

Para la mayoría de clases en este programa se validaron los valores ingresados (clientes, productos, valores, etc.) utilizando mensajes de clase como en el siguiente ejemplo:

```
conNombre: unNombre precio: unPrecio yCantidad: unaCantidad
    self validarValorIngresado: unPrecio.
    self validarValorIngresado: unaCantidad.

    nombre := unNombre.
    precio := unPrecio.
    cantidad := unaCantidad.

validarValorIngresado: unValor
    (unValor <= 0) ifTrue: [ ValorInvalidoError signal ].
```

4. Excepciones

ElClienteYaTieneUnPedido La misma tiene el objetivo de informar cuando se intenta crear un pedido, para retirar o con delivery, a un cliente que ya tiene un pedido creado.

ClienteNoTienePedido En este caso se informa cuando se desea modificar información del pedido de un cliente que no tiene pedido creado.

NoExisteProducto Esta excepción informa el error que se produce al querer modificar un producto que nunca fue declarado.

ValorInvalido La misma informa cuando se ingresa un precio negativo o igual a cero; una cantidad negativa o igual a cero; un porcentaje menor o igual a cero, mayor o igual a cien; un valor de cupon fijo menor a 0.

YaFueAgregadoElProducto Esta excepción se lanza cuando se intenta agregar un producto que ya se encuentra en un pedido.

NoSePuedeAsignarValorADescuentoNulo Informa cuando se le intenta asignar un valor a DescuentoNulo.

5. Diagramas de clase

5.1. Diagrama general

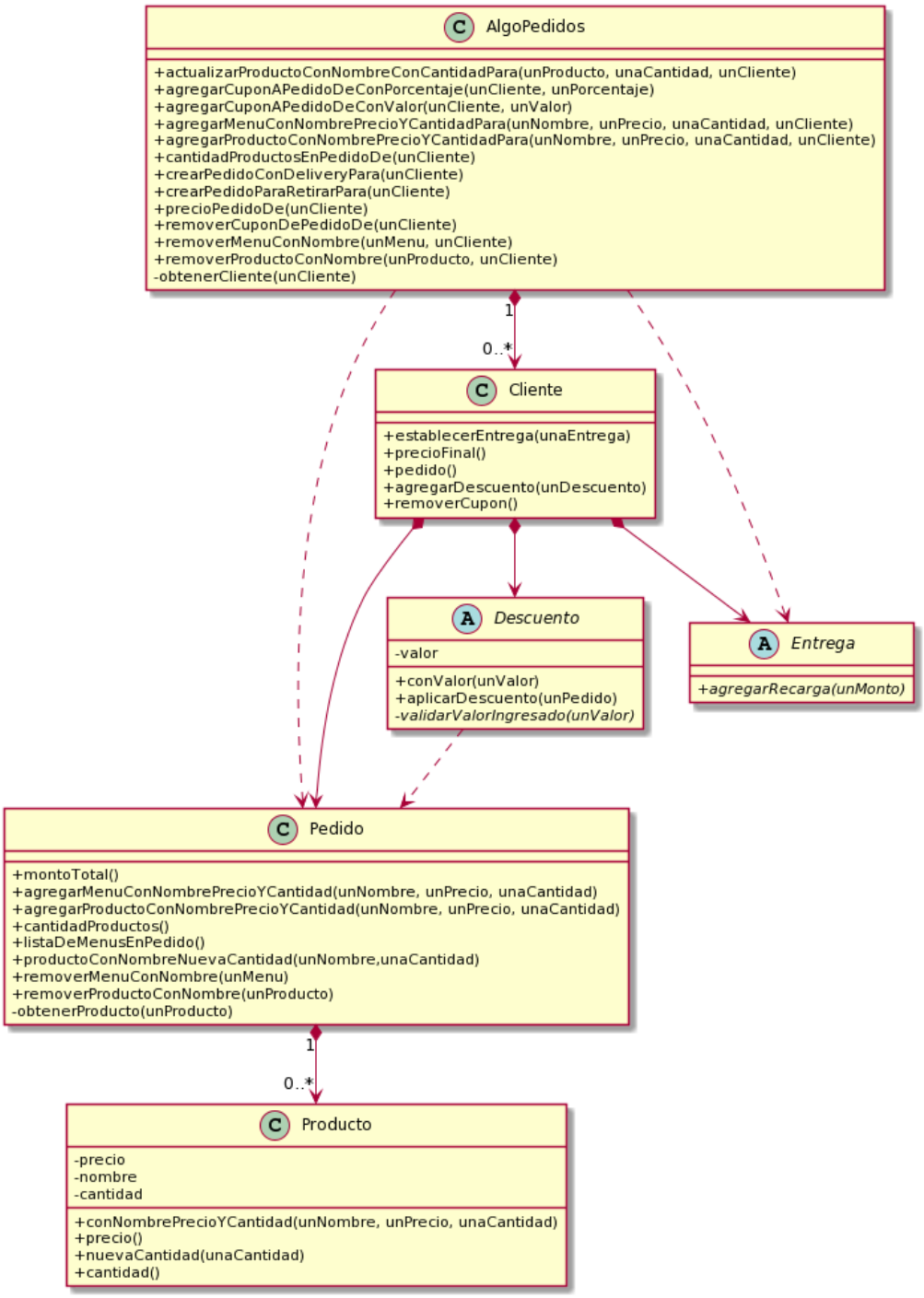

Figura 1: Diagrama de clases general.

5.2. Jerarquía de clases

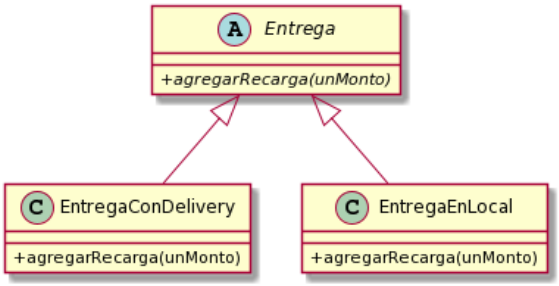

Figura 2: Diagrama de jerarquía de Entrega.

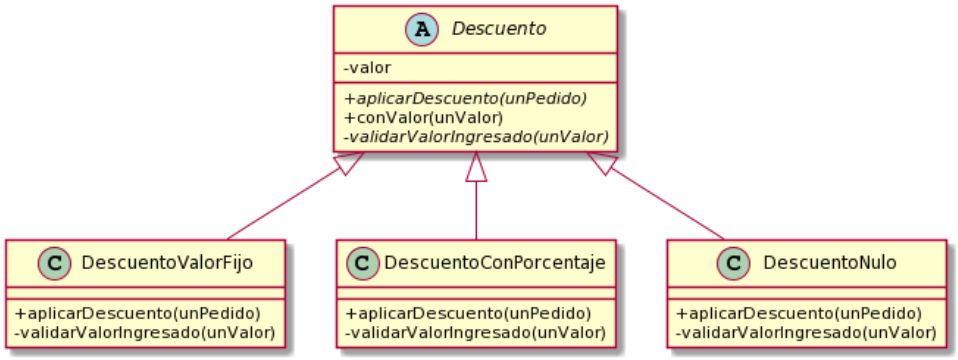

Figura 3: Diagrama de jerarquía de Descuento.

6. Diagramas de secuencia

6.1. Creación de pedido con delivery

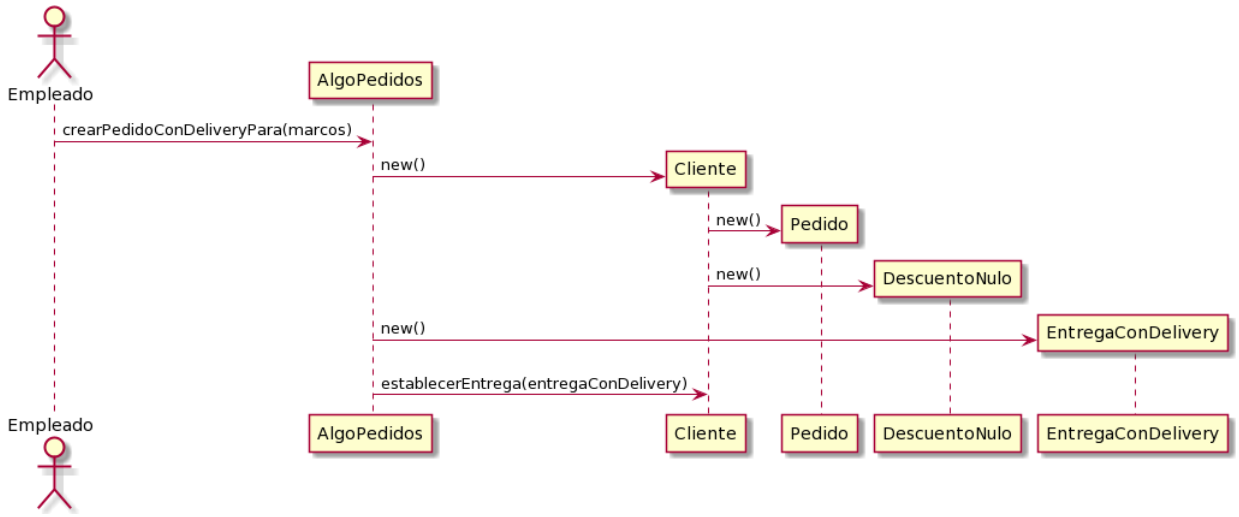

Figura 4: Secuencia de creación de pedido con delivery.

6.2. Eliminación de producto inexistente

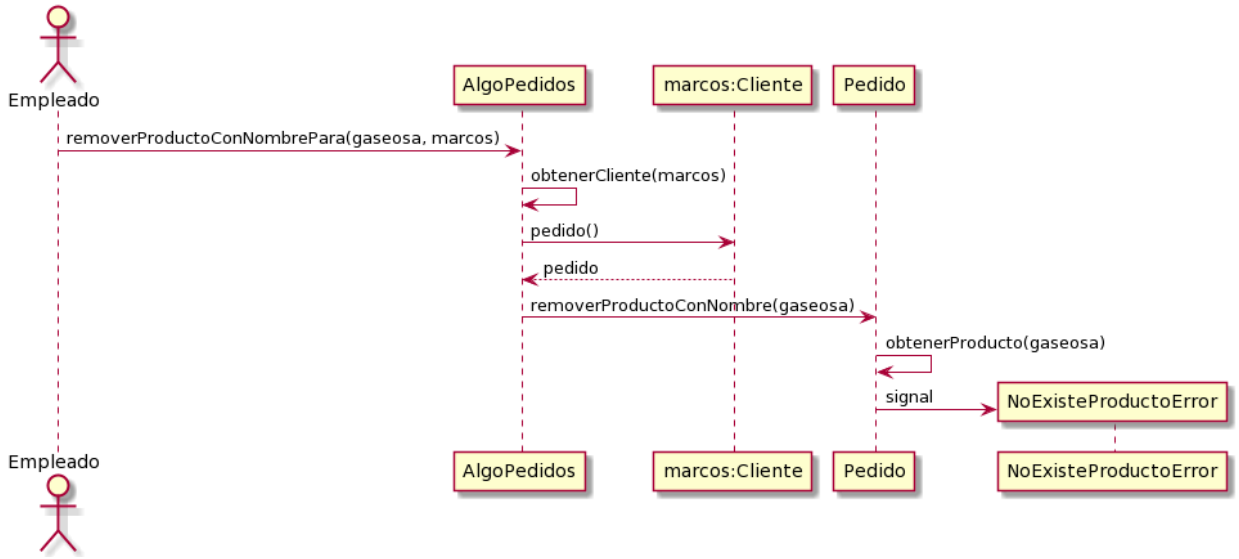

Figura 5: Secuencia de eliminación de producto.

6.3. Obtención de cantidad de productos de pedido

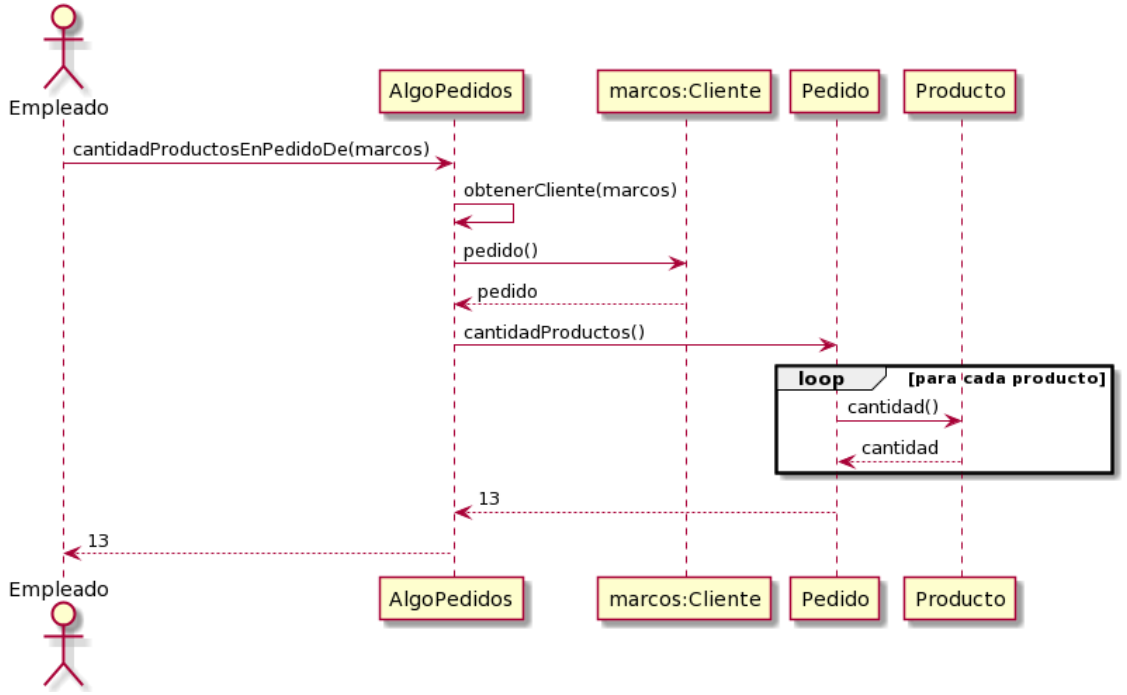

Figura 6: Secuencia de obtención de cantidad de productos.

6.4. Obtención de precio de pedido

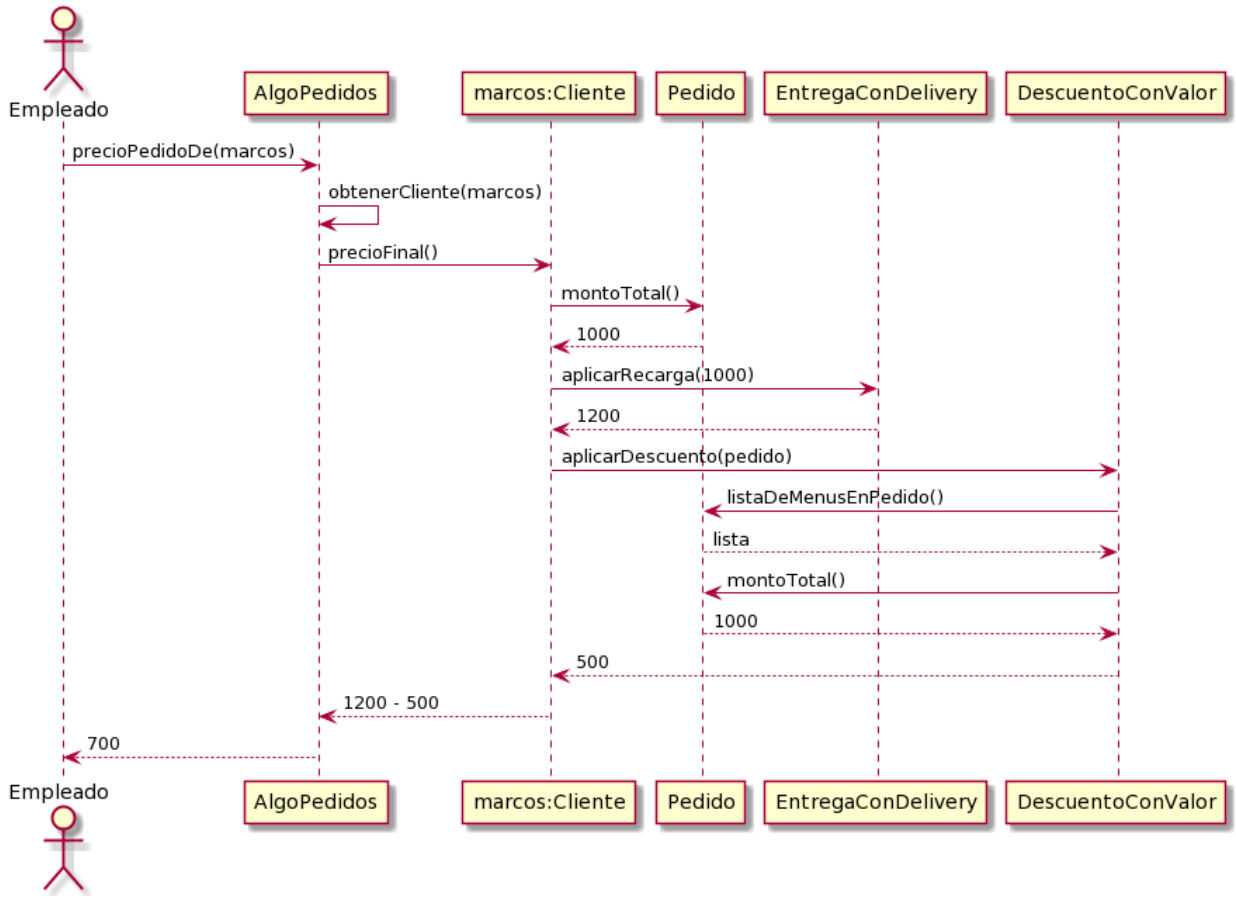

Figura 7: Secuencia de obtención de monto final de un cliente.

En esta secuencia se muestra como es el procedimiento para conocer el precio final del pedido de un cliente. En este caso el cliente tiene la lista de menus vacía y por eso se devuelve el valor del descuento fijo, que es 500, para que se reste con el monto total. En caso de que tuviese un menu en su pedido se devolvería 0 y no se aplicaria ningún descuento al monto.