Trabajo Práctico 1 — Java

[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

El presente informe reúne la documentación de la solución del trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar una aplicación de un sistema de comercio electrónico en Java 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? Para cubrir tanto compras con envío como compras con garantías como compras con cupones, lo ideado es que la compra creada tenga distintos atributos que vayan manejando estos valores. De esta forma, una compra puede tener bajo el atributo **adicionables** un envío y una garantía definida como no tenerlos, mientras que bajo el atributo **cupón** contenga un cupón (este cupón es único ya que, bajo el contexto, siempre una compra mantiene el mayor cupón recibido). Pero, de no contenerlos (o en terminos técnicos también teniendolos con valor nulo), la compra debe comportarse con la forma esperada, sin crear nunca una compra de clase distinta, o haciendo operaciones fuera de lo normal. Una compra, con o sin atributos, debe tener su precio, y este debe ser calculable, no más que eso.

Por otro lado, y tambien respecto a la clase compra, es importante ver que una vez que el mercado le mande un mensaje a la compra con que productos contendrá, ya no es importante especificamente el nombre del producto, si no que solo es importante el precio de esta. La sumatoria de estos precios será lo que lleve a un precio en bruto de la compra, y sobre esto se aplicarán los atributos mencionados anteriormente.

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, por ejemplo, se tiene un registro de cuanto cuesta cada producto en vez de 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 precio de un bien que yo vaya a comprar tiene que ser situado en el objeto compra para poder finalizar la transacción. Luego, esta compra puede tener también modificadores, en el nombre de adicionables o cupones, que harán que varie el precio total de la compra.

Adicionable Actuando como interfaz de modificadores de precios de una compra, estos adicionables hacen que la compra disminuye o aumente su precio total. En el contexto dado, hay dos adicionables:

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

En el diagrama de clase se muestra como todo pasa por la clase AlgoBay, la moderadora del resto de los objetos. Crear una compra es en contexto de un mercado, crear un producto y agregarlo a la compra también. Por otro lado tenemos las dos clases que implementan de Adicionable, referenciados en la clase compra, según lo requiera AlgoBay a la hora de crearla, y la clase cupón que actua también como modificadora del precio de la compra.

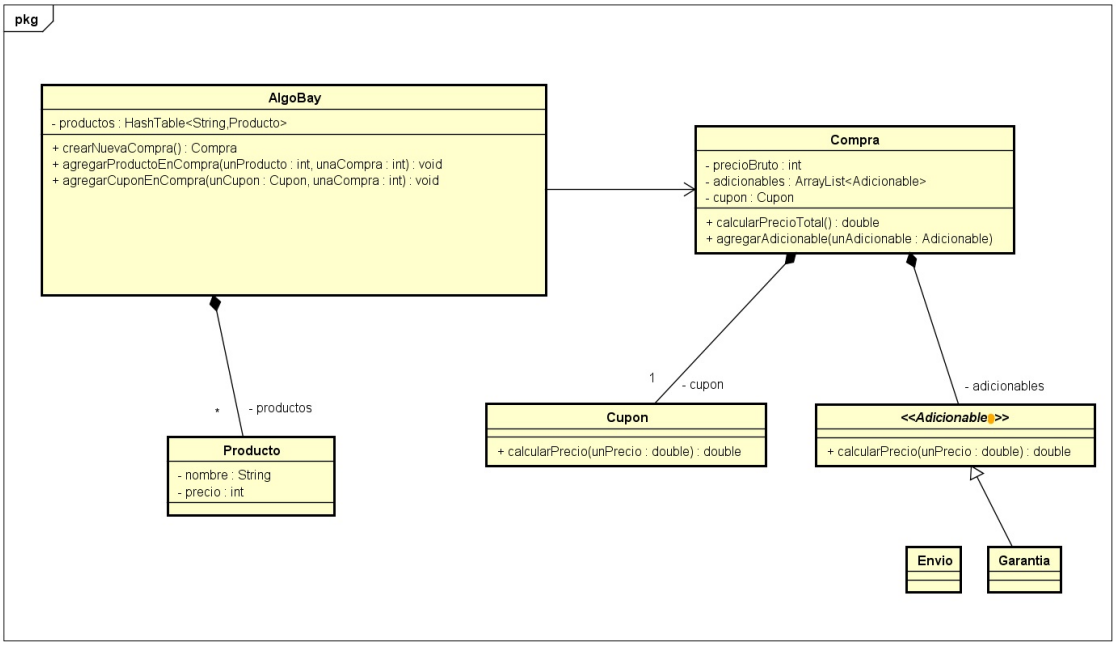

Figura 1: Diagrama conciso del programa.

5. Detalles de implementación

5.1. AlgoBay

AlgoBay, al actuar como moderadora, tiene una implementación bastante directa y sin desvíos, ya que en su mayoría se reduce a bajar el mensaje recibido. De recibir el deber de crear una compra, llama al constructor de compra. De recibir el deber de crear una compra con algun adicionable, llama no solo al constructor pero también a un metodo de esta que se encargue del labor. De esta forma se puede ver como, en sus métodos, delega la responsabilidad. Por ejemplo, al momento de crear un cupón y de agregarlo a una compra:

```
public Cupon crearCuponConPorcentaje(double unPorcentaje){
    return new Cupon(unPorcentaje);
}

public void agregarCuponEnCompra(Cupon unCupon, Compra unaCompra){
    unaCompra.agregarCupon(unCupon);
}
```

5.2. Compra

El mayor problema en la clase **Compra** fue la idea de dejar de lado el nombre del producto y que solo se encargue de sus precios, de esta forma haciendo que el precio de la compra, antes de aplicarle modificadores, sea una simple sumatoria. Una vez introducidos los modificadores, la decisión tomada fue la de armar una lista (**ArrayList**) de adicionables que, de ser el metodo `agregarAdicionable` invocado, irá llenando una lista de los adicionables de la compra, mientras que por otro lado se tenga una referencia al **único** cupón de esta. Teniendo esto, se puede calcular el precio total solamente invocando el método **calcularPrecio** sobre todos sus modificadores.

```
public double calcularPrecioTotal(){
double precioTotal = precioBruto;
for (Adicionable adicionable : adicionables)
precioTotal = adicionable.calcularPrecio(precioTotal);
if(cupon != null)
precioTotal = cupon.calcularPrecio(precioTotal);
return precioTotal;
}
```

5.3. Modificadores

Lo complejo de los modificadores fue el hacer que cada uno pueda responder al mismo mensaje, el aplicado por compra, de manera distinta. Por lo tanto, se aplicó polimorfismo con herencia. De esta forma, **envío** y **garantía** implementan la interfaz adicionable mientras que el cupón actua independientemente y las tres contienen el mismo método, **calcularPrecio**, cada una aplicandolo de forma distinta.

```
Envio >>> calcularPrecio
public double calcularPrecio(double unPrecio){
if(unPrecio < 5000)
return unPrecio+valor;
return unPrecio;
}

Garantia >>> calcularPrecio
public double calcularPrecio(double unPrecio){
return unPrecio*( (100+valor) /100);
}

Cupon >>> calcularPrecio
public double calcularPrecio(double unPrecio){
   return unPrecio - (unPrecio * (valor/100));
}
```

Por otro lado fue complejo el comportamiento al momento de recibir más de un cupón para la misma compra, ya que estas solo deben tener uno, el mayor. Para esto, lo que se decidió fue que la clase de mercado simplemente llame al método de agregar un cupón a una compra y que la clase compra sea la que elija el mejor cupon para ella, que en este caso es el mayor. Para determinar si el cupon es el mayor, se llama al mensaje de cupón que lidia con eso: **esMayorQue**.

```
public boolean esMayorQue(Cupon otroCupon){
   return valor >= otroCupon.valor;
}
```

6. Excepciones

ProductoNoPresenteError 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.

CuponPorcentajeInvalidoError 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.

7. Diagramas de secuencia

El primer diagrama de secuencia representa lo que pasa cuando se agregan productos al mercado y a su vez estos se agregan a una compra con envío, mostrando la interacción entre **AlgoBay** y **Compra** con **envío** (quien implementa de **«Adicionable»**). Luego, a esta compra se le pide el precio total.

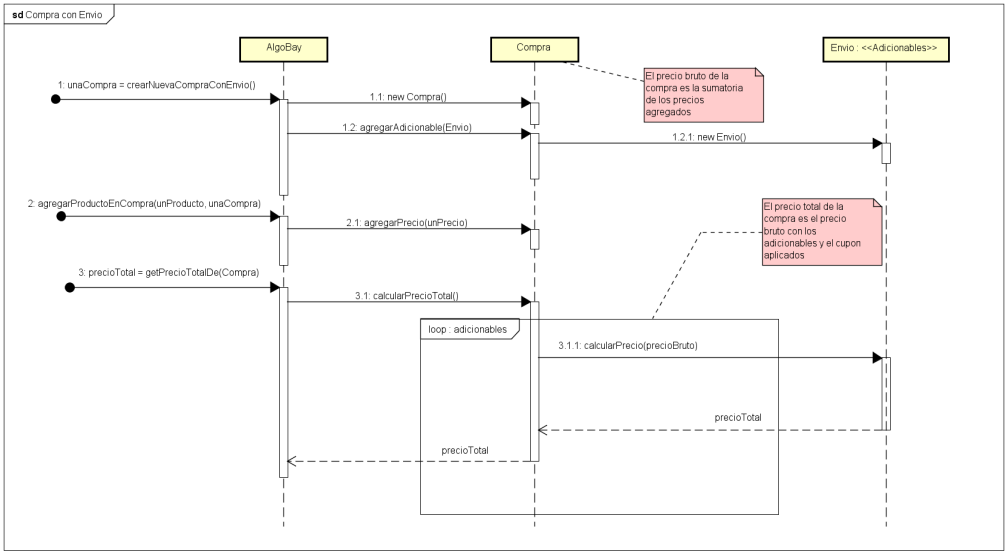

Figura 2: Comrpa con envío.

En el segundo diagrama de secuencia se ve como se comporta una compra a la que se le agreguen tanto un adicionable como un cupón

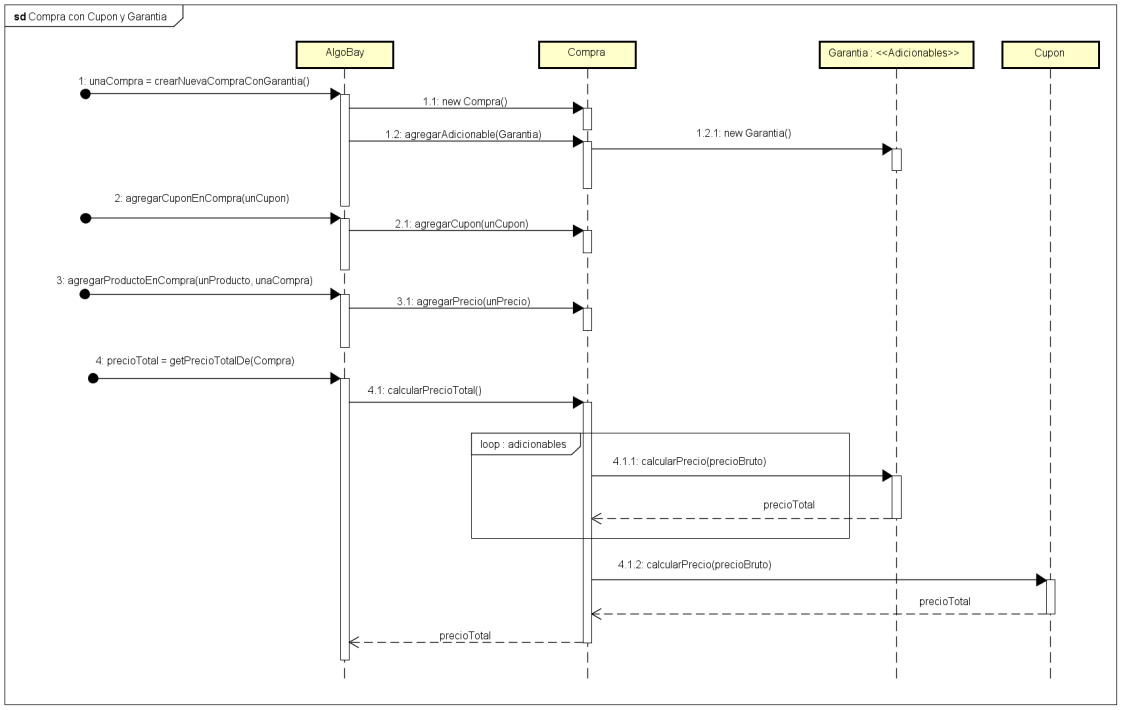

Figura 3: Compra con garantía y cupón.

El último diagrama muestra como es la interacción entre **AlgoBay**, **Compra** y **Cupón** cuando se intenta de agregar más de un cupón a una compra, ya que el proceso de por sí es interesante por la delegación de responsabilidades

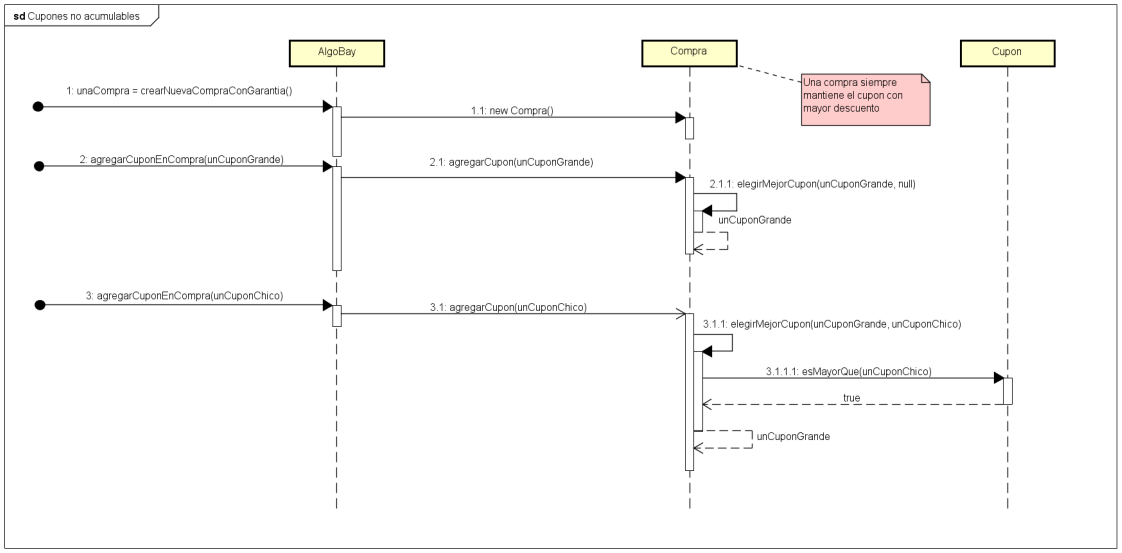

Figura 4: Cupones no acumulables.