Trabajo Práctico 1 - Smalltalk

[7507/9502] Algoritmos y Programación III
Curso 1

Mateo Rojas
104985
marojas@fi.uba.ar

Introducción:

En el presente trabajo práctico se implementa un programa que realiza pedidos de comida, tanto como para retirar como con delivery. Se le pueden agregar distintos cupones de descuento y el usuario podrá manejar las cantidades de los productos. El objetivo es familiarizarse con la Programación Orientada a Objetos por ende se hace uso de todas sus principales propiedades.

Supuestos:

1. Si en un pedido con determinado precio se le aplica un cupón de descuento fijo con valor mayor al valor total, el pedido será gratis.
2. Un cupón porcentual no puede tener porcentaje mayor a 100.
3. Una vez ejecutada la acción precioPedidoDe() se da por terminada la ejecución del programa y no se podrá volver a pedir el precio.

Diagramas de Clase:

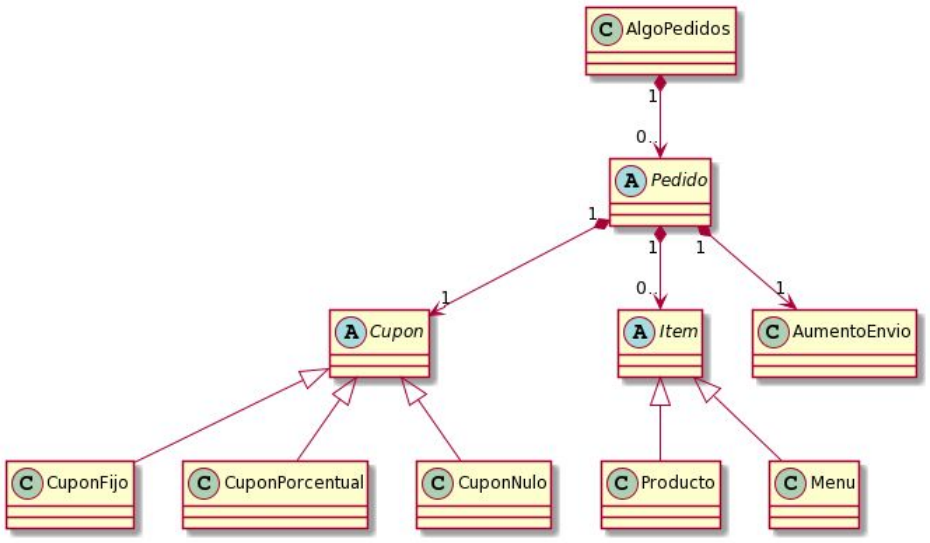

fig 1: diagrama general

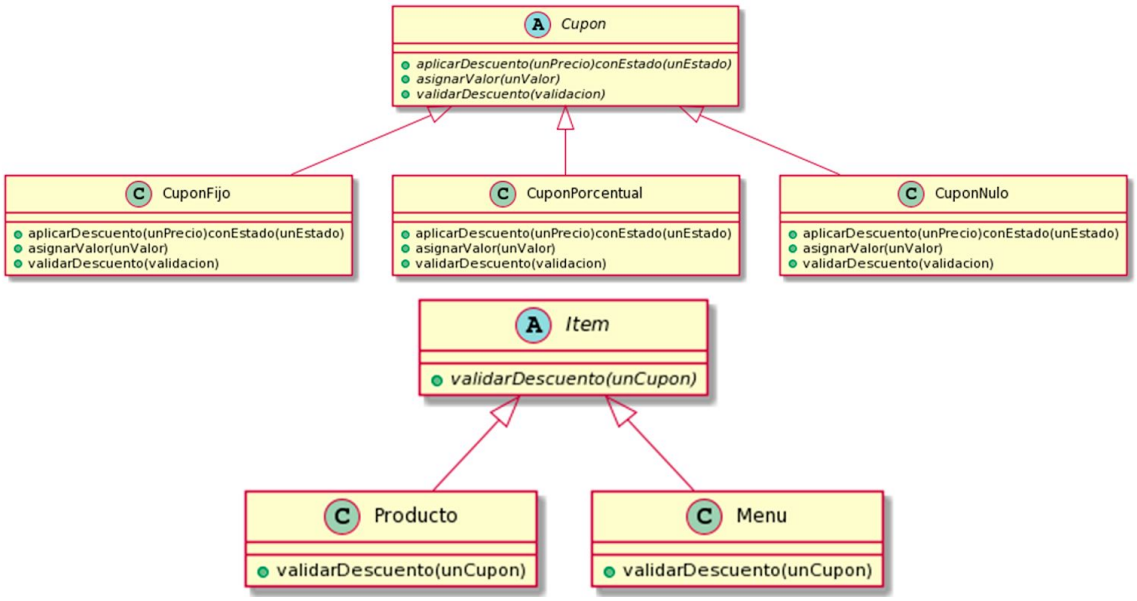

fig 2: Dos veces donde se usó polimorfismo

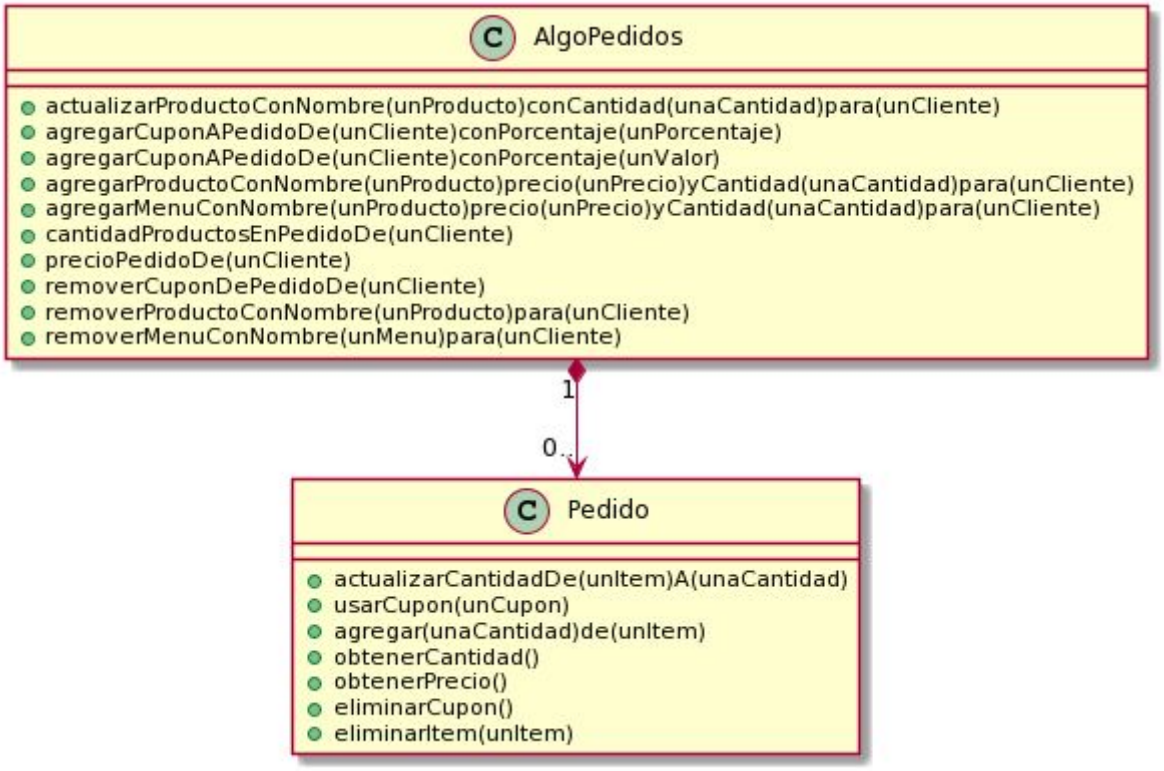

fig 3: Delegación

Aclaración: no consideré necesario mostrar los atributos de las clases para simplificar los diagramas.

Detalles de Implementación:

Al estar orientado a objetos, este programa se basa en clases. Comenzamos por AlgoPedidos, la clase obligatoria para que todo marche. Esta clase es la que recibe las

instrucciones del usuario y se encarga del trabajo principal. Es aquí donde podemos ver claramente una delegación. Básicamente para cada método de la clase AlgoPedidos existe un método de la clase pedido, la cual se encarga de hacer esta determinada tarea (ver fig 3), en esta imagen vemos los métodos de AlgoPedidos y los respectivos métodos para el pedido.
La delegación nos facilita mucho la comprensión de nuestro código a su vez que ayuda a orientar a objetos el programa.
En el caso de la herencia, podemos ver que se usó varias veces a conveniencia (ver fig 1). Hay una clase abstracta Item que tiene dos hijas, Producto y Menu. También está Cupon, con CuponFijo, CuponPorcentual y CuponNulo. Use herencia en estas clases ya que quería evitar la reutilización de código, a la vez que esta nos permite facilitar la extensión de aplicaciones ya que las clases hijas pueden usar métodos y atributos de las clases madres.
La herencia nos lleva a otro gran pilar, el Polimorfismo. Este nos permite hacer que diferentes clases entiendan el mismo mensaje y realicen funciones diferentes aparte de simplificar el código. Aquí se usó bastante (ver fig 2), podemos ver que los cupones aplican el descuento de manera diferente. También podemos ver que cada ítem actúa de manera diferente a la hora de validar un descuento, considero que al aplicar funciones completamente diferentes, Producto y Menu fueron correctamente separadas, permitiendo, como se pedía, que cada ítem se encargue de habilitar o deshabilitar el descuento. Esto también aplica al encapsulamiento porque como anteriormente mencione, el pedido no sabe nada de sus ítems y estos hacen todo el trabajo.

Excepciones:

Aclaración: para facilitar la legibilidad de código, las excepciones actúan de varias formas, es decir a dos errores parecidos, la excepción será la misma.

- PedidoNoEstaError: Como dice su nombre, este error aparecerá cuando se quiere acceder a un pedido que no fue creado.
- ItemNoEstaError: Exactamente lo mismo que el otro, solo que con items (menu o producto)
- ValorInvalidoError: Este error aparecerá varias veces. Cuando se ingresen precios, cantidades y porcentajes menores a cero. También cuando los porcentajes sean mayores a cien.

Diagramas de Secuencia:

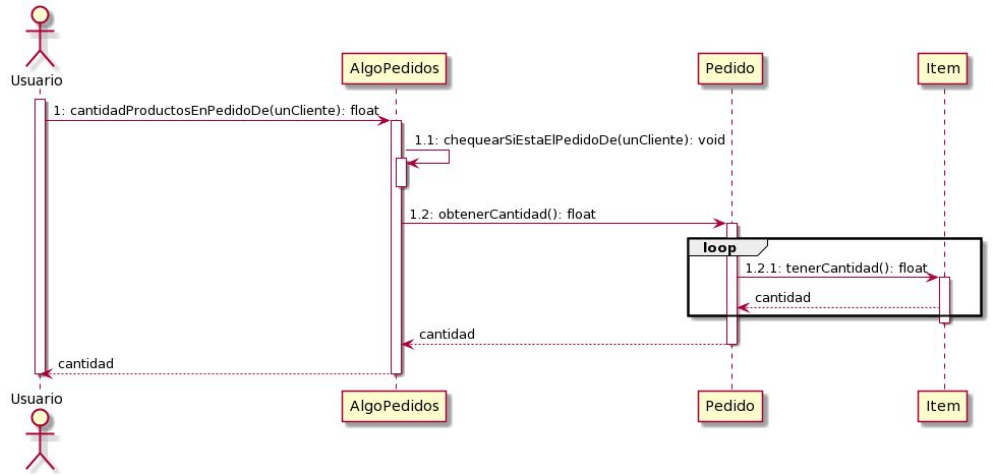

fig 4: cantidadProductosEnPedidoDe

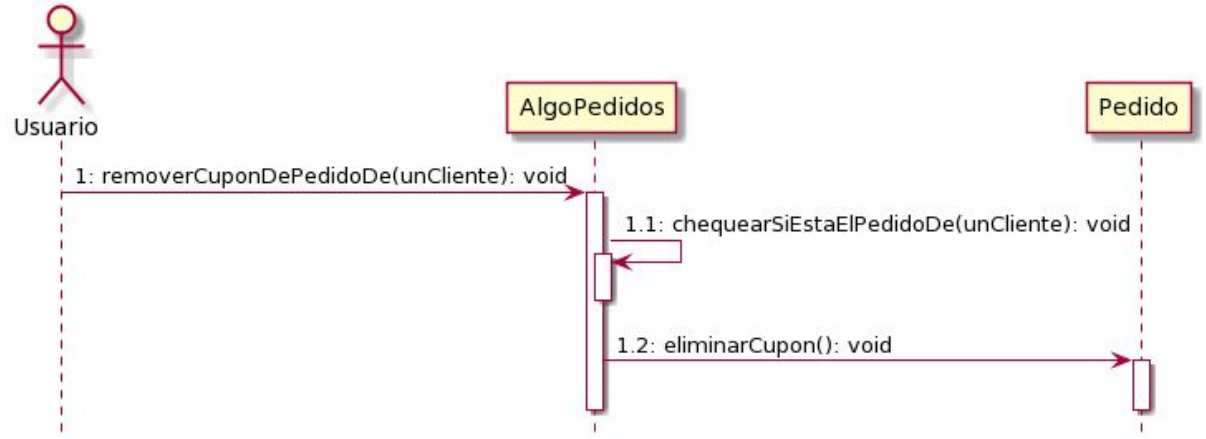

fig 5: removerCuponDePedidoDe