# Desarrollo de Software Basado en Modelos

# Información general

1- Nombre de la herramienta, empresa que la desarrolló, versión y año.

Url: https://plantuml.com/
Github: https://github.com/plantuml/plantuml
Herramienta : PlantUML
Año de Inicio del proyecto: Abril 22 del 2009
Versión actual: 1.2021.4
Licencia : GPL2 pero existen versiones en otras licencias. (Apache, MIT, Eclipse)

Es una herramienta que permite graficar diagramas uml, manejando el layout automáticamente. Lo cual tiene sus pros y sus contras, que detallaré al final del informe.

2- ¿Qué diagramas permite realizar la herramienta? ¿Sobre qué versión de UML?

Trabaja sobre uml2.

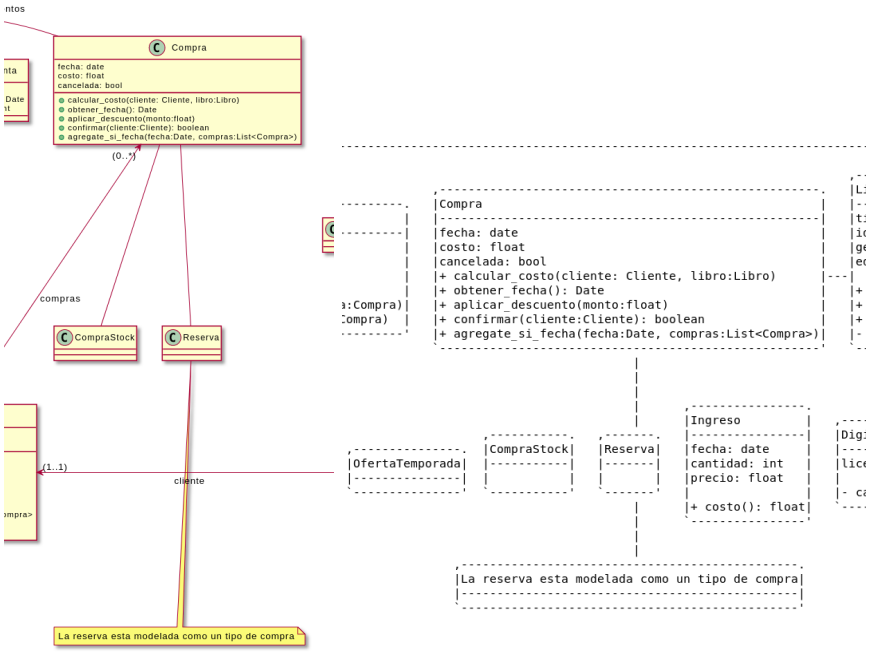
Diagramas de secuencia
Diagramas de caso de uso
Diagramas de clase
Diagramas de Actividad
Diagramas de Componentes
Diagramas de Estado
Diagramas de Objetos

Y algunos otros diagramas que no son parte de uml.
En el sitio principal se tiene una barra superior con todos los diagramas posibles de generar.


3- ¿La herramienta elegida permite generar una documentación completa del modelo? ¿En qué formatos? Ejemplifique que información se encuentra en la documentación generada, y que información no.

No, no permite generar una documentación. La herramienta funciona generando un gráfico desde una descripción textual del modelo, permite prototipar modelos rápidamente para analizar visualmente.

Los formatos de exportación del gráfico son varios (png, svg, txt). El txt genera en ascii una representación del gráfico, lo cual puede ser útil en algunos casos de diagramas no complejos. Ej:




4- ¿La herramienta permite generar código? ¿En qué lenguaje/s?

No permite generar código por si misma. Pero existen proyectos que lo realizan a partir de la descripción textual del modelo de plantuml.
https://github.com/jupe/puml2code
Permite generar código en los siguientes lenguajes.

- CoffeeScript (coffeescript)
- C# (csharp)
- C++ (cpp)
- ECMAScript5 (ecmascript5)
- ECMAScript6 (ecmascript6) [default]
- Java (java)
- PHP (php)
- python (python)
- Ruby (ruby)
- TypeScript (typescript)

5- Si permite generar código, ¿permite hacer ingeniería inversa?

La herramienta no permite ingeniería inversa por si misma. Pero existen otros proyectos que generan a partir del código de un lenguaje específico, el texto para ser interpretado por plantuml.
https://pypi.org/project/py2puml/
Con esto se podría tomar codigo en python y convertirlo a diagramas de plantuml para analizarlo de forma gráfica.

6- Documente alguna facilidad que haya encontrado en la herramienta, como la posibilidad de definir un glosario de términos para el proyecto, control de cambios, etc.

No es una herramienta de modelado por lo que no tiene grandes capacidades mas que la mera traducción de un modelo en texto a gráfico. Pero esto lo hace interesante para algunas aplicaciones específicas. Por ejemplo es muy simple versionar los diagramas usando git, ya que son meros archivos de texto.
Es muy simple prototipar rápido diagramas no complejos ya que se encarga automáticamente del layout.

7- ¿Permite importar / exportar a otros formatos? ¿A cuáles?
Solo permite exportar los diagramas a :


# Casos de Uso

1- ¿La herramienta elegida provee alguna facilidad para escribir las conversaciones del caso de uso? Muestre cómo las pudo escribir.

No permite, hay que escribirlos aparte en otra herramienta.
En mi caso utilicé la plantilla brindada por la cátedra usando google docs para editarla y manejar las versiones.
El problema que veo es que es complejo sincronizar los cambios, si cambiamos la conversación se debe realizar la identificación manualmente para posteriormente modificarlo en el texto del modelo de la herramienta.

2- En UML, para indicar que un caso de uso está incluido en otro se subraya el nombre del caso de uso en la conversación. ¿Pudo hacer eso con la herramienta?

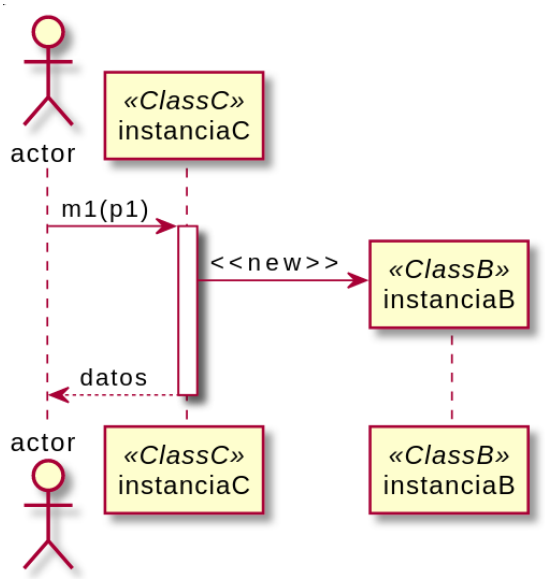
No, se hace aparte dentro de la plantilla del documento usando la herramienta elegida para documentarlo. En mi caso fue google docs usando la plantilla provista por la cátedra.

# Información de los diagramas de secuencia

1- ¿Como indica en su herramienta que un objeto es una instancia de una clase en particular?

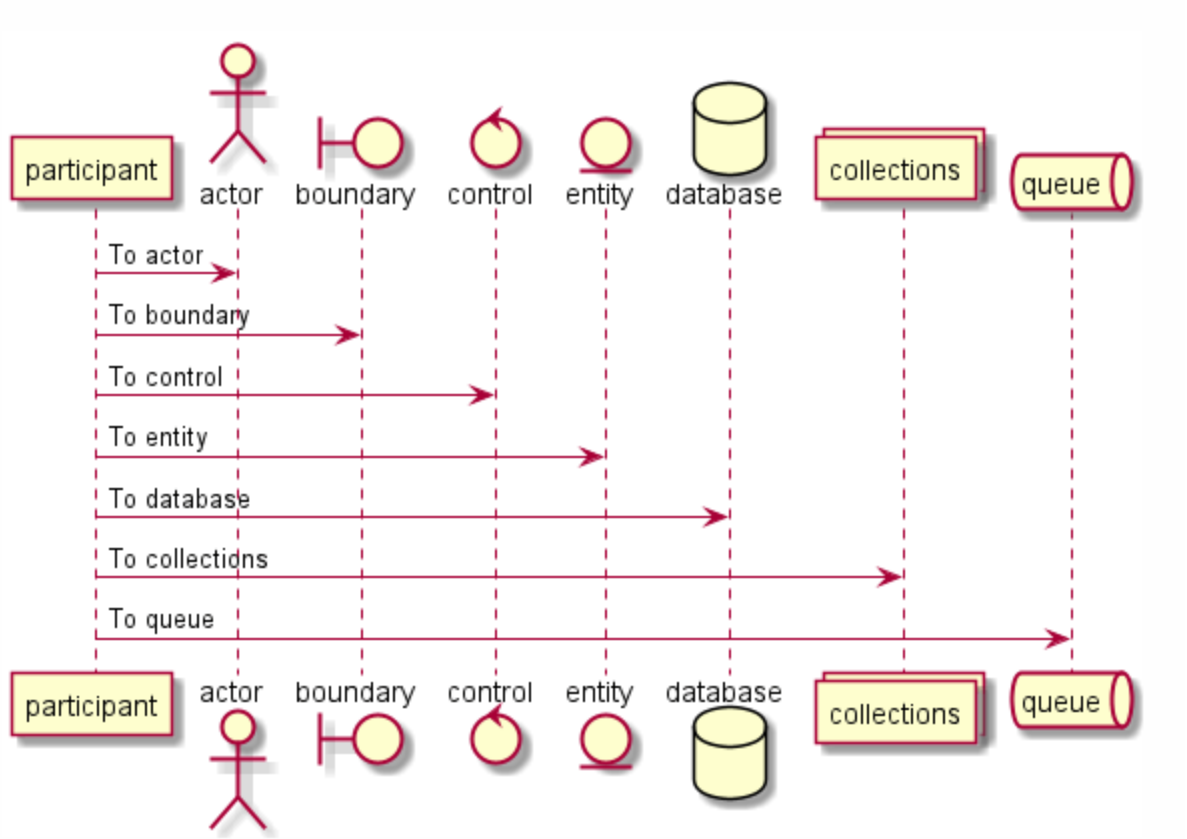
Usando la sintaxis << >> en el código de generación del gráfico.
Ej:

```
@startuml

actor actor
participant instanciaC as c <<ClassC>>
participant instanciaB as b <<ClassB>>

actor -> c : m1(p1)
activate c
    c -> b ** : <<new>>
    return datos
deactivate c

@enduml
```




También permite distintos tipos de gráficos de los participantes del diagrama.

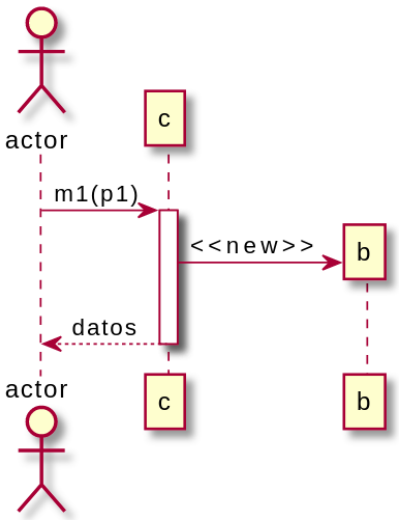
participant
actor
boundary
control
entity
database
collections
queue




2- En UML, si un mensaje es de creación, la caja del objeto se desplaza hasta la recepción del mensaje de creación. ¿Permite hacerlo su herramienta? ¿Cómo indica que un mensaje es de creación?

Si, permite indicarlo en el envío del mensaje usando **

```
@startuml

actor actor

actor -> c : m1(p1)
activate c
    c -> b ** : <<new>>
    return datos
deactivate c

@enduml
```



3- Algunas herramientas, a medida que se agregan mensajes en el diagrama de secuencia, agregan los métodos en el diagrama de clases. ¿Su herramienta lo hace? ¿Cómo permite relacionar un mensaje con el método especificado en la clase?

No, no existe relación entre cada uno de los diagramas. Se podría llegar a generar mediante plugins que analicen el código que genera el diagrama. Pero no existe nada actualmente.

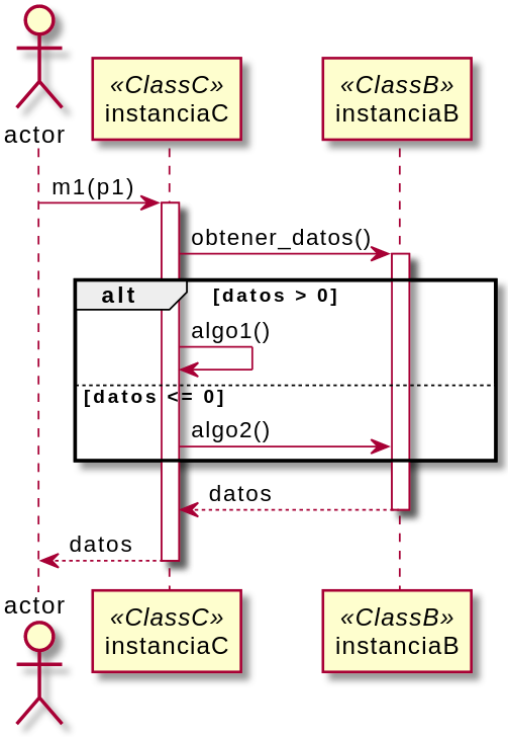
4- ¿Permite especificar frames alternativos, o loops?

Si, permite agrupar los mensajes usando las siguientes palabras clave.

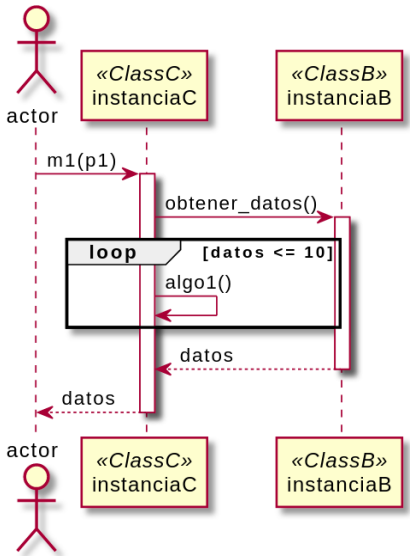
- alt/else
- opt
- loop
- par
- break
- critical
- group, followed by a text to be displayed

Para el caso de los loops y alternativas por ejemplo:

```
@startuml

actor actor
participant instanciaC as c <<ClassC>>
participant instanciaB as b <<ClassB>>

actor -> c : m1(p1)
activate c
    c -> b : obtener_datos()
    activate b
        alt datos > 0
            c -> c : algo1()
        else datos <= 0
            c -> b : algo2()
        end
        return datos
    deactivate b

    actor <-- c : datos
deactivate c

@enduml
```



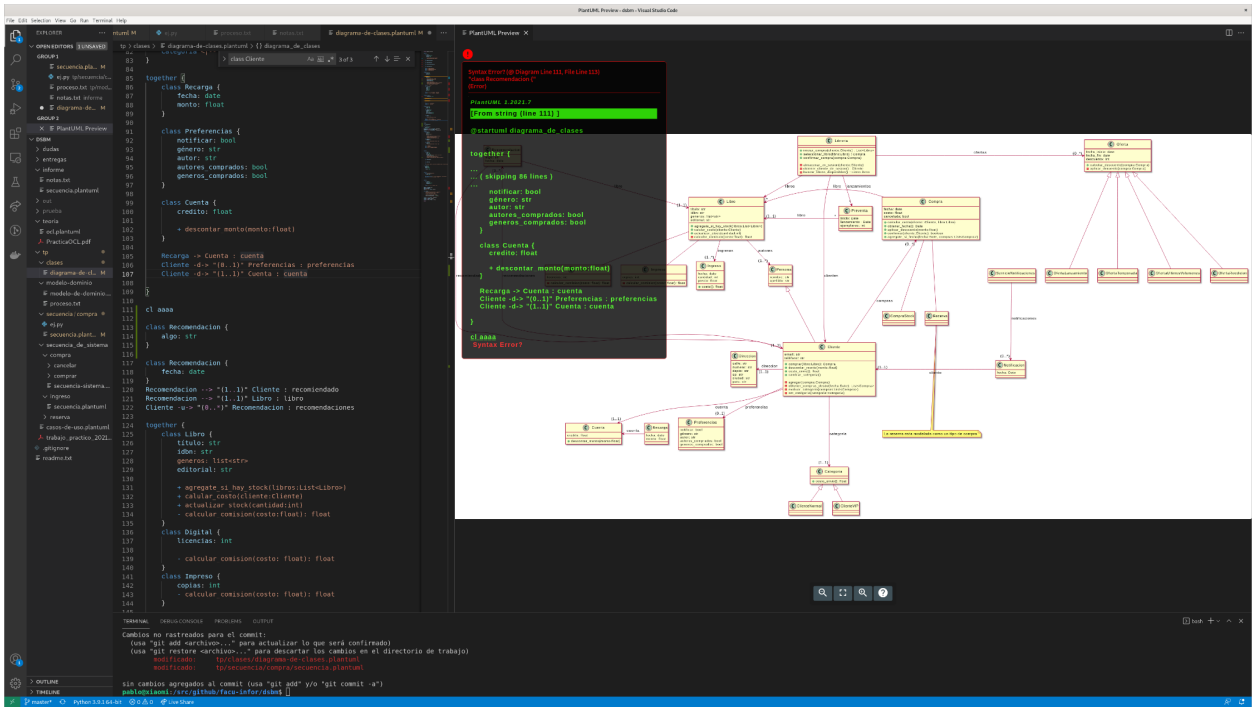
```
@startuml

actor actor
participant instanciaC as c <<ClassC>>
participant instanciaB as b <<ClassB>>

actor -> c : m1(p1)
activate c
    c -> b : obtener_datos()
    activate b
        loop datos <= 10
            c -> c : algo1()
        end
        return datos
    deactivate b

    actor <-- c : datos
deactivate c

@enduml
```



# Diagramas de clases

1- La herramienta elegida permite escribir modelos que no estén bien formados, como por ejemplo, tener dos clases con el mismo nombre, o dos atributos con el mismo nombre en la misma clase ¿Advierte mediante algún warning?

No chequea el modelo, por lo que permite generar modelos inválidos. Pero como la generación del gráfico se basa en el procesamiento de una sintaxis, este mismo procedimiento puede detectar ciertos errores.
En casos de error de sintaxis de la definición el sistema informa con un error en la línea del error.



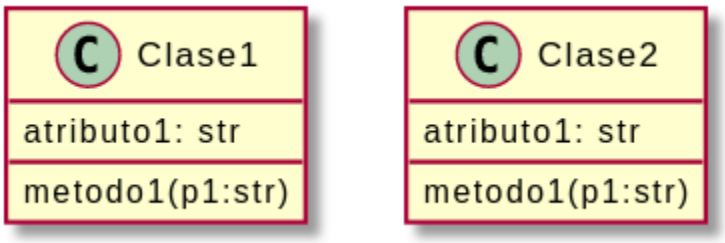
Pero permite definir múltiples clases con el mismo nombre, que tiene el efecto de combinar las definiciones en el gráfico final y no mostrar dos clases distintas con el mismo nombre.

```
class Recomendacion {
    algo: str
}

class Recomendacion {
    fecha: date
}
```

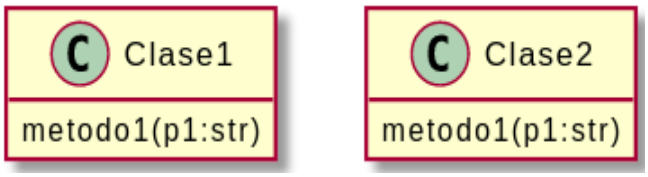

2- ¿Permite ocultar ciertos elementos, por ejemplo, ocultar en alguna vista ciertos atributos? Y ciertas relaciones?

Si, mediante definiciones de hide/show
Se pueden ocultar atributos y métodos selectivamente, por clase o para todas, mediante hide members, hide fields, hide methods
Ej:

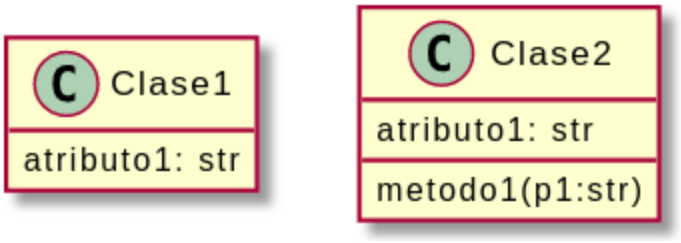
```
class Clase1 {
    atributo1: str
    metodo1(p1:str)
}

class Clase2 {
    atributo1: str
    metodo1(p1:str)
}
```




```
class Clase1 {
    atributo1: str
    metodo1(p1:str)
}

class Clase2 {
    atributo1: str
    metodo1(p1:str)
}

hide fields
```

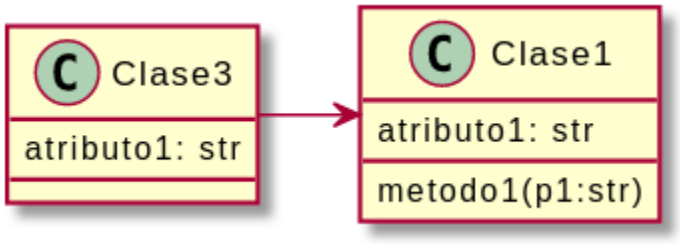


```
class Clase1 {
    atributo1: str
    metodo1(p1:str)
}

class Clase2 {
    atributo1: str
    metodo1(p1:str)
}

hide Clase1 methods
```




Inclusive permite ocultar o mostrar las clases que no tengan asociaciones definidas..

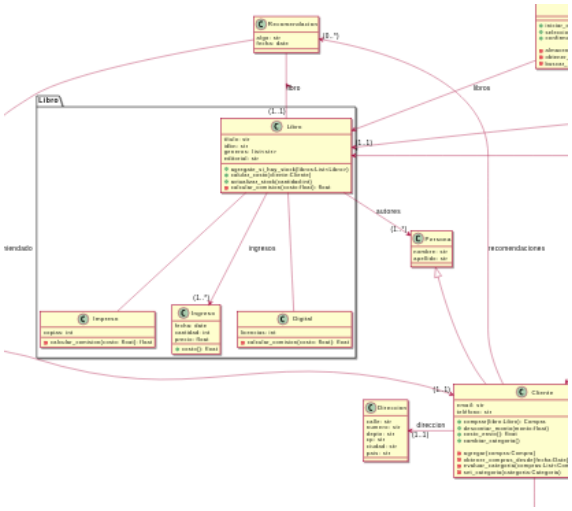
```
class Clase1 {
    atributo1: str
    metodo1(p1:str)
}

class Clase2 {
    atributo1: str
    metodo1(p1:str)
}

class Clase3 {
    atributo1: str
}

Clase3 -> Clase1

hide @unlinked
```



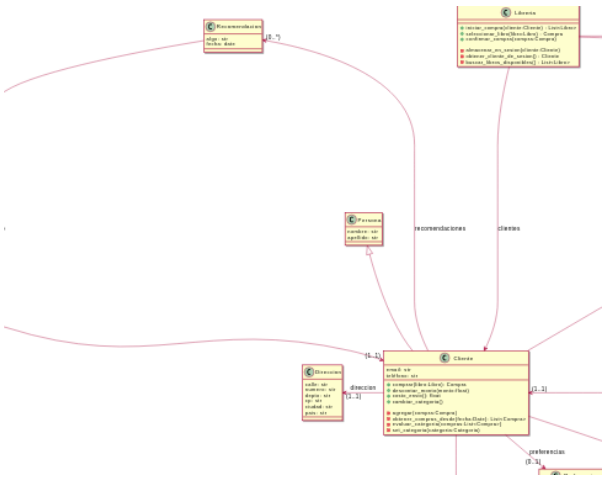

También se puede ocultar paquetes para cuando se incluyen archivos de otros diagramas y solo se quieren ocultar partes seleccionadas para proporcionar vistas del mismo.

```
@startuml informe

!include diagrama-de-clases.plantuml

hide Libro

@enduml
```

Antes del hide:



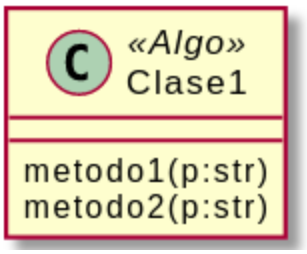
Posterior al hide:

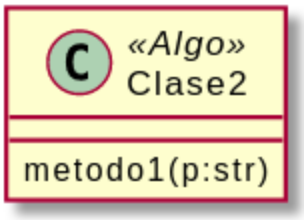


3- ¿Permite agrupar los atributos o métodos con algún estereotipo? Pruebe agregar el estereotipo <<creation>> a dos o más métodos y documente si los agrupa.

No se pueden asignar estereotipos a métodos.
Solo a clases mediante << >>

```
class Clase1 <<Algo>> {
    metodo1(p:str)
    metodo2(p:str)
}

class Clase2 <<Algo>> {
    metodo1(p:str)
}
```




No agrupa por estereotipo.

4- Algunas herramientas, a medida que se agregan las clases se muestran en un árbol junto con los demás elementos del modelo. ¿Ocurre esto con su herramienta? ¿Qué pasa con las asociaciones? ¿Cómo se muestran?

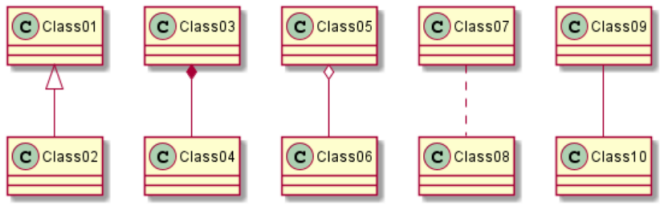
No se muestra en formato árbol ya que la herramienta traduce una sintaxis simplificada de un modelo a un diagrama.
El editor que usemos para generar ese archivo de sintaxis puede ser cualquiera. En mi caso estoy usando Visual Studio Code. el cual no tiene un plugin para interpretar la sintaxis y mostrarla visualmente en algún otro formato.
Las asociaciones son lineas textuales entre dos entidades, se pueden definir de distintas formas.

| Type | Symbol | Drawing |
|---|---|---|
| Extension | `<\|--` |  |
| Composition | `*--` |  |
| Aggregation | `o--` |  |

Un ejemplo con todos los casos posibles:

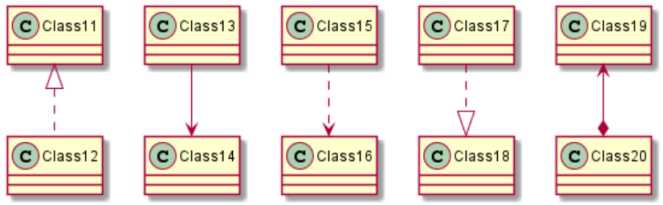
Edit online

```
@startuml
Class01 <|-- Class02
Class03 *-- Class04
Class05 o-- Class06
Class07 .. Class08
Class09 -- Class10
@enduml
```



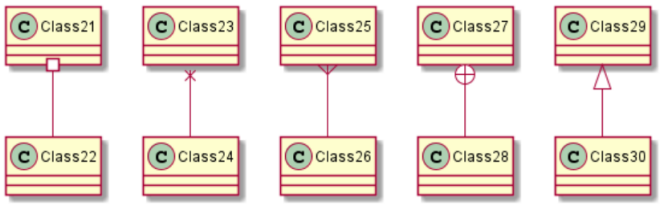

Edit online

```
@startuml
Class11 <|.. Class12
Class13 --> Class14
Class15 ..> Class16
Class17 ..|> Class18
Class19 <--* Class20
@enduml
```




Edit online

```
@startuml
Class21 #-- Class22
Class23 x-- Class24
Class25 }-- Class26
Class27 +-- Class28
Class29 ^-- Class30
@enduml
```

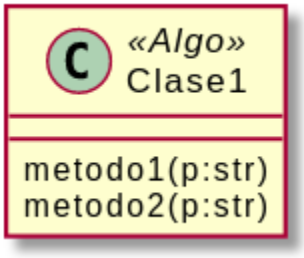


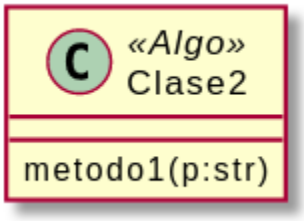

5- En toda herramienta hay dos tipos de borrado: borrado de la vista y borrado del modelo. Documente cómo hace para borrar una clase y una asociación con los dos tipos de borrado.

Para eliminar de la vista se usa “hide”
Para eliminar del modelo se usa “remove”

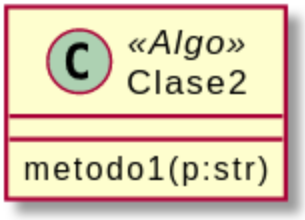
Ej de eliminación de una clase de la vista

```
@startuml informe


class Clase1 <<Algo>> {
    metodo1(p:str)
    metodo2(p:str)
}

class Clase2 <<Algo>> {
    metodo1(p:str)
}


@enduml       You, 2 wee
```



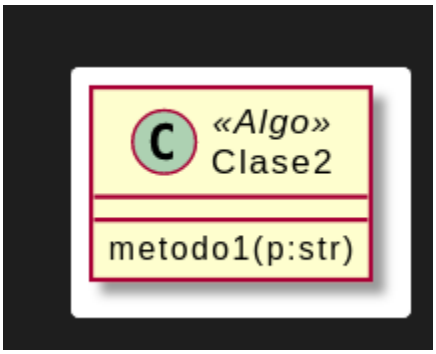



```
@startuml informe


class Clase1 <<Algo>> {
    metodo1(p:str)
    metodo2(p:str)
}

class Clase2 <<Algo>> {
    metodo1(p:str)
}

hide Clase1

@enduml
```



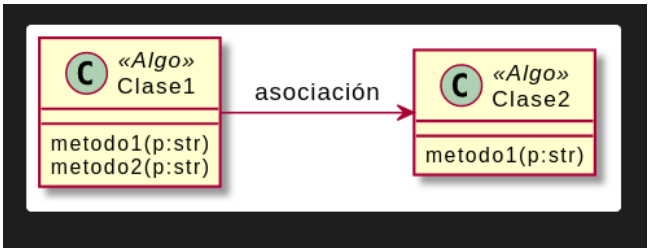

Ej de eliminación de la clase del modelo.

```
@startuml informe


class Clase1 <<Algo>> {
    metodo1(p:str)
    metodo2(p:str)
}

class Clase2 <<Algo>> {
    metodo1(p:str)
}

remove Clase1

@enduml
```




No se puede eliminar o ocultar asociaciones. Solo las clases de destino/origen lo que tiene el efecto de ocultar la asociación.

```
@startuml informe


class Clase1 <<Algo>> {
    metodo1(p:str)
    metodo2(p:str)
}

class Clase2 <<Algo>> {
    metodo1(p:str)
}

Clase1 -> Clase2 : asociación
      You, seconds ago • Uncom

@enduml
```

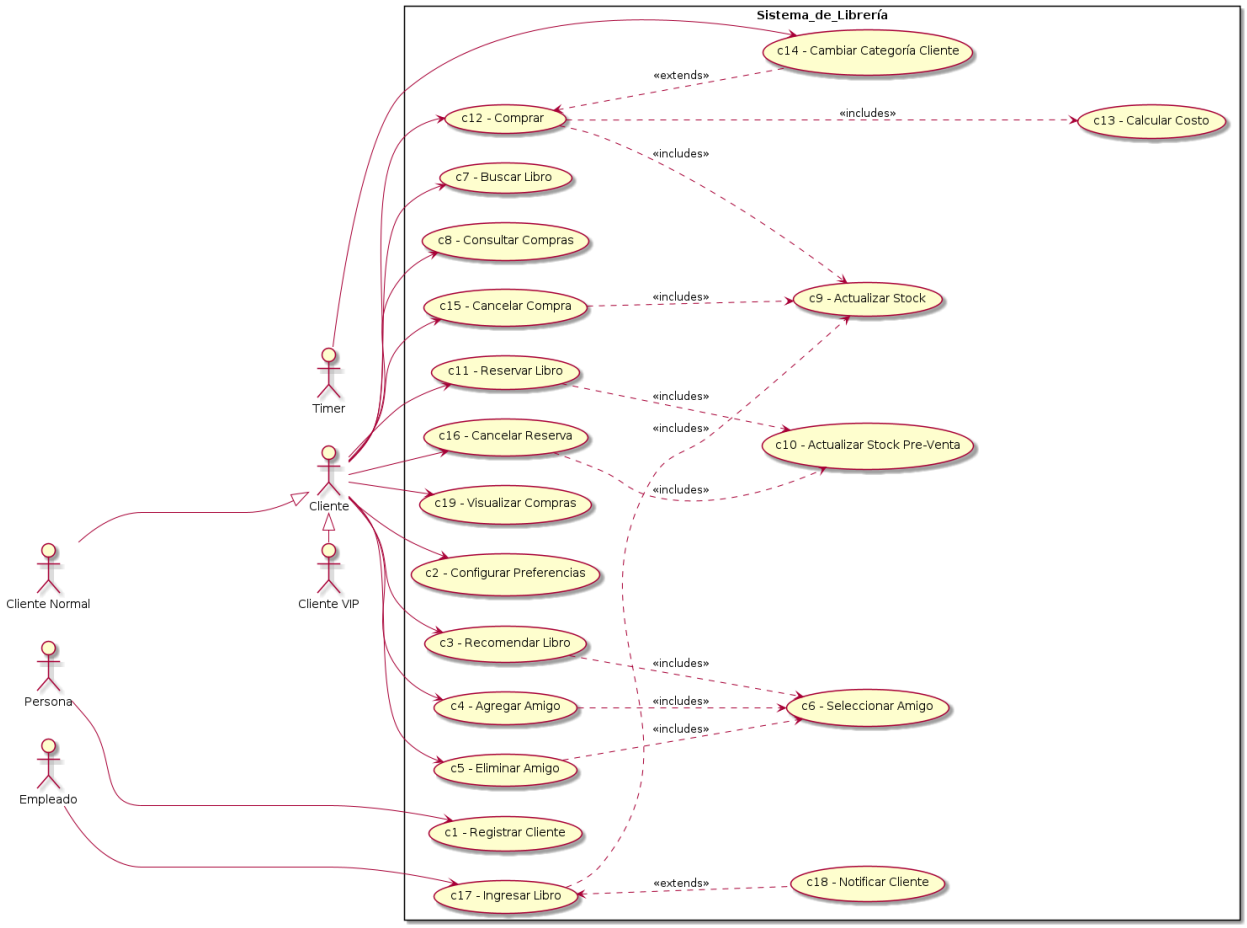



Ej:

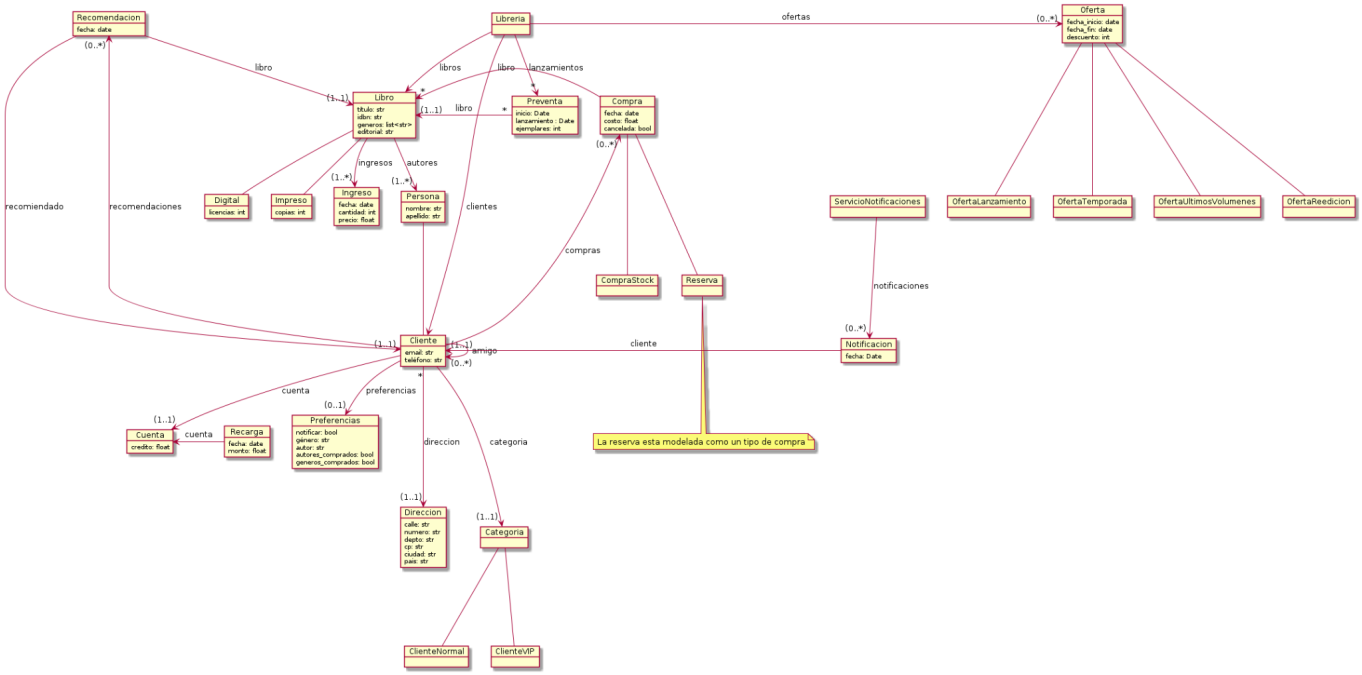
```
@startuml informe


class Clase1 <<Algo>> {
    metodo1(p:str)
    metodo2(p:str)
}

class Clase2 <<Algo>> {
    metodo1(p:str)
}

Clase1 -> Clase2 : asociación
remove Clase2       You, second

@enduml
```

# Diagramas generados para el TP.

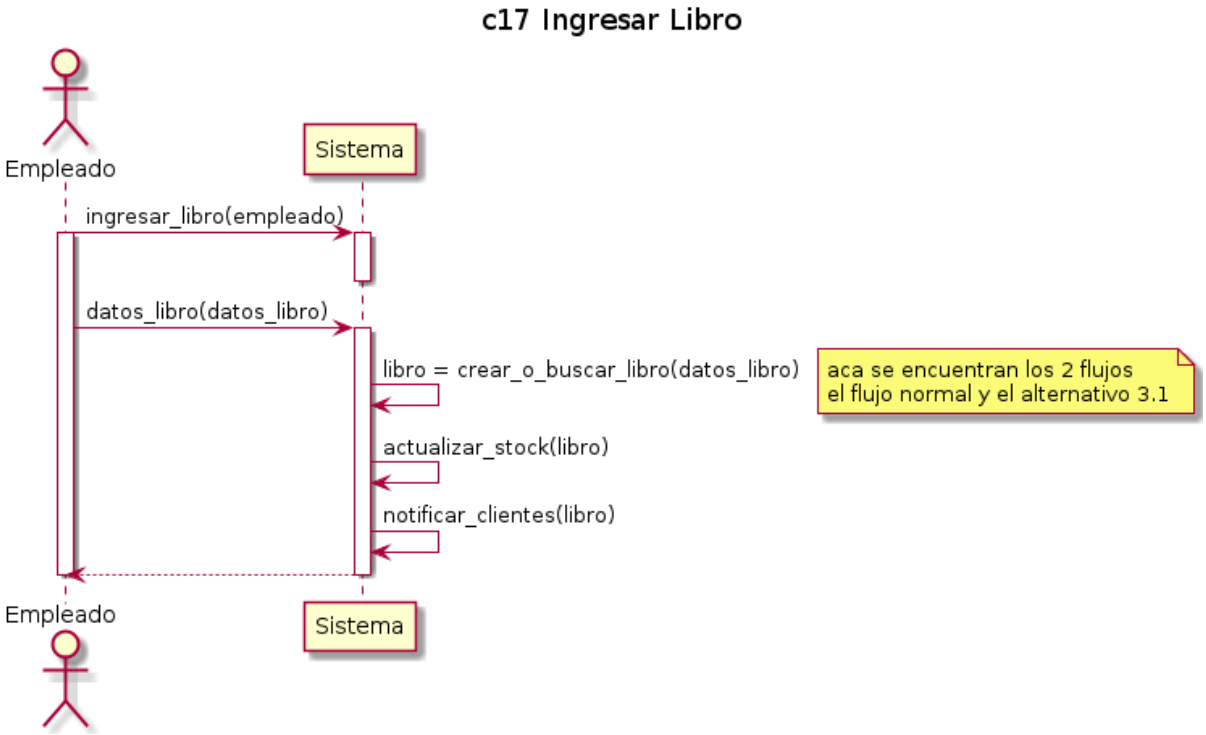
## Casos de Uso

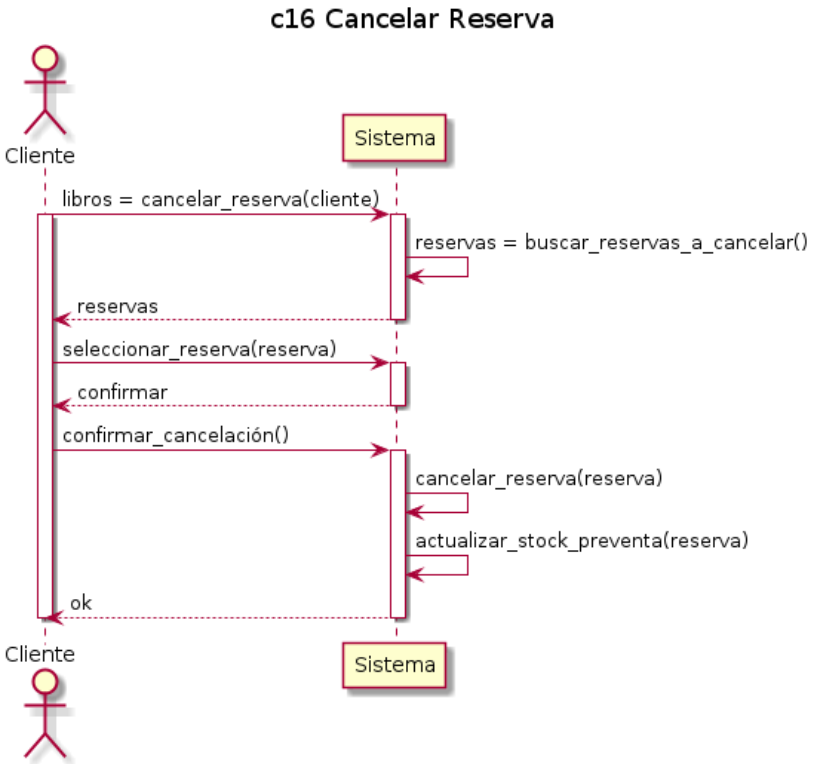


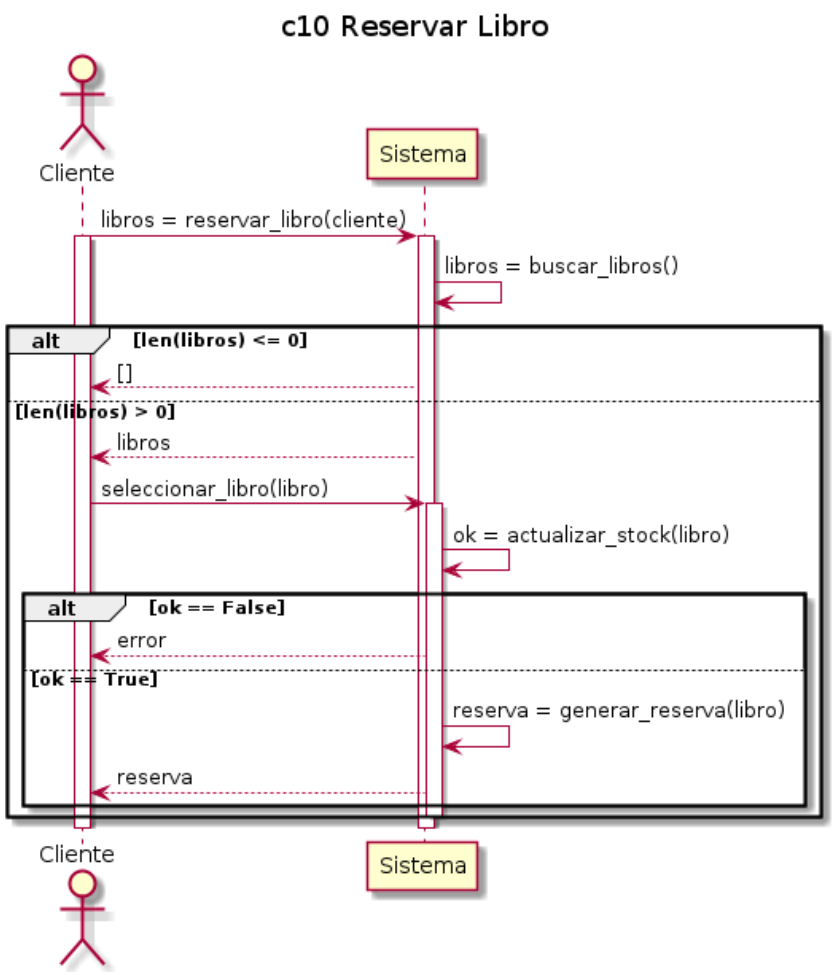
# Modelo de Dominio

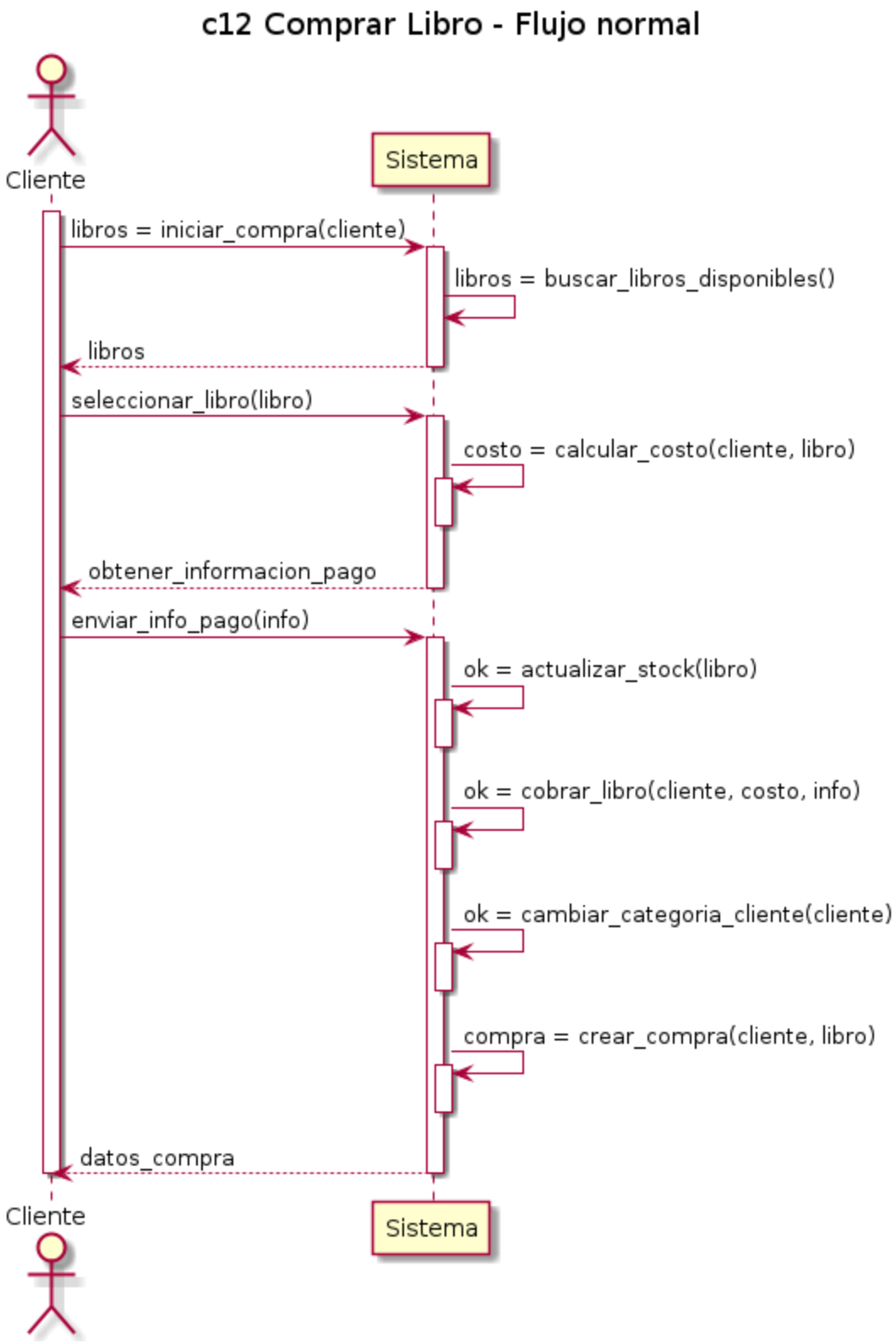


# Secuencia de Sistema

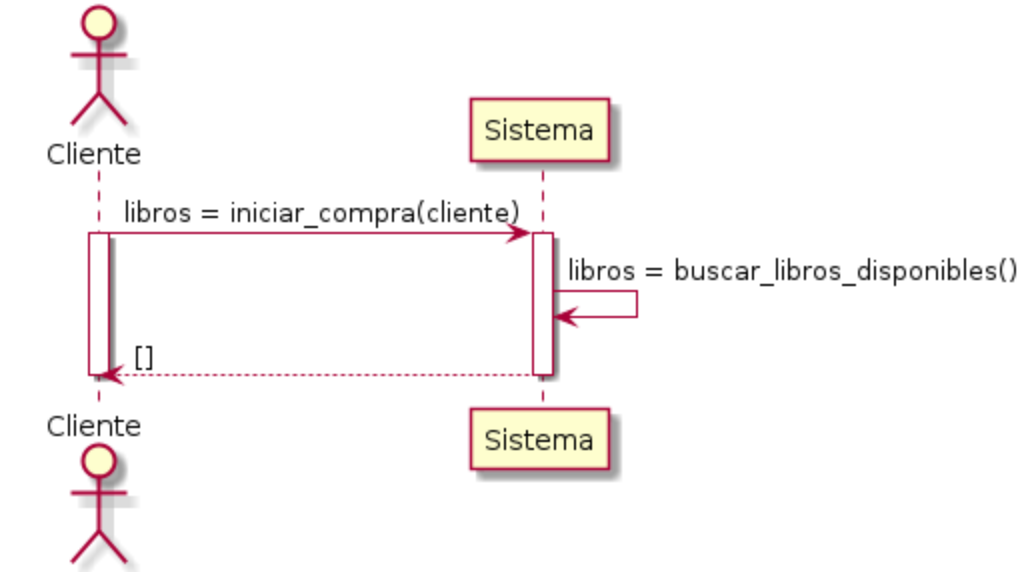


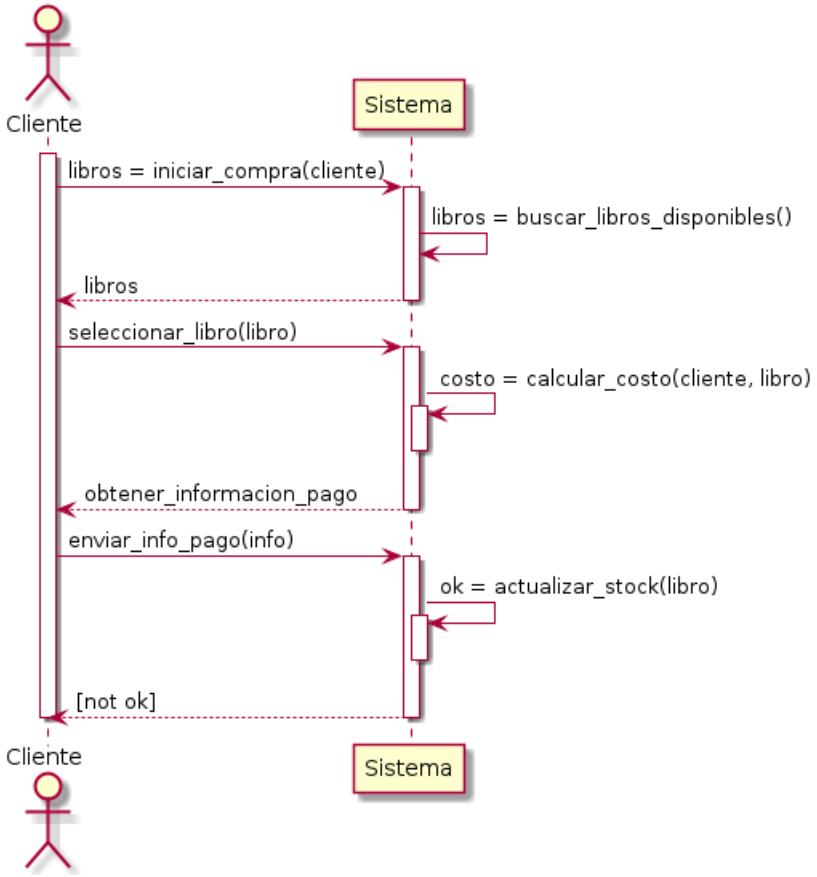

c10 Reservar Libro
Cliente
Sistema
libros = reservar_libro(cliente)
libros = buscar_libros()
alt
[len(libros) <= 0]
[]
[len(libros) > 0]
libros
seleccionar_libro(libro)
ok = actualizar_stock(libro)
alt
[ok == False]
error
[ok == True]
reserva = generar_reserva(libro)
reserva
Cliente
Sistema


c12 Comprar Libro - Flujo normal
Cliente
Sistema
libros = iniciar_compra(cliente)
libros = buscar_libros_disponibles()
libros
seleccionar_libro(libro)
costo = calcular_costo(cliente, libro)
obtener_informacion_pago
enviar_info_pago(info)
ok = actualizar_stock(libro)
ok = cobrar_libro(cliente, costo, info)
ok = cambiar_categoria_cliente(cliente)
compra = crear_compra(cliente, libro)
datos_compra
Cliente
Sistema

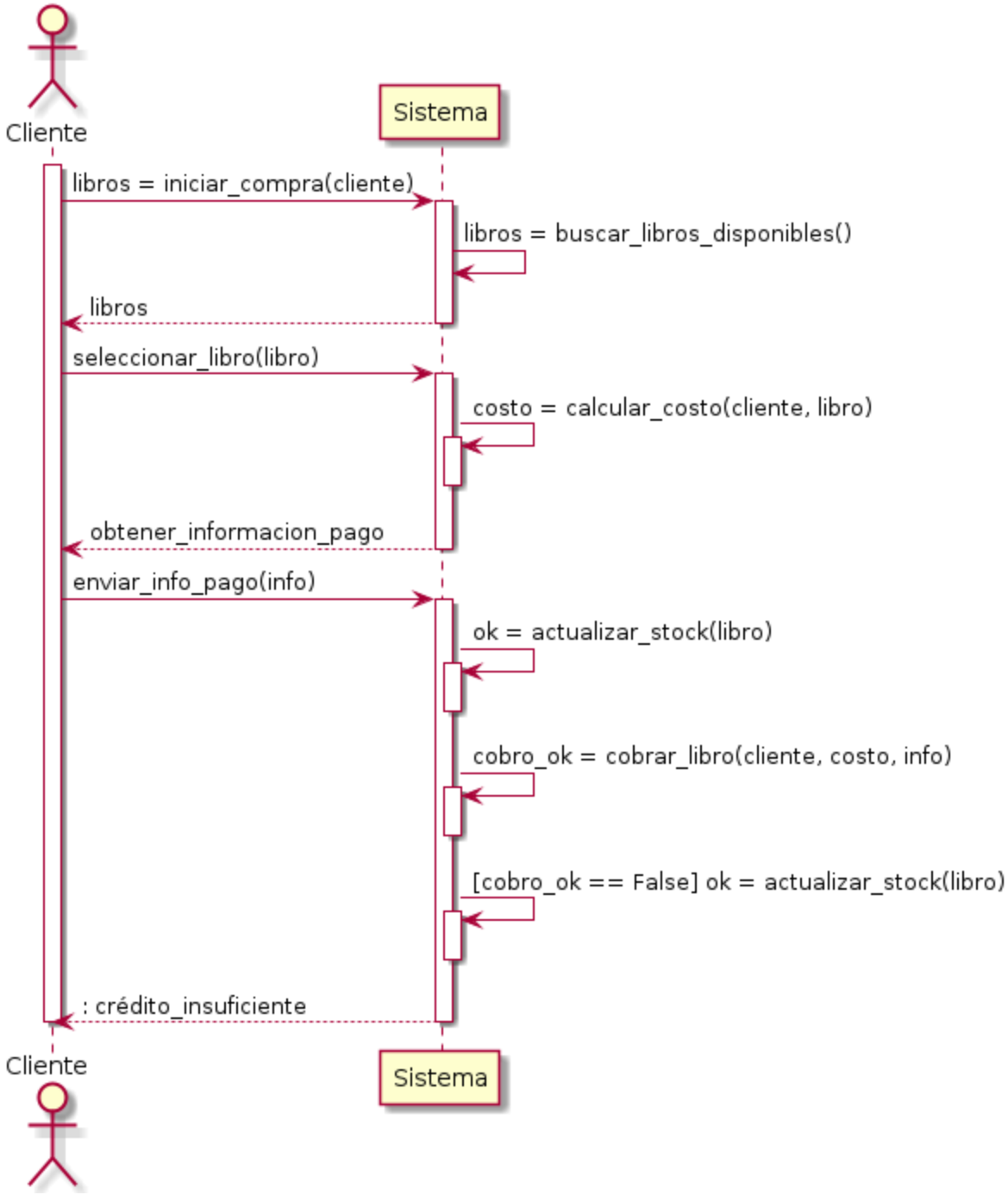
## c12 Comprar Libro - Flujo alternativo - Libros no disponibles



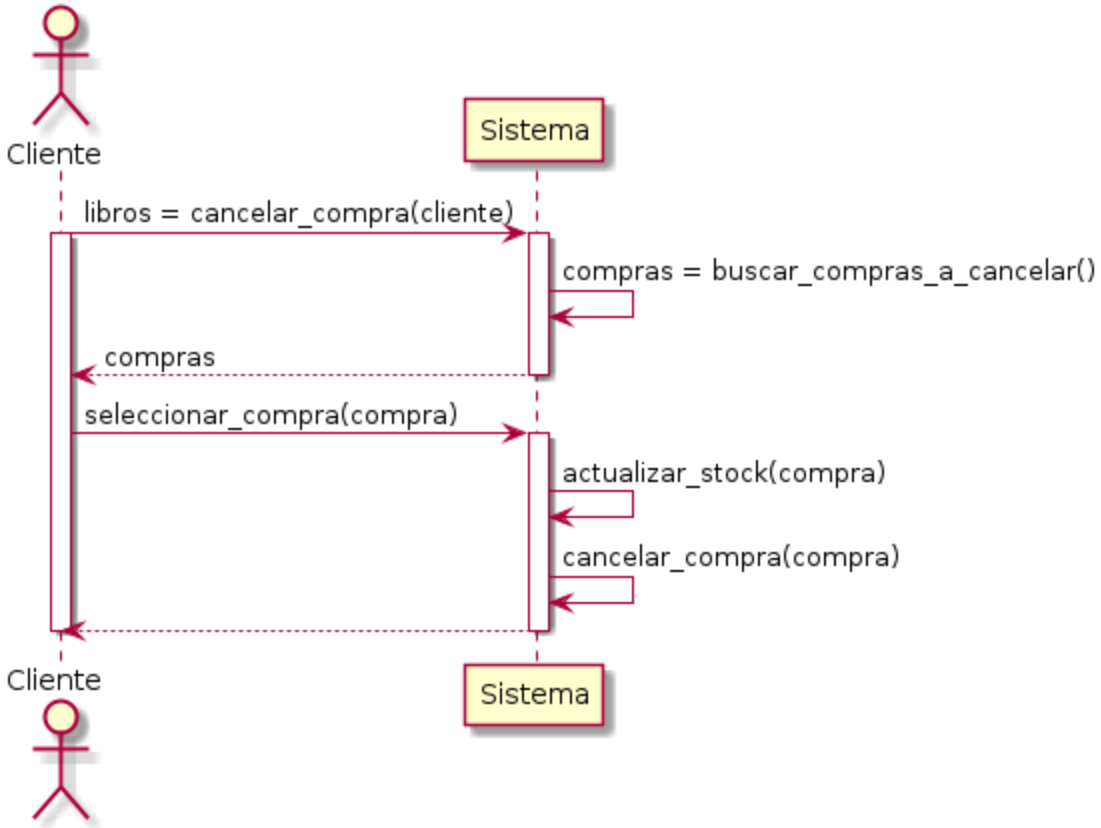

## c12 Comprar Libro - Fujo alternativo - Stock Insuficiente



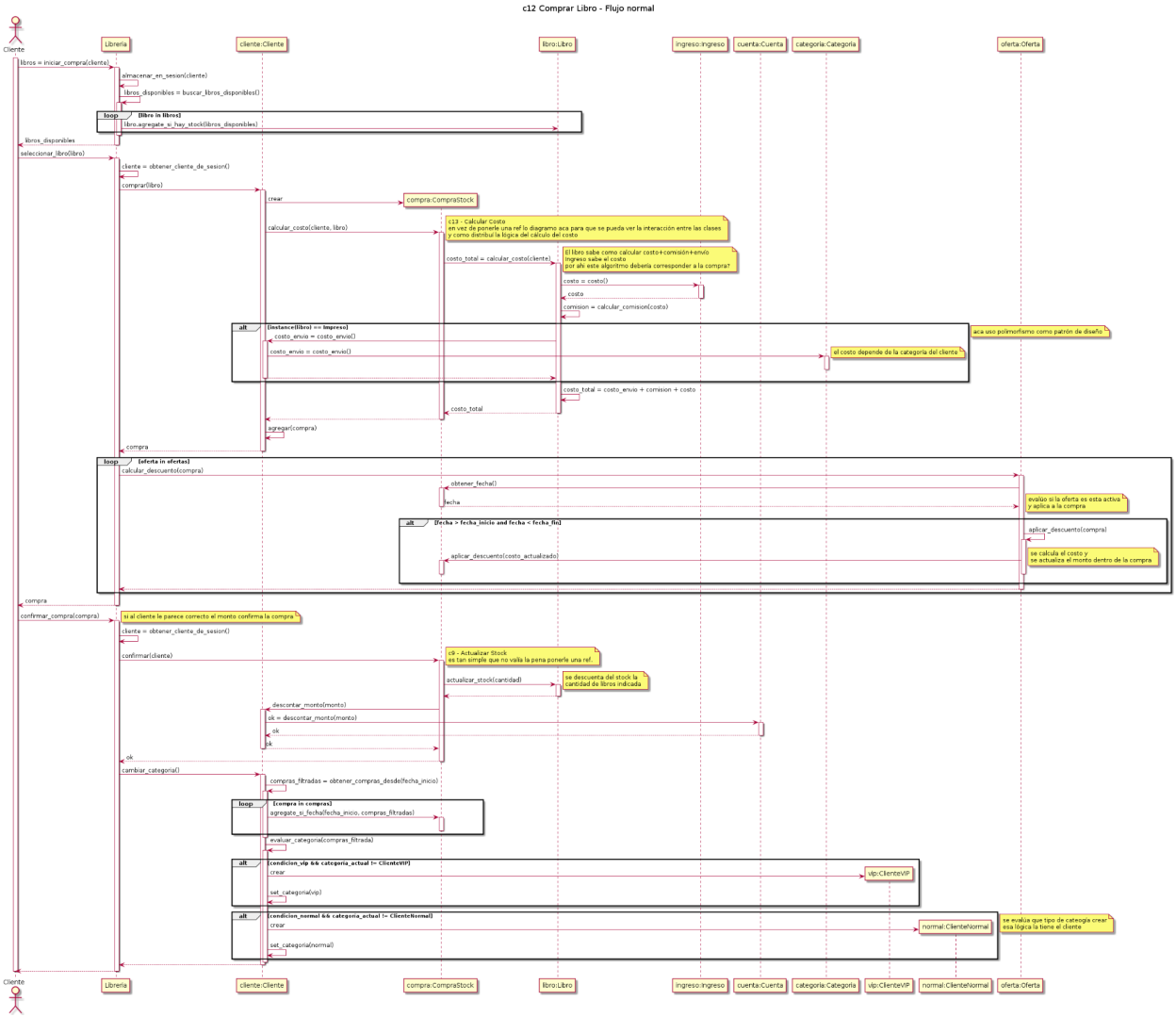
## c12 Comprar Libro - Flujo alternativo - Crédito Insuficiente



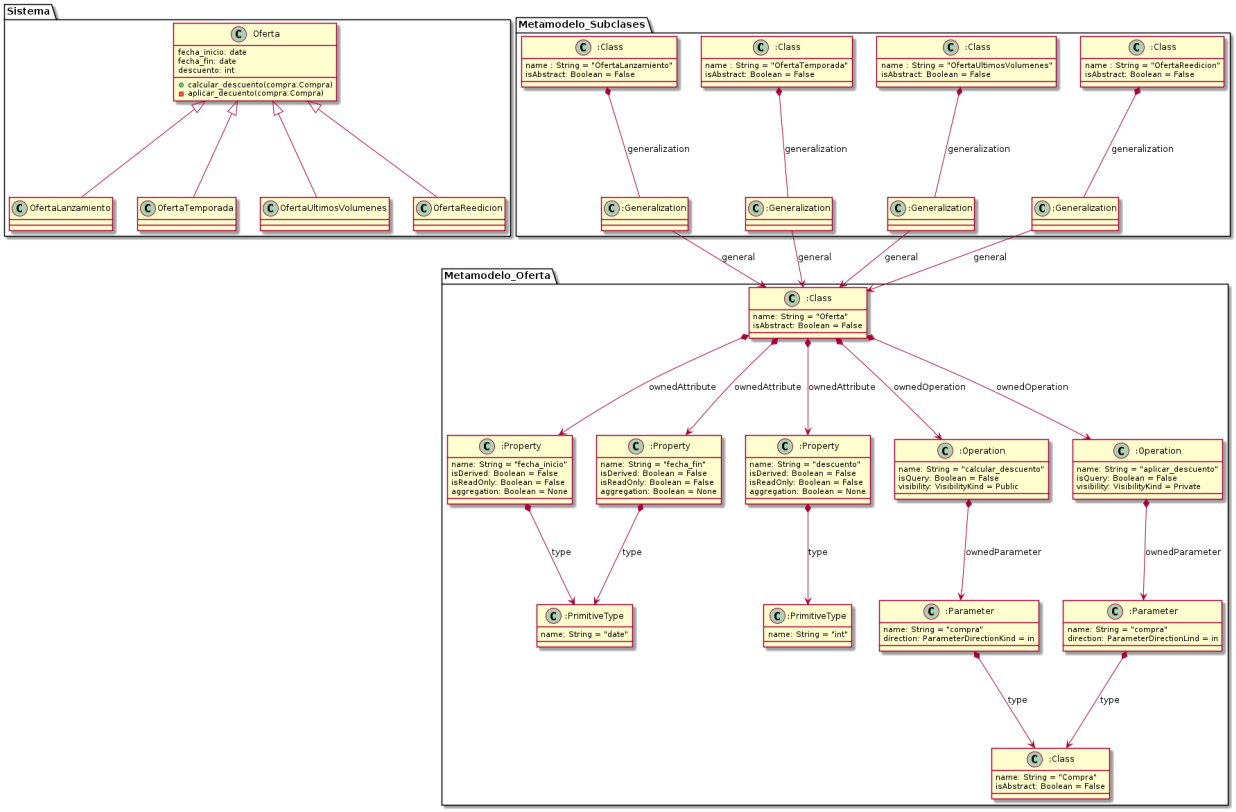
## c15 Cancelar Compra



# Diagramas de Secuencia



# Metamodelo de Herencia



# Consideraciones finales de la herramienta

A medida que se fue desarrollando el tp fuí encontrando pros y contras a la herramienta elegida. Las principales motivaciones para usar la herramienta fueron:
1 - Conocimiento previo ya que la uso para otros tipos de diagrama
2 - Simplificar el manejo del layout de los componentes del diagrama (o eso esperaba)
3 - Teniendo en cuenta que el proceso es iterativo e incremental, me pareció ideal para generar alternativas de representaciones de un modelo de forma simple. El cambio en el modelo era esperado.

A medida que desarrollaba el tp fui evaluando algunos casos que la herramienta no es ideal para manejar.

Problemas para unificar documentos!!. Al no ser una herramienta de modelado uml compleja, no realiza seguimiento de los cambios de los distintos documentos (casos de uso, diagrama de clases, diagramas de secuencia, etc). Con lo cual cada modificación realizada en uno de ellos, hay que “trackearla” manualmente y aplicarla a los otros diagramas y documentos del modelo. Esto es un GRAN problema ya que los diagramas están todos relacionados, con lo cual en modelos complejos el tracking de diferencias se hace complejo y manual.

Otro problema encontrado fue visualizar múltiples diagramas a la vez. No es posible salvo que se tengan múltiples ventanas del editor abiertas en distintos monitores. Por lo que, generar los mismos métodos en un diagrama de clase y de secuencia se torna propenso a errores. También las modificaciones en uno de esos diagramas implica modificar manualmente los demás sin poder verlos en simultáneo.

No permite asociar un texto describiendo el problema con los diagramas asociados a ese texto. por ejemplo:
“El comercio mantiene un control sobre su inventario. En el caso de los libros digitales, se tiene una licencia con un número determinado de copias para vender. Para el caso de los libros impresos, el control se lleva en las copias de cada libro”
Asociarlo con parte de los diagramas donde el concepto está representado.
En los casos de uso poder asociarlo con que casos de uso son.
En el caso de los diagramas de clase y dominio de sistema asociarlo con esos conceptos.

No encontré forma de anotar consideraciones generales en los diagramas. O sea notas sin asociarlas a algún componente en especial.
Ej:
precio = precio de costo + comisión + gastos envío
Lo que simplificaría el análisis del diagrama e interpretar la lógica correctamente.

Tampoco es posible generar anotaciones relacionadas a los diagramas para poder evaluar a futuro, dudas, etc. Por ejemplo dudas que surgieron a lo largo del análisis del tp que debía consultar, solo fueron posibles de anotar en archivos aparte del diagrama. En notas internas al diagrama se tornaba engorroso y modificaba el layout del mismo.

En diagramas de clases complejos no es simple identificar las asociaciones entre las entidades. Ya que se torna en demasiadas flechas mezcladas (esto es debido a que el layout lo realiza automáticamente). Para poder identificarlas mejor hay que manualmente modificar el color o el espesor de cada una de las asociaciones.

Con algunos cambios muy simples del modelo, cambia el layout completo del diagrama. Lo cual torna complejo identificar partes del mismo ya que hay que analizar el gráfico nuevamente. Ej la jerarquía de persona, aparece a la izquierda y con una modificación aparece a la derecha, etc.

No permite agrupar elementos dentro de una sección del diagrama, o no los agrupa de forma simple. Poniendo por ejemplo las clases dentro de packages se puede realizar, etc. pero no en todos los diagramas. Lo cual hace complejo para visualizar e identificar los conceptos. Existen formas de realizarlo usando inclusión de archivos, pero en el uso del tp no lo supe implementar correctamente y complicaba el desarrollo.
Ej : La Categoría de un Cliente me gustaría verla cerca del Cliente y no al otro extremo del diagrama.

Así y todo, esta herramienta tiene sus fuertes a la hora de modelar. Me permitió entre otras cosas:

Realizar consultas de forma ágil sin mucho trabajo de edición, ya que justamente uno de los contras descriptos anteriormente, ahora se transforma en un fuerte. El layout automático permite tener un diagrama con poco esfuerzo. Con lo cual es trivial generar alternativas para poder consultar aspectos del diagrama.

Otro fuerte es que permite mantener distintos diagramas en un mismo archivo, con lo cual la organización del código se favorece en determinadas situaciones. Permite incluir archivos separados en un mismo diagrama, y permite definir múltiples diagramas en un solo archivo.

El proceso de versionado es trivial, ya que se puede versionar usando git sobre archivos de texto, lo cual también permite analizar los cambios ocurridos en el diagrama a medida que se va iterando sobre la solución.

Poner notas cortas también es muy simple. Y el proceso del layout automático permite reacomodar el diagrama de forma simple. Dentro del diagrama de secuencia esto fue sumamente útil.

Y una de las grandes ventajas que le veo al proyecto es que es código abierto, en un estado estable y usable. Con posibilidad de integración con varias herramientas a futuro.
En mi caso lo usé integrado con Visual Studio Code, pero existen versiones para integrarlo con eclipse. Permite colaboración ya que se puede correr usando docker en modo “servidor” y compartiendo la url del mismo analizar y ver el mismo gráfico entre varios desarrolladores.

Conclusiones:

Usando la herramienta del modo que la usé para el tp, no la recomiendo para el modelado de grandes sistemas complejos. Creo que para esto, todavía hace falta desarrollar plugins para “trackear” los cambios entre los diagramas. Y compilar patrones de diseño de sintaxis plantuml que permitan manejar los problemas de layout normales que me he encontrado.

Si la recomendaría para prototipado rápido de un modelo, análisis de algunos conceptos de forma visual, modelado de sistemas simples o como etapa final generadora de gráficos de sistemas de modelado que controlen los aspectos detallados arriba.

Pablo Daniel Rey
2806/4