Informe de “Gestión de Billetes”

Grupo 24-08

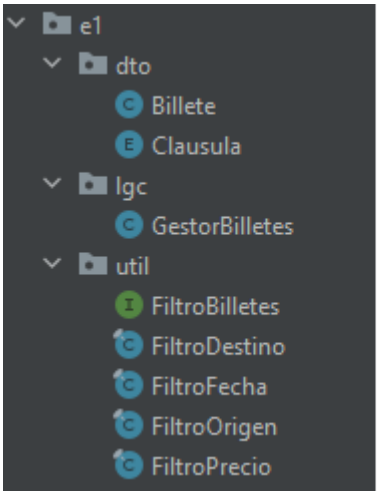
Bosco Suárez-LLanos Outeiriño

Samanta K. Machuca Montalvo

1. Principios de diseño

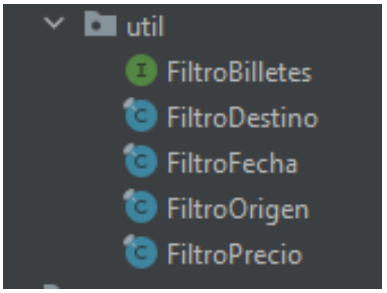
- Principio de Responsabilidad Unica

  Todas las clases cumplen con este principio ya que todas tienen alta cohesión entre ellas y tienen un solo objetivo.



- Principio Abierto-Cerrado

  FiltroBilletes está declarada como una interfaz sellada y solo permite a los cuatro tipos de filtros definidos



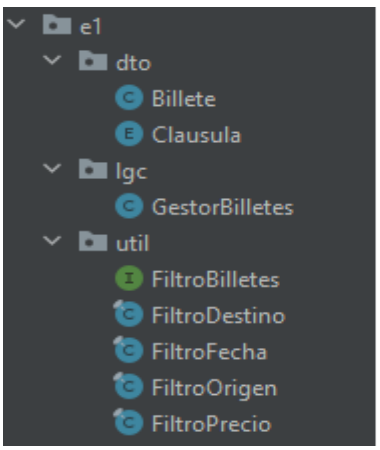
- Principio de Sustitución de Liskov

```
public List<Billete> filtrar(Clausula clausula, List<FiltroBilletes> filtros)
```

En la clase GestorBilletes existe el principio de sustitución de Liskov porque cumple el principio de subcontratación. Recibe por parámetro una lista de FiltroBilletes, la cual al ser una interfaz acepta que le pasen todas sus distintas implementaciones.

- Principio de Inversión de la Dependencia

Prácticamente en todo el ejercicio se hace uso de este principio en la definición de los List, así como el uso de los filtros gracias a la interfaz FiltroBilletes.

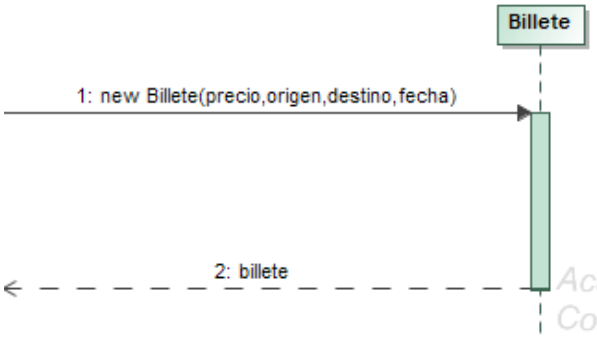



```
private List<Billete> billetes = new ArrayList<>();
```

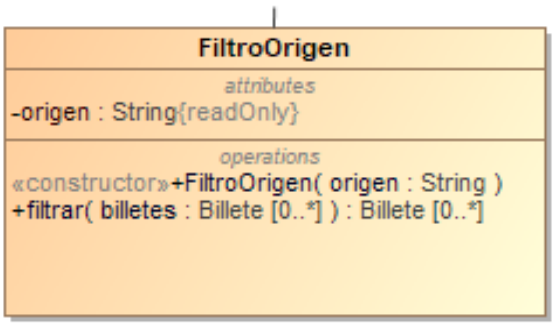
2. Patrones de diseño

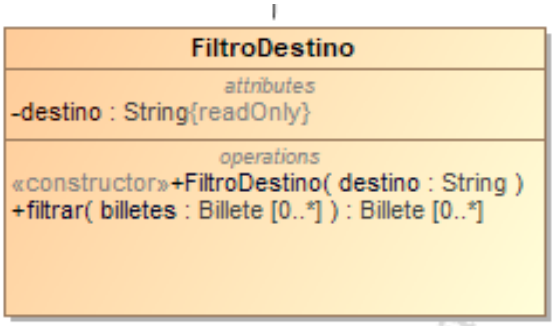
- **Principio “Favorece la inmutabilidad” (Patron Inmutable)**

Utilizamos el patrón inmutable en las clases Billete y los tipos de filtros ya que toda la información contenida en dichas clases es proporcionada en el momento de la creación declarándose en los atributos y no puede modificarse durante el tiempo de vida del objeto.



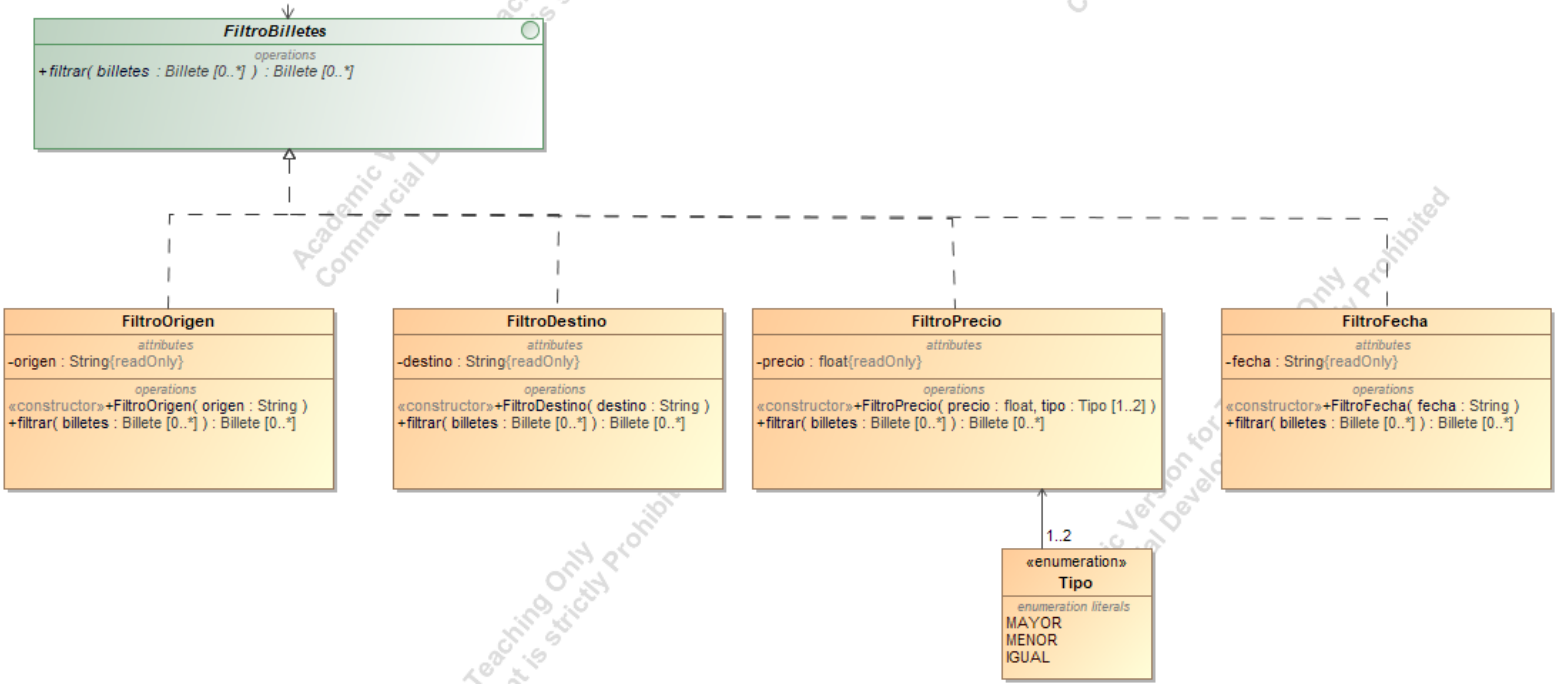
Atributos privados y finales, solo métodos getters u operaciones.

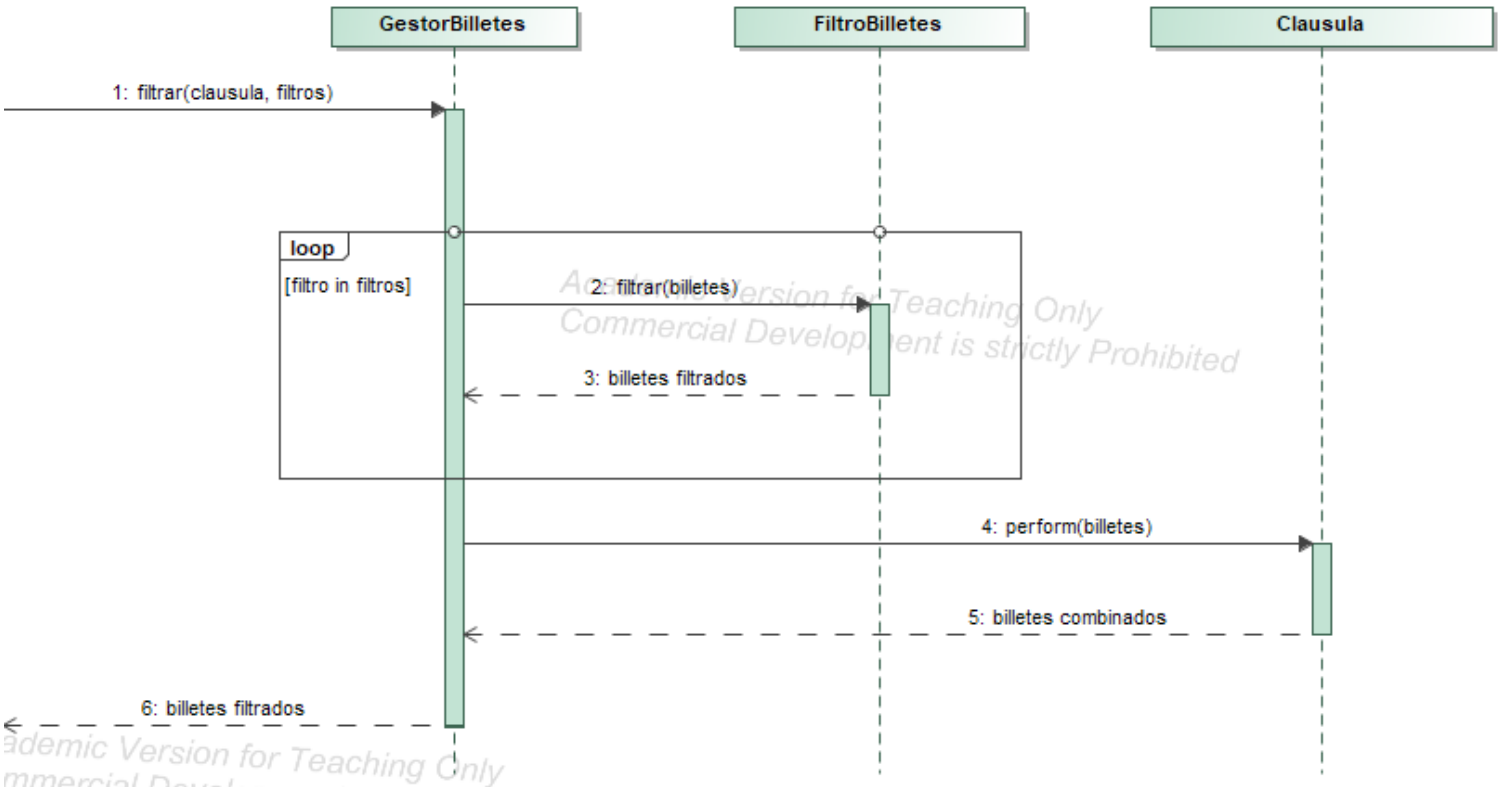




- **<u>Principio “Encapsula lo que varía” (Patrón Estrategia)</u>**

Los filtros son un buen ejemplo del uso del patrón estrategia, ya que las interfaces fueron creadas precisamente para ese propósito, y poder ampliarlas en un futuro.



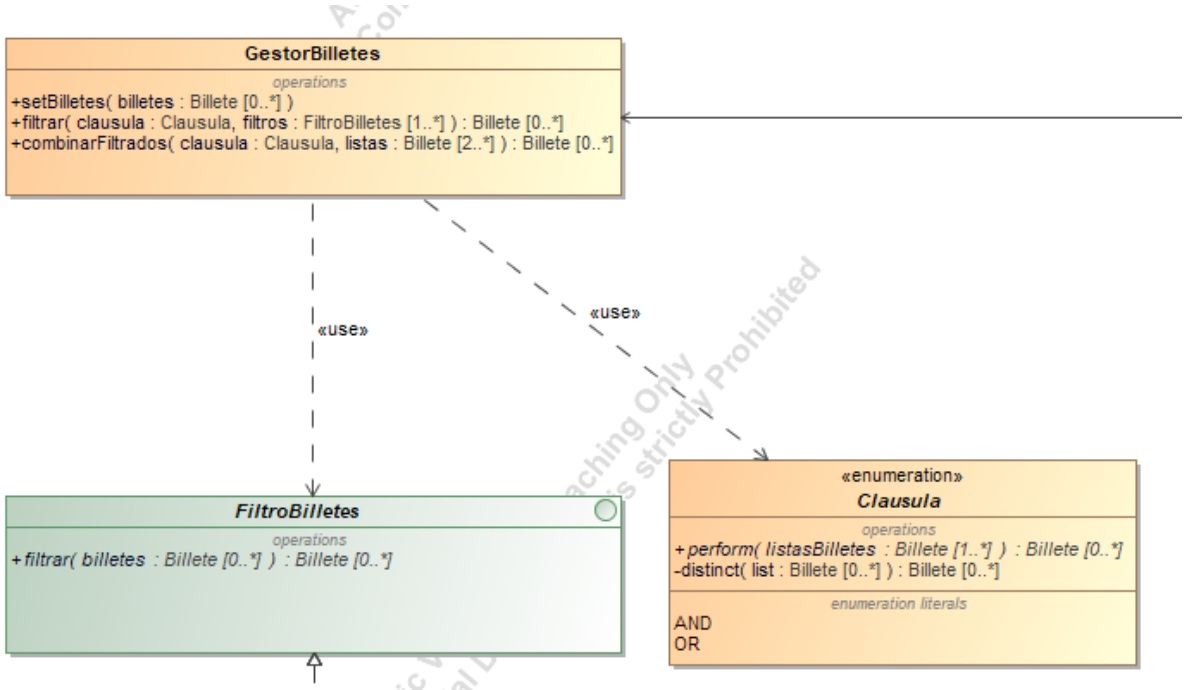


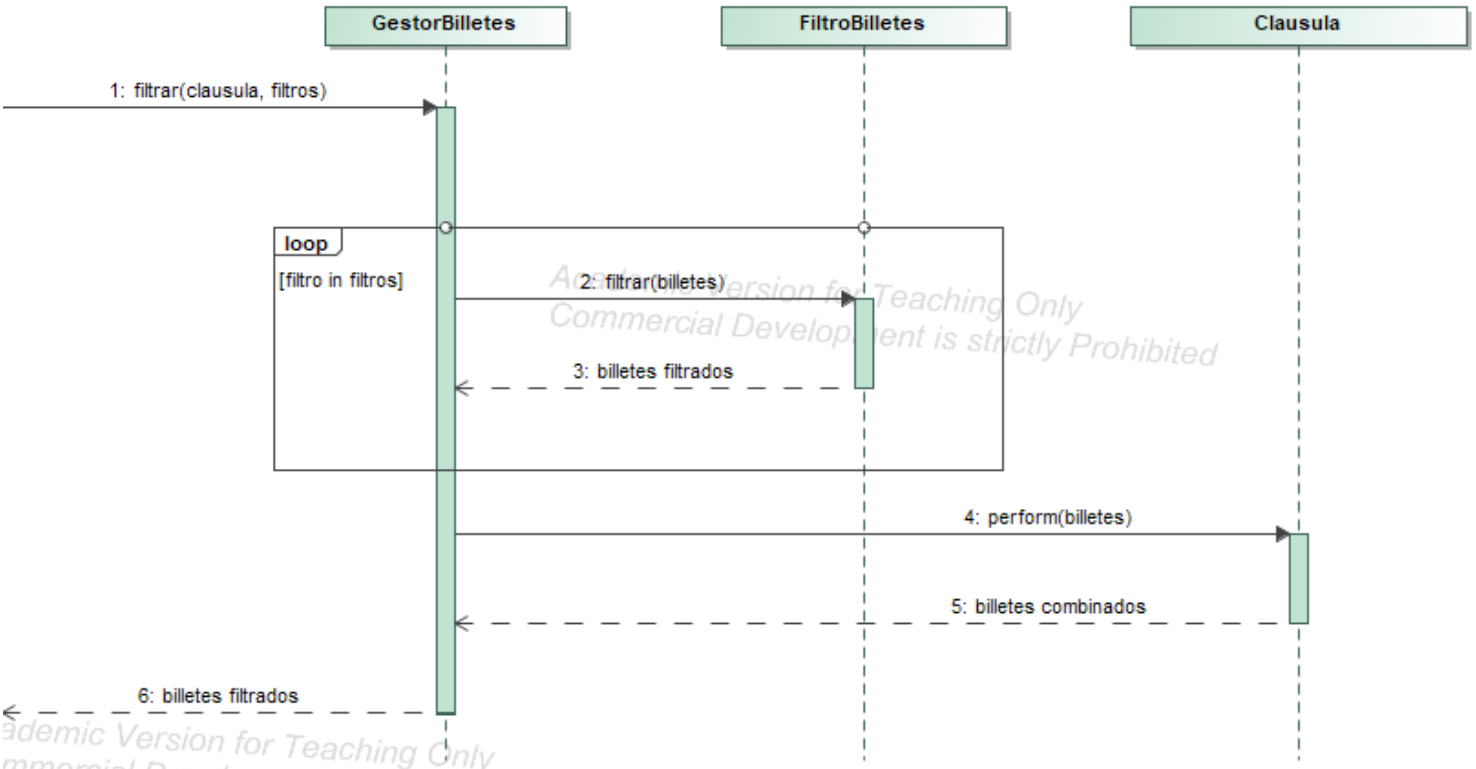
- **<u>Principio de “Bajo Acoplamiento”</u>**

Todas las clases cumplen con este principio ya que todas desconocen información de las demás clases delegando en ellas operaciones secundarias.

ej:

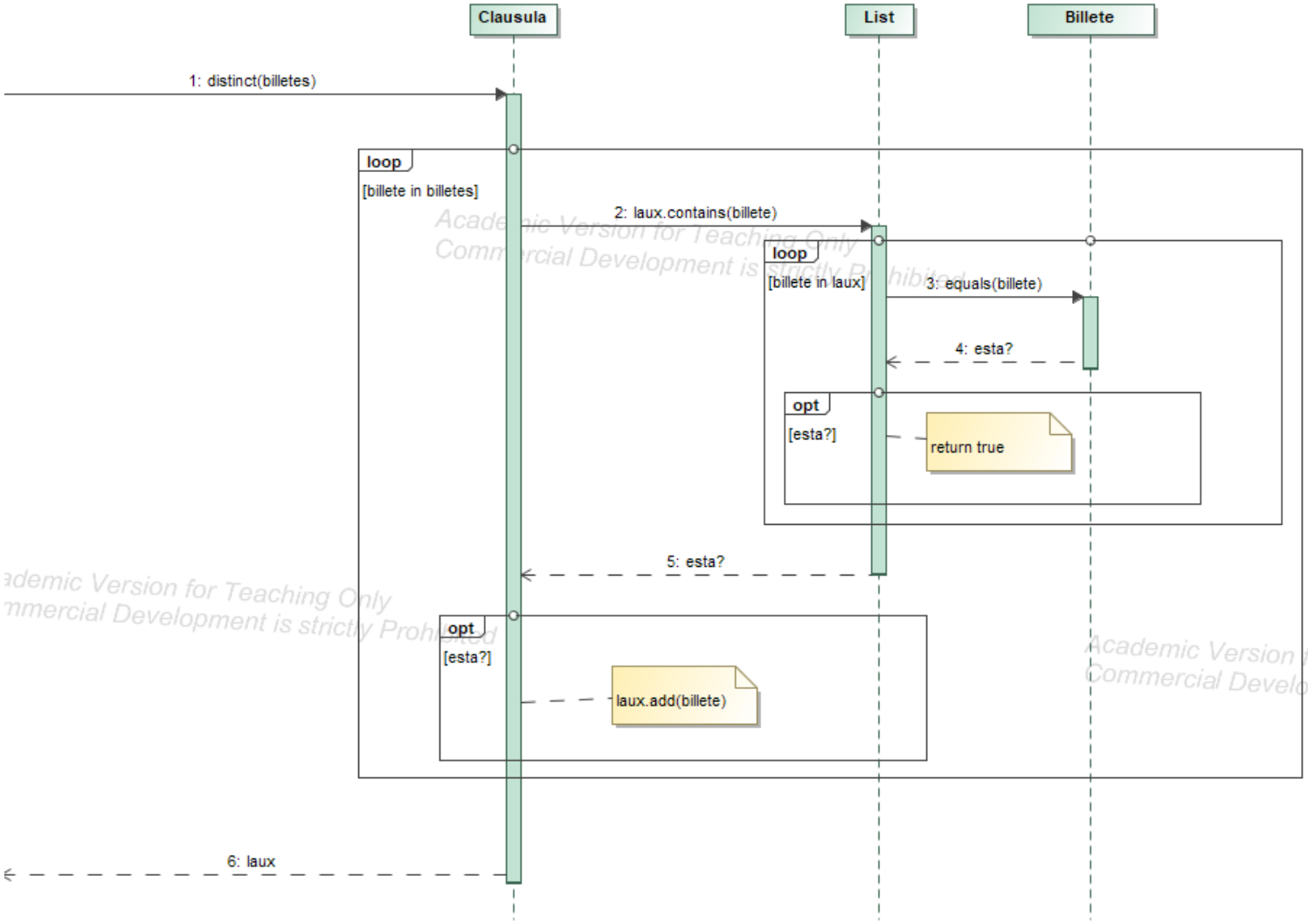
```
return clausula.perform(aux);
```





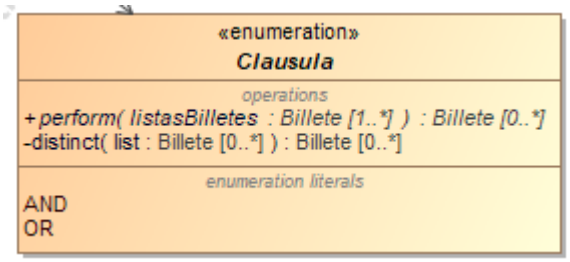
- **<u>Principio “Tell, Don’t Ask”</u>**

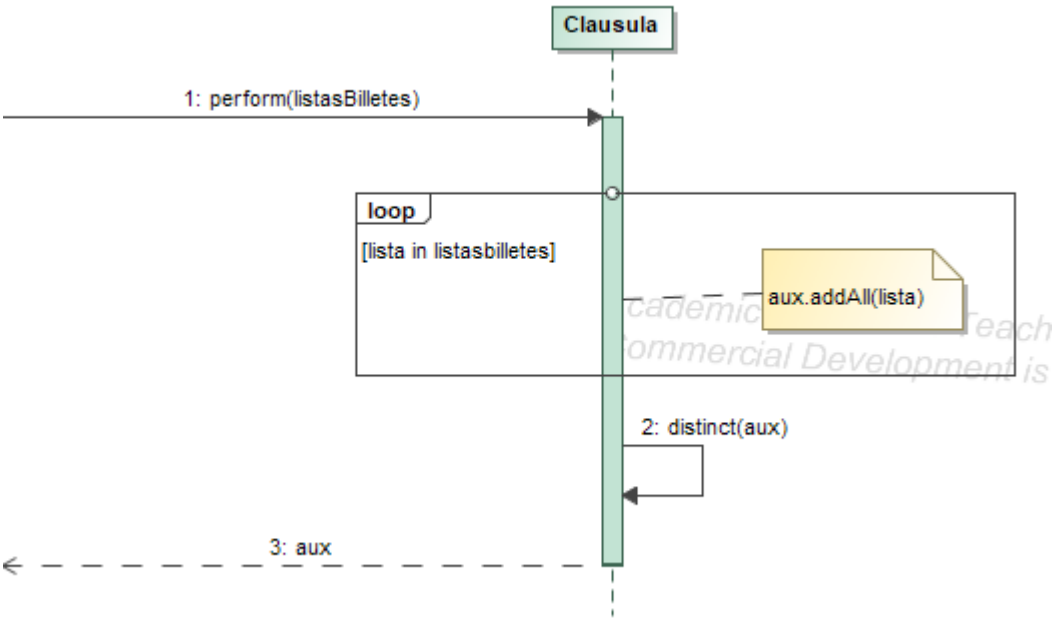
Definimos el método equals en la clase Billete para que se encargue de hacer las comparaciones.



- **Principio DRY “Don’t Repeat Yourself”**

Este principio lo usamos por ejemplo en la clase Cláusula al definir el método distinct para su uso dentro de la clase.





- **Principio YAGNI “You Aren’t Gonna Need It”**

Todas las clases dentro de este ejercicio cumplen un propósito y son usadas, no se ha realizado sobreingeniería, de modo que no hay clases que no se necesitan.

DIAGRAMA DE CLASES COMPLETO