Trabajo Práctico 1 — Smalltalk

[7507/9502] Algoritmos y Programación III
Curso 1
Segundo cuatrimestre de 2017

Alumno:	del Mazo, Federico
Número de padrón:	100029
Email:	federicodelmazo@hotmail.com

Índice

1. Introducción **2**

2. Supuestos **2**

3. Modelo de dominio **2**

4. Diagramas de clase **2**

5. Detalles de implementación **3**
5.1. AlgoBay 3
5.2. Compra 4
5.3. Adicional 4

6. Excepciones **4**

7. Diagramas de secuencia **5**

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 comercio electrónico en Pharo utilizando los conceptos del paradigma de la orientación a objetos vistos hasta ahora en el curso.

2. Supuestos

El mayor supuesto encontrado es a la hora de modificar el precio de una compra. ¿Cuál es la mejor forma de modelar que una compra pueda ser modificada por distintos conceptos? Pensarlo de manera que solo funcione para una compra con envío o una compra con garantía termina pareciendo una forma muy cerrada de plantearlo. En cambio, la solución propuesta es una clase nueva dedicada específicamente a estas 'especialidades' en las compras, la clase **Adicional**. Sea un cupón de descuento o un cobro de envío, la idea es crear una forma genérica de que la compra pueda acceder a distintos adicionales, que sean intercambiables y combinables, y pueda aplicarlos todos sobre su propio precio. De esta forma, si mañana se quisiese, por ejemplo, agregar funcionalidad a "Anticipo", "Pago en cuotas" u otros, podría hacerse sin tener que indagar en la clase Compra, ya que ahora hay una clase dedicada específicamente a ello.

3. Modelo de dominio

AlgoBay La clase principal del trabajo modela un mercado y tiene como responsabilidad ser la intermediaria entre todo objeto presente. Si se crea un producto, una compra, un cupón o demás, estos tienen que pasar por esta clase. De esta forma, se tiene un registro de cuanto cuesta cada producto, en que compra están situados, y no tenerlos a todos estos trabajando independientemente sin moderador.

Producto Los productos son los bienes que van a ser comprados por las compras, dentro de los mercados.

Compra Ya con un mercado creado con productos dentro, todo bien que yo vaya a comprar tiene que ser situado en el objeto compra para poder finalizar la transacción

Adicional Siempre que una compra tenga un modificador especial que le cambie el precio, sea para aumentar o disminuir, tendrá un adicional. Actualmente hay implementados 3 adicionales como subclases de esta clase abstracta:

Envío: El cobro por envío de las compras adiciona un valor a la compra

Garantía: El cobro de una garantía adicionará un porcentaje de la compra a esta

Cupón: Un cupón de descuento disminuirá en un porcentaje de la compra cuanto es el valor de esta misma

4. Diagramas de clase

Para el diagrama de clase muestro como todo pasa por la clase AlgoBay. Crear una compra es en contexto de un mercado, crear un producto y agregarlo a la compra también. Por otro lado tenemos las tres clases que heredan de Adicional, abstracta, que son agregados a la clase Compra, según lo requiera AlgoBay a la hora de crearla.

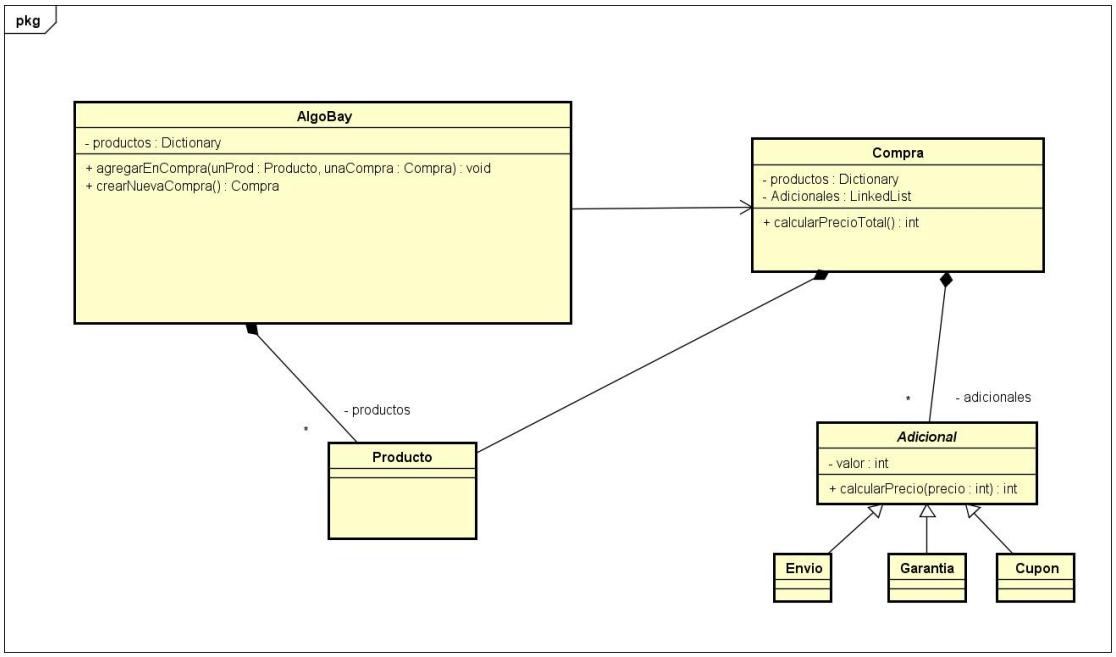

Figura 1: Diagrama conciso del programa.

5. Detalles de implementación

5.1. AlgoBay

Surgió una cuestión de cómo implementar la idea de que no haya cupones de descuento acumulables. El problema surge en que hay que asumir de quien es la responsabilidad de decidir cual es el cupón vigente. Sea del mercado, de la compra, o del mismo cupón, de alguna forma hay que encargarse de que no se pisen dos cupones en simultaneo. La solución propuesta para esto fue que el mercado, AlgoBay, sea el encargado de que cada vez que agregue un cupón a una compra, retire el presente y compare cual es el vigente de los dos, en este caso, el más grande. Luego, simplemente adiciona el cupón a la compra. De esta forma, la compra misma no es la encargada de recibir dos cupones e inestablemente elegir el mejor para ella, sino que solamente recibe las ordenes de arriba.

```
AlgoBay>>> agregarCupon: enCompra:
|cupon_actual mejor_cupon|
cupon_actual := unaCompra retirarCupon.
mejor_cupon:= unaCompra elegirMejorCupon: cupon_actual y: unCupon.
unaCompra agregarAdicional: mejor_cupon
```

Un segundo supuesto planteado es en el de los valores iniciales de los objetos creados por las pruebas, en este caso, envíos y garantías. En el contexto del trabajo, la garantía es siempre del 10 % y el envío es siempre de $100. La implementación propone que al momento de crear cada adicional, en AlgoBay, se permita seleccionar el valor para este objeto, de manera que no este dentro del código del adicional, pero si del mercado, y de esta forma solo tenga que modificarse la creación de este y no el código base. Para concluir con esta implementación, se agregaron las excepciones correspondientes a poner valores no permitidos.

5.2. Compra

El mayor problema en la clase **Compra** fue el cálculo de los precios, teniendo en cuenta los adicionales de la compra. La decisión tomada fue la de armar una lista (**LinkedList**) de los adicionales de la compra, y por un lado tener el método **calcularPrecioBruto** y el resultado de este ser el que utilice **calcularPrecioTotal** iterando sobre la lista de adicionales.

```
Compra >>> calcularPrecioBruto
|sumatoria|
sumatoria := 0.
productos valuesDo: [:producto | sumatoria:=sumatoria + (producto getPrecio)].
^ sumatoria
```

```
Compra >>> calcularPrecioTotal
|precio|
precio:= self calcularPrecioBruto.
adicionales do: [:atributo | precio:= atributo calcularPrecio: precio].
^ precio.
```

5.3. Adicional

Lo complejo de la clase adicional fue como hacer que cada subclase pueda responder al mismo mensaje, el aplicado por compra, de manera distinta. Por lo tanto, se aplicó polimorfismo con herencia, donde todas las clases hijas heredan el método **calcularPrecio** de la clase abstracta *Adicional* pero cada una la aplica de manera distinta.

```
Envio >>> calcularPrecio
|precio_act|
precio_act:=unPrecio.
(precio_act<5000) ifTrue:
        [precio_act:=precio_act+valor].
^ precio_act
```

```
Garantia >>> calcularPrecio
^unPrecio * (valor/100) + unPrecio
```

```
Cupon >>> calcularPrecio
^ unPrecio - (unPrecio*(valor/100)).
```

Lo mismo sucede con el atributo **Valor** de la clase, ya que tiene que haber distintas implementaciones para el mismo llamado desde **Compra**.

En un **Envío** el valor es un número positivo para adicionar directamente
En una **Garantía** el valor es un porcentaje a sumarle al precio
En un **Cupón** el valor es un porcentaje a descontar.

6. Excepciones

AlgoBayProductoNoPresente Teniendo el método **getProducto** para obtener un producto del mercado, si este no es encontrado, se lanzará esta excepción.

ProductoPrecioNegativoError Al agregar un producto al mercado, si este intenta ser agregado con un precio negativo, saltara esta excepción, ya que esto rompe el modelo planteado.

AdicionalPorcentajeInvalidoError Sea a la hora de crear un cupón de descuento, o de crear una garantía, si el valor no es un porcentaje valido, saltará esta excepción.

AdicionalEnvioNegativoError Siendo que en el modelo vigente siempre se cobra de más por un envío, y nunca se descuenta, si se intenta crear un envío negativo se lanza esta excepción.

7. Diagramas de secuencia

El primer diagrama de secuencia representa lo que pasa cuando se agregan productos al mercado, mostrando la interacción entre **AlgoBay** y **Producto**.

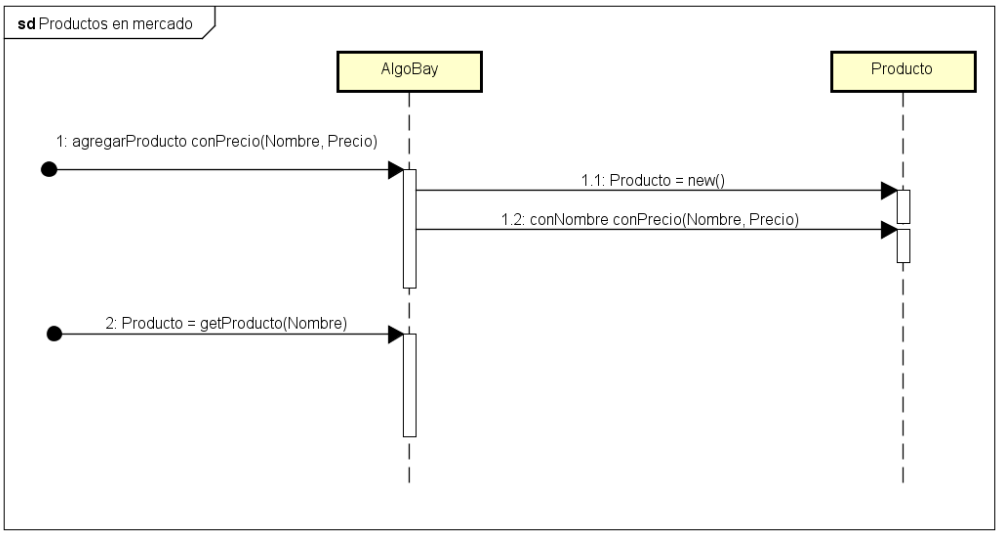

Figura 2: Agregar un producto al mercado.

En el segundo se ve como se crea una compra simple, sin adicionales, y se pide el precio total de esta.

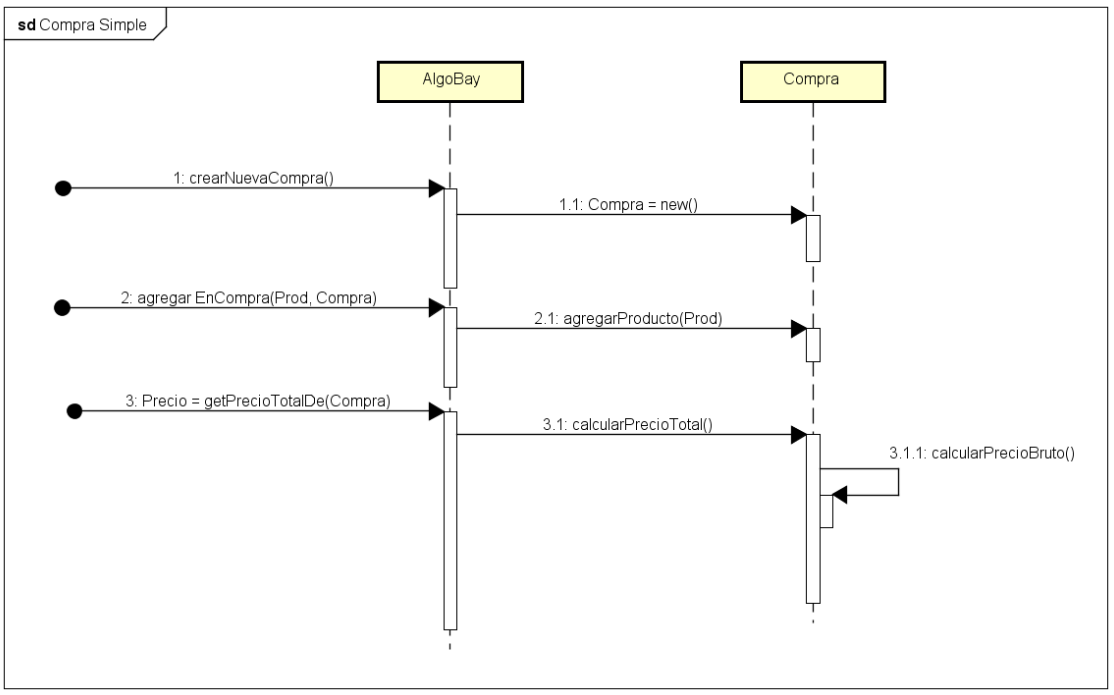

Figura 3: Crear una nueva compra, agregar productos y pedir su precio.

En el tercer diagrama se muestra como es la creación de una compra con adicionales, usando de ejemplo el adicional **Envío**.

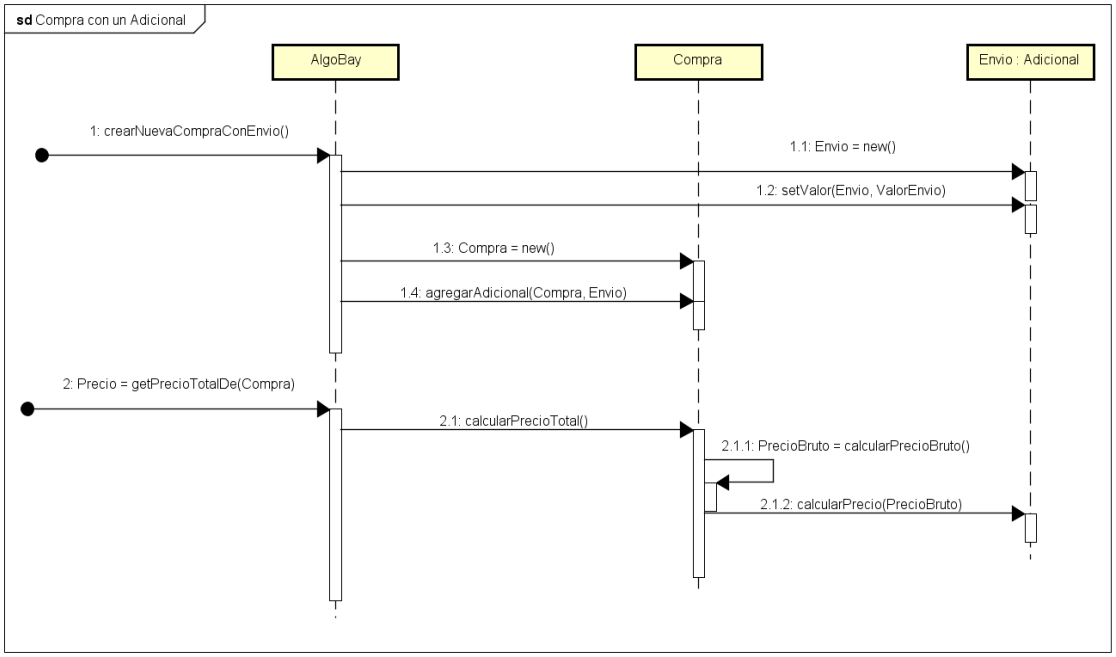

Figura 4: Crear una compra con un adicional.

El ultimo diagrama muestra lo sucedido al intentar agregar dos cupones a la misma compra.

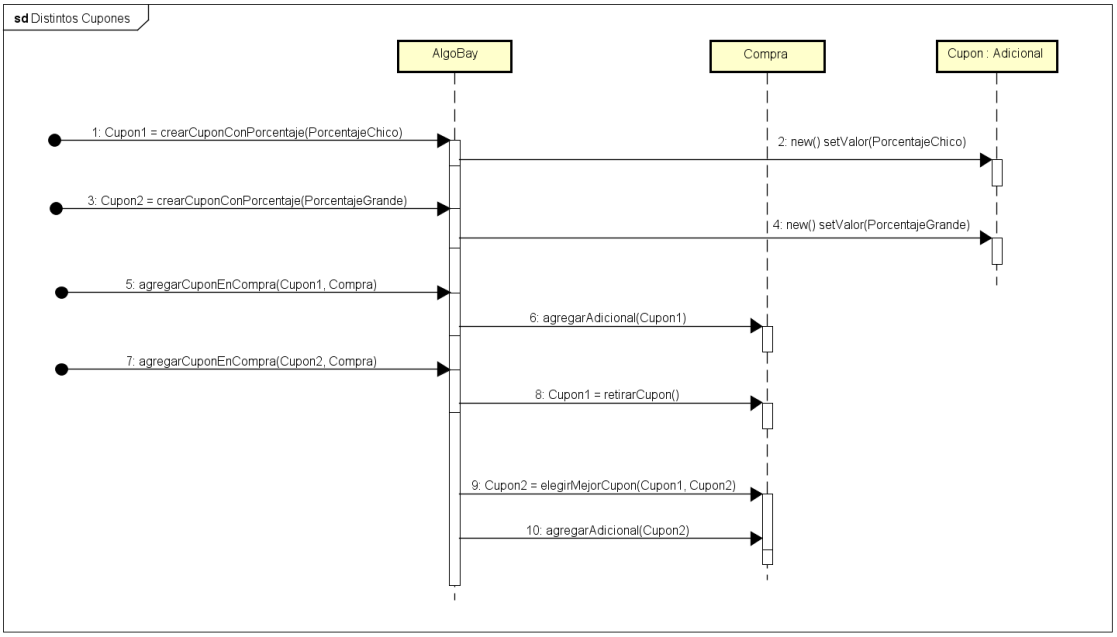

Figura 5: Manejo de cupones en simultaneo.