3.1.2.- Invocación a la Lógica del Negocio
3.2.1.- Patrón CONTROLADOR

Nombre: CONTROLADOR

PROBLEMA: ¿A quién se le asigna la responsabilidad de recibir o manejar eventos de entrada al sistema?

[Un evento de entrada al sistema es un evento generado por un actor externo, que se asocia con una operación del sistema.]

SOLUCIÓN: A un objeto (clase) que representa al sistema global, dispositivo o subsistema .

Es un único OBJETO para todo el sistema donde se colocan TODAS LAS OPERACIONES DEL SISTEMA. También se suele llamar objeto "FACADE" (o FACHADA).

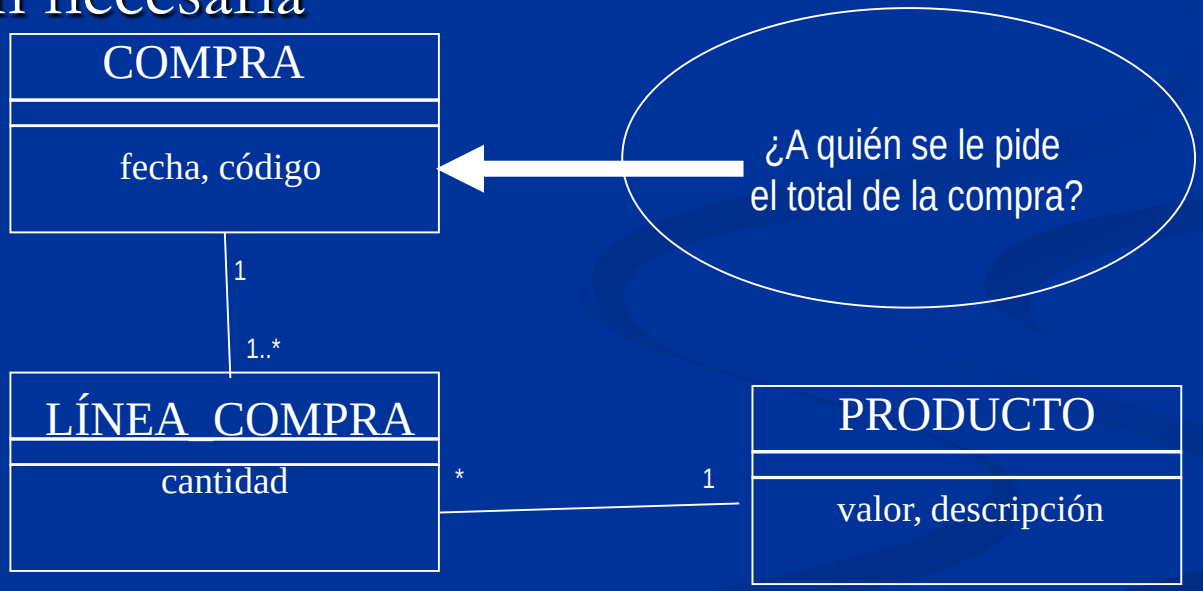
El CONTROLADOR es el que ofrecerá las operaciones de la LÓGICA DEL NEGOCIO

3.2.2.- Patrón EXPERTO

Nombre: EXPERTO

PROBLEMA: ¿A quién se le pide que busque un determinado dato o genere información?

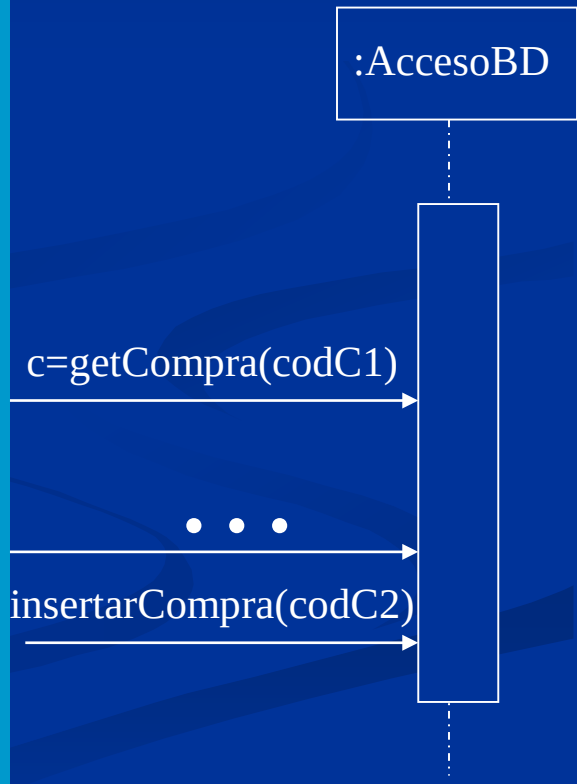
SOLUCIÓN: Al objeto (clase) que cuenta con los datos o información necesaria



NOTA IMPORTANTE: si es un objeto del dominio, hay que asegurarse de que se encuentra cargado en la memoria principal. En ese caso, ese objeto del dominio sería el EXPERTO.

Acceso al nivel de datos para cargar los objetos del dominio

Los objetos del dominio que queramos invocar deben estar cargados en la memoria principal. Lo haremos usando un objeto de una clase (AccesoBD) que será:
1) el **controlador** de la BDOO que se encargará de gestionar la recuperación (además de inserción, borrado y modificación) en la BD;
o bien 2) (tal vez) **experto** en acceder a los objetos del dominio



3.2.3.- Patrón CREADOR

Nombre: CREADOR

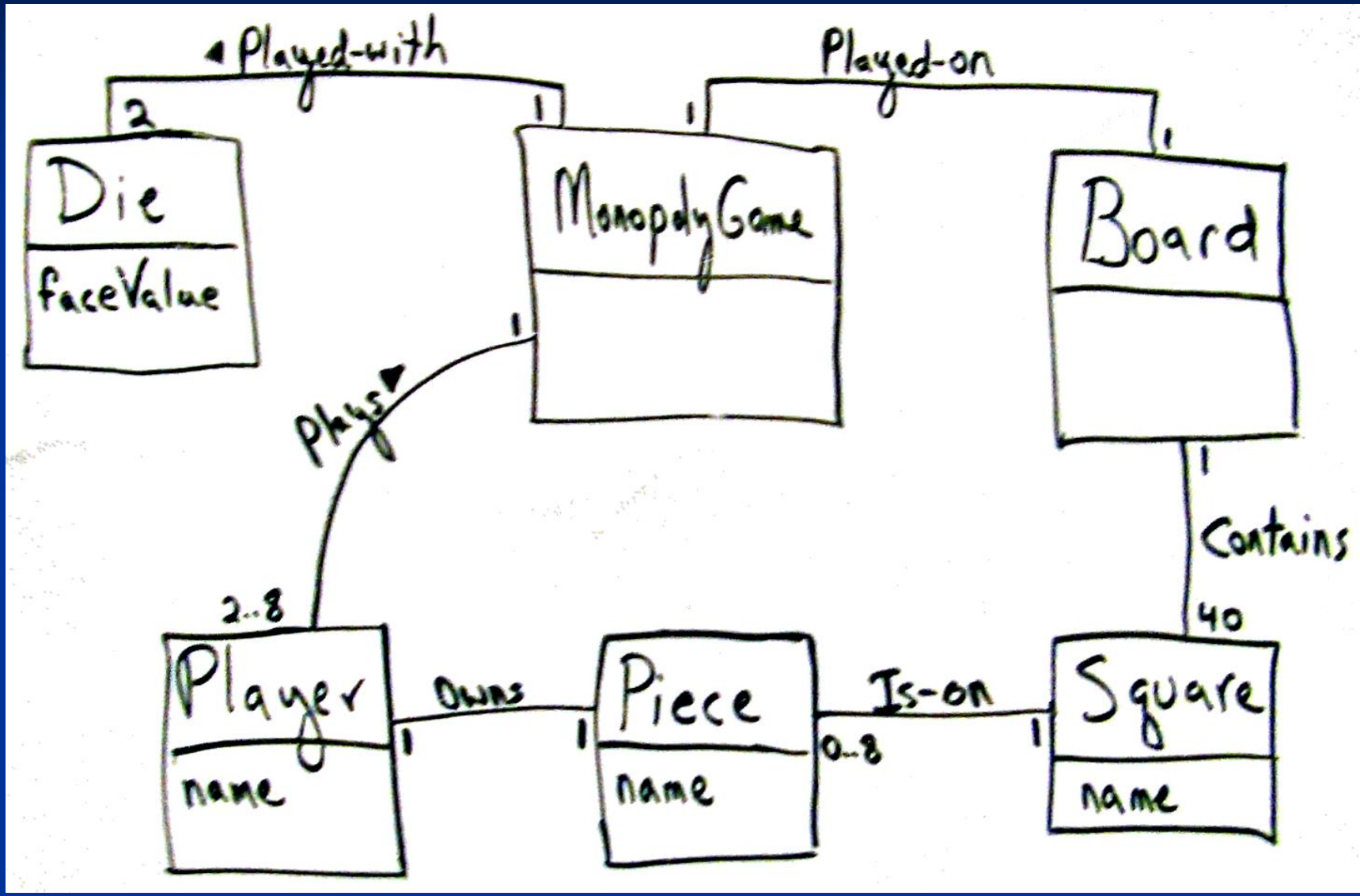
PROBLEMA: ¿Quién debe crear los objetos de una clase A?

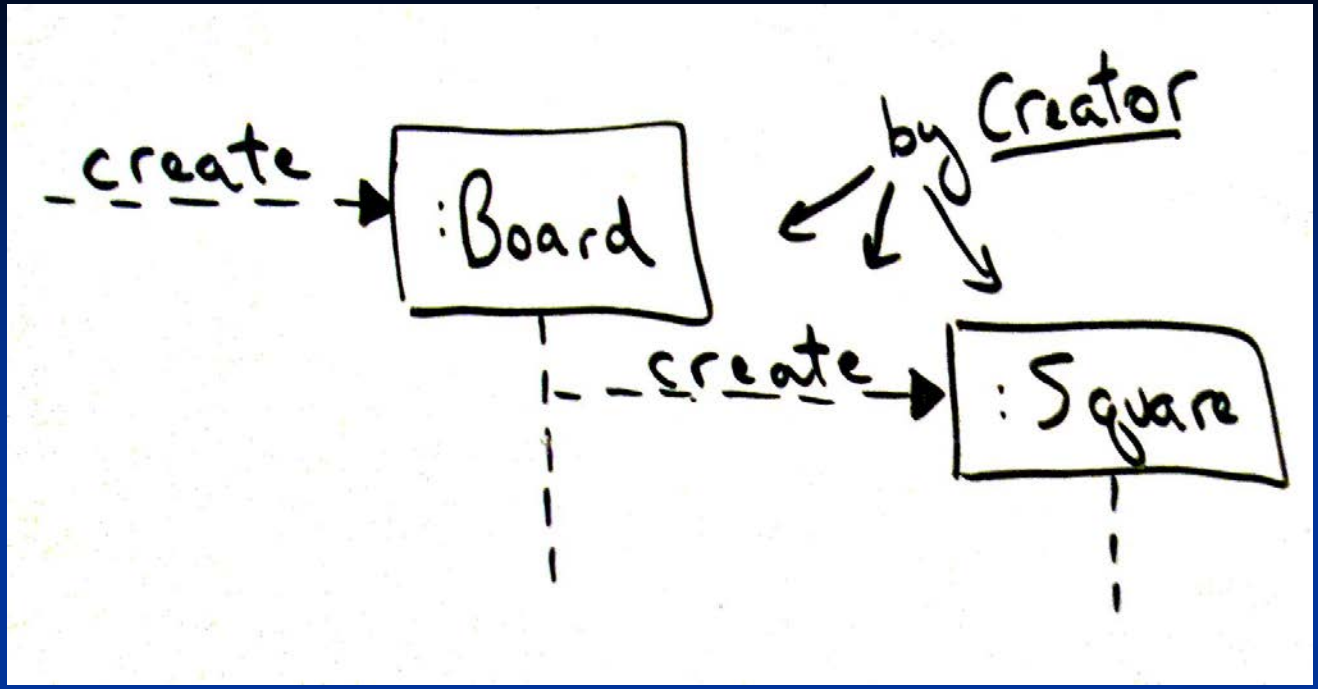
SOLUCIÓN: Esa responsabilidad se le añadirá a la clase B si se cumple alguna de estas condiciones:

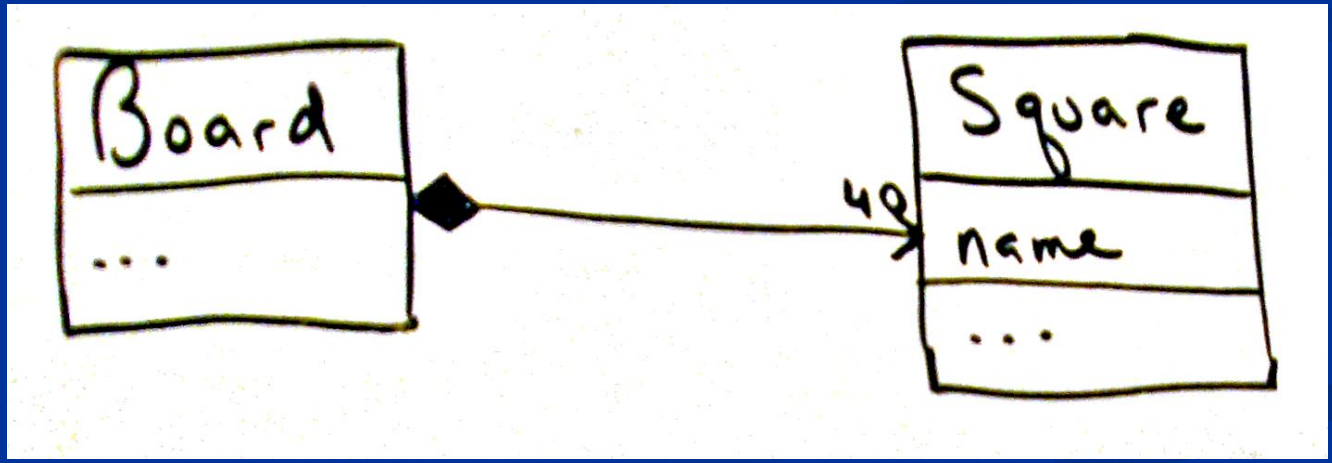
- La clase B guarda los objetos de la clase A
- La clase B está formada por objetos de A (AGREGACIÓN/COMPOSICIÓN)
- Cuando hay que crear un objeto de A, B tiene todos los datos de inicialización necesarios

  (B es un EXPERTO en la creación de A)

¿Quién crea los cuadros (Squares)?




create
:Board
by Creator
create
:Square


Board
...
49
Square
name
...

3.2.4.- Patrón BAJO ACOPLAMIENTO

Nombre: BAJO ACOPLAMIENTO

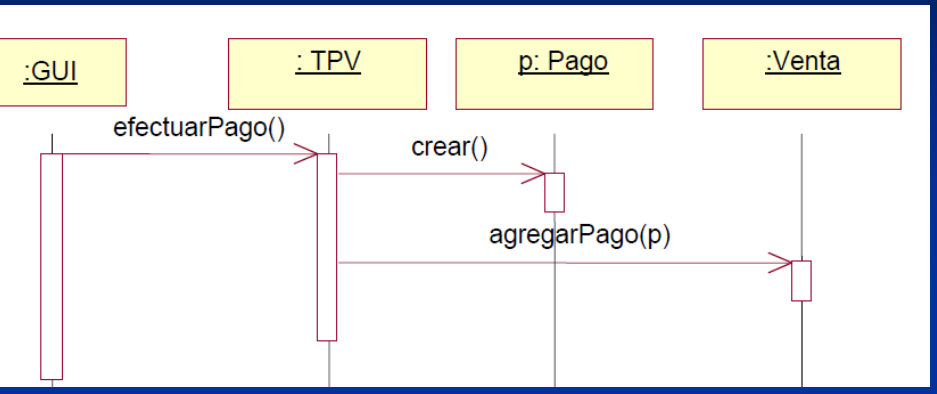
PROBLEMA: ¿Cómo reducir las dependencias entre clases?

SOLUCIÓN: Asignar la responsabilidad de manera que el acoplamiento permanezca bajo

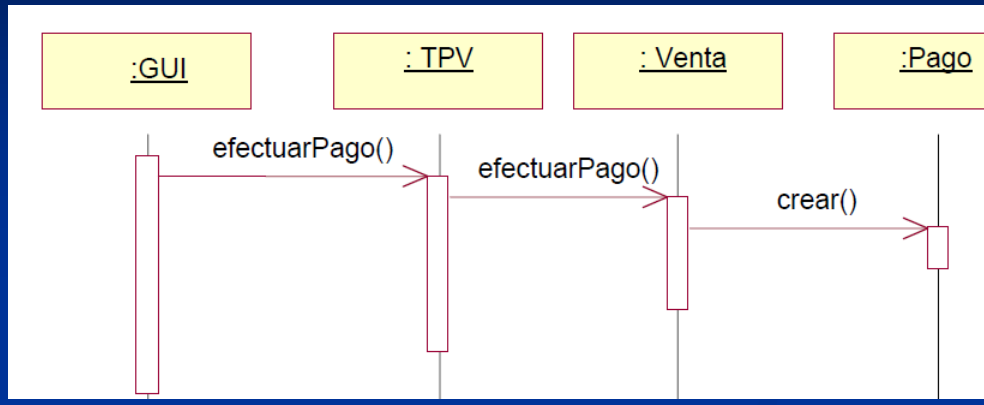
Existe acoplamiento entre A y B si A usa B (A tiene atributo del tipo B, o bien un método que usa o devuelve B,…)

Ejemplo: BAJO ACOPLAMIENTO

(1)



(2)



El diseño (2) tiene acoplamiento más bajo:

- En ambos Venta está acoplada a Pago
- En (1) TPV está acoplada a Pago y a Venta
- En (2) TPV está acoplada a Venta, ¡pero no a Pago !

Nota: el nivel de acoplamiento no se puede considerar de manera aislada a otros patrones como el EXPERTO y el ALTA COHESIÓN

3.2.5.- Patrón ALTA COHESIÓN

Nombre: ALTA COHESIÓN

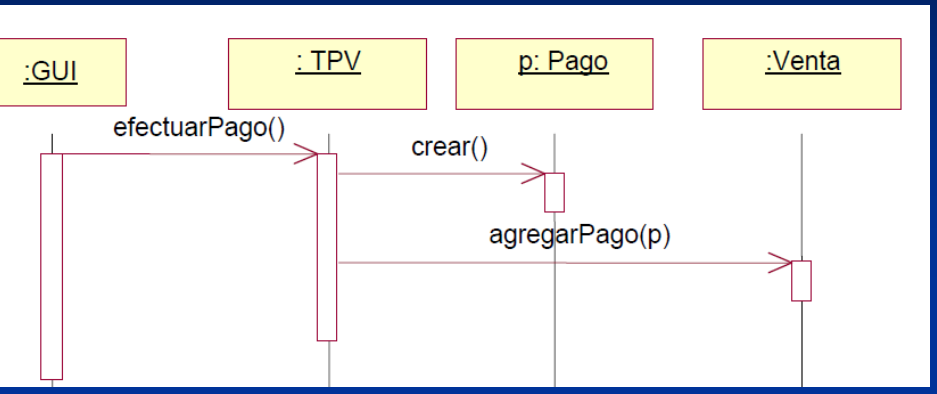
PROBLEMA: ¿Cuánto están relacionadas las responsabilidades de una clase? ¿Cómo mantener la complejidad manejable?

SOLUCIÓN: Asignar una responsabilidad de manera que la cohesión permanezca alta.

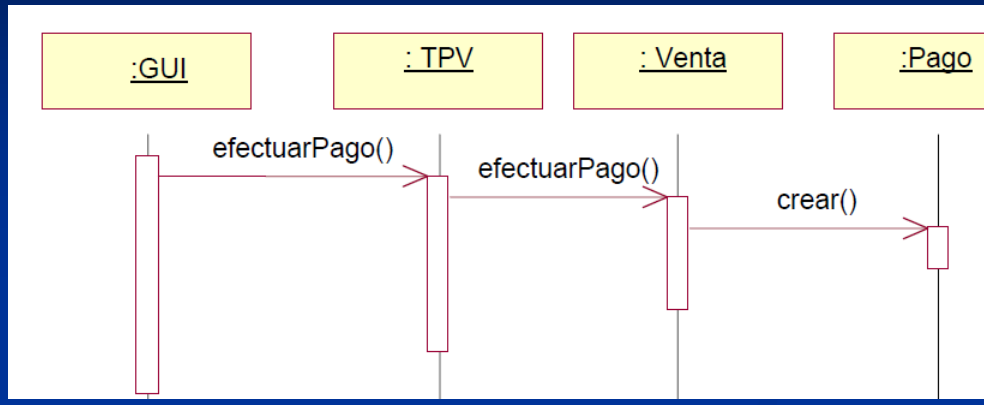
Una clase tiene baja cohesión, si es una clase que tiene muchas responsabilidades no relacionadas, que hace demasiado trabajo, que no delega. Son clases difíciles de entender, reutilizar, mantener,…

Ejemplo: ALTA COHESIÓN

(1)



(2)



El diseño (1) tiene una cohesión más baja:

- Tiene una clase (TPV) que se encarga de crear el Pago, no ha delegado la creación del Pago en Venta.

Nota: el nivel de cohesión no se puede considerar de manera aislada a otros patrones como el EXPERTO y el BAJO ACOPLAMIENTO

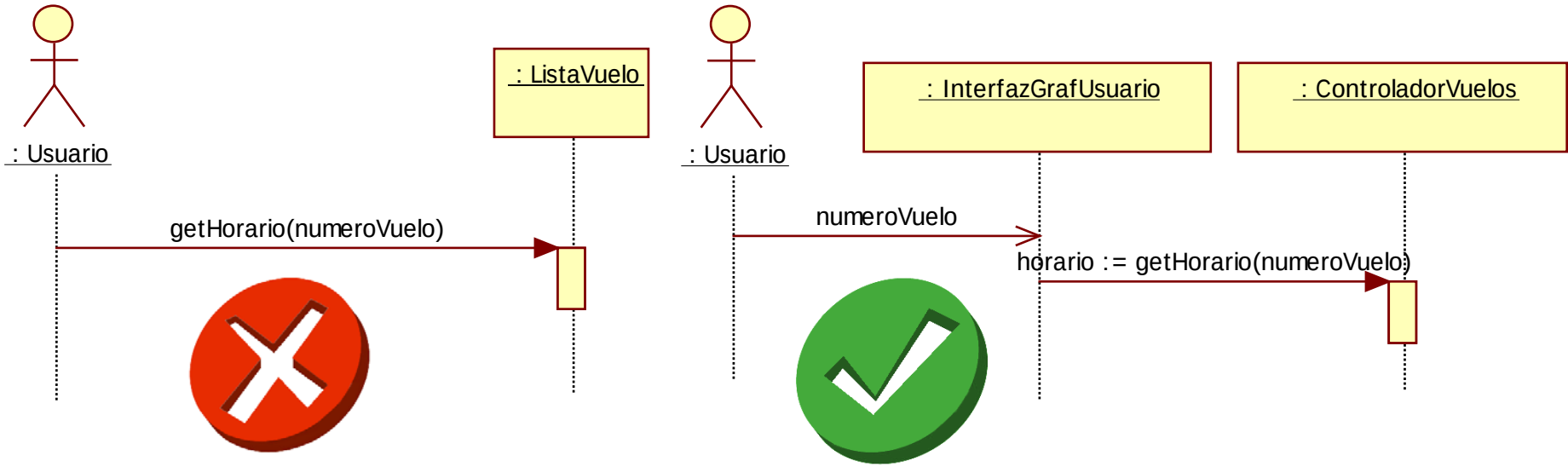
Errores típicos y “cosas” permitidas y convenientes

Error típico: no utilizar una clase interfaz/frontera para interactuar con el actor

Flujo de Eventos:

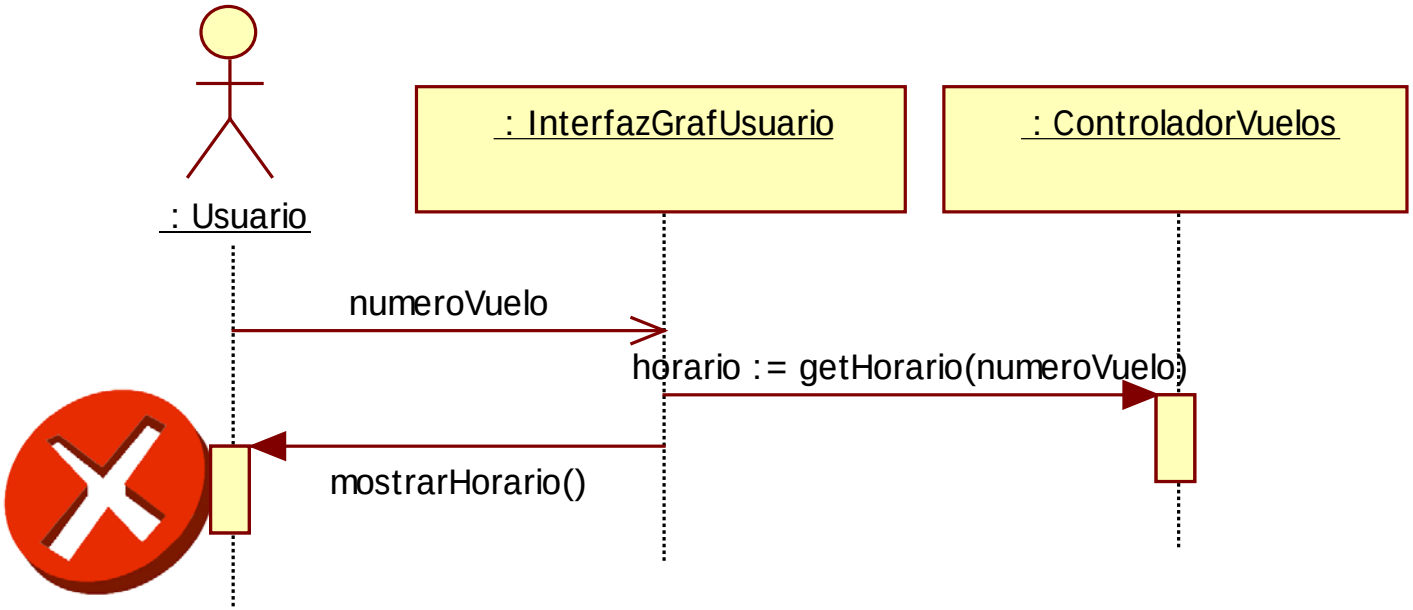
1.- El usuario proporciona un número de vuelo

2.- El sistema obtiene el horario de dicho vuelo y se le muestra al usuario



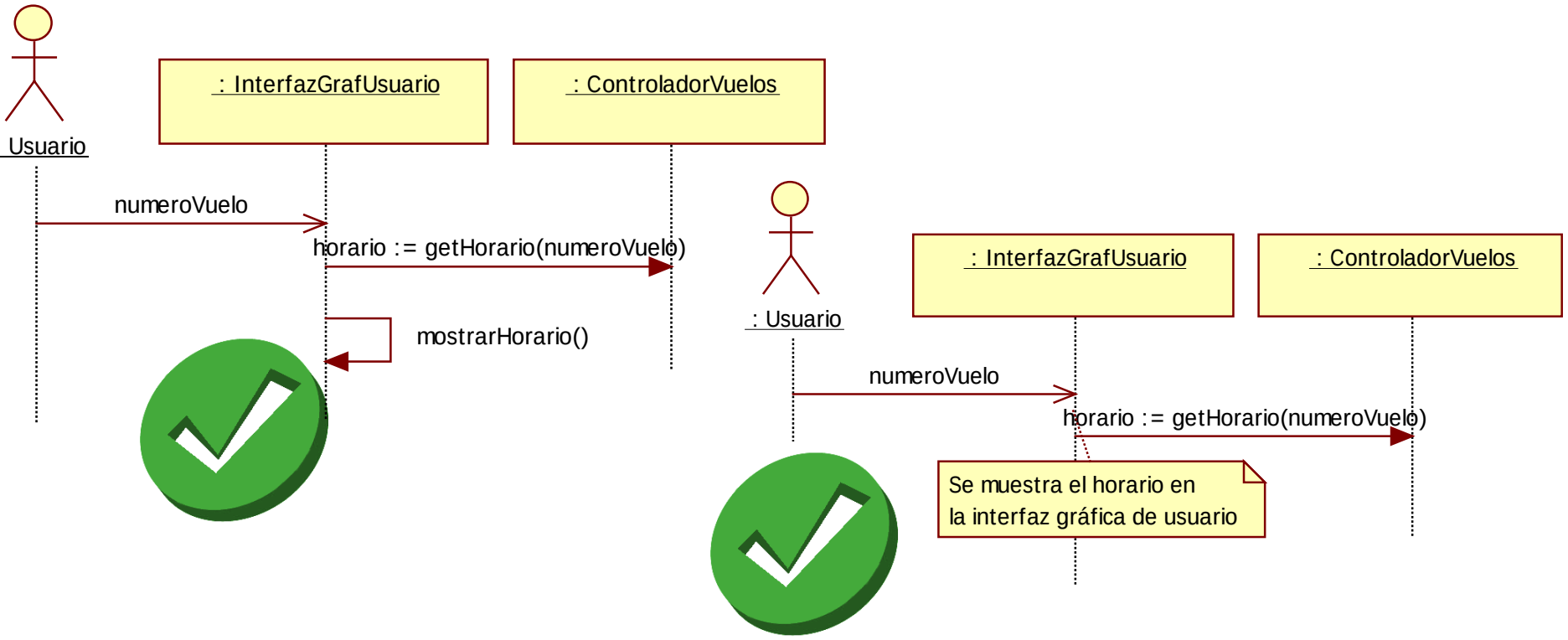
Error típico: solicitar que un actor ejecute un método

Flujo de Eventos:
1.- El usuario proporciona un número de vuelo
2.- El sistema obtiene el horario de dicho vuelo y se le muestra al usuario

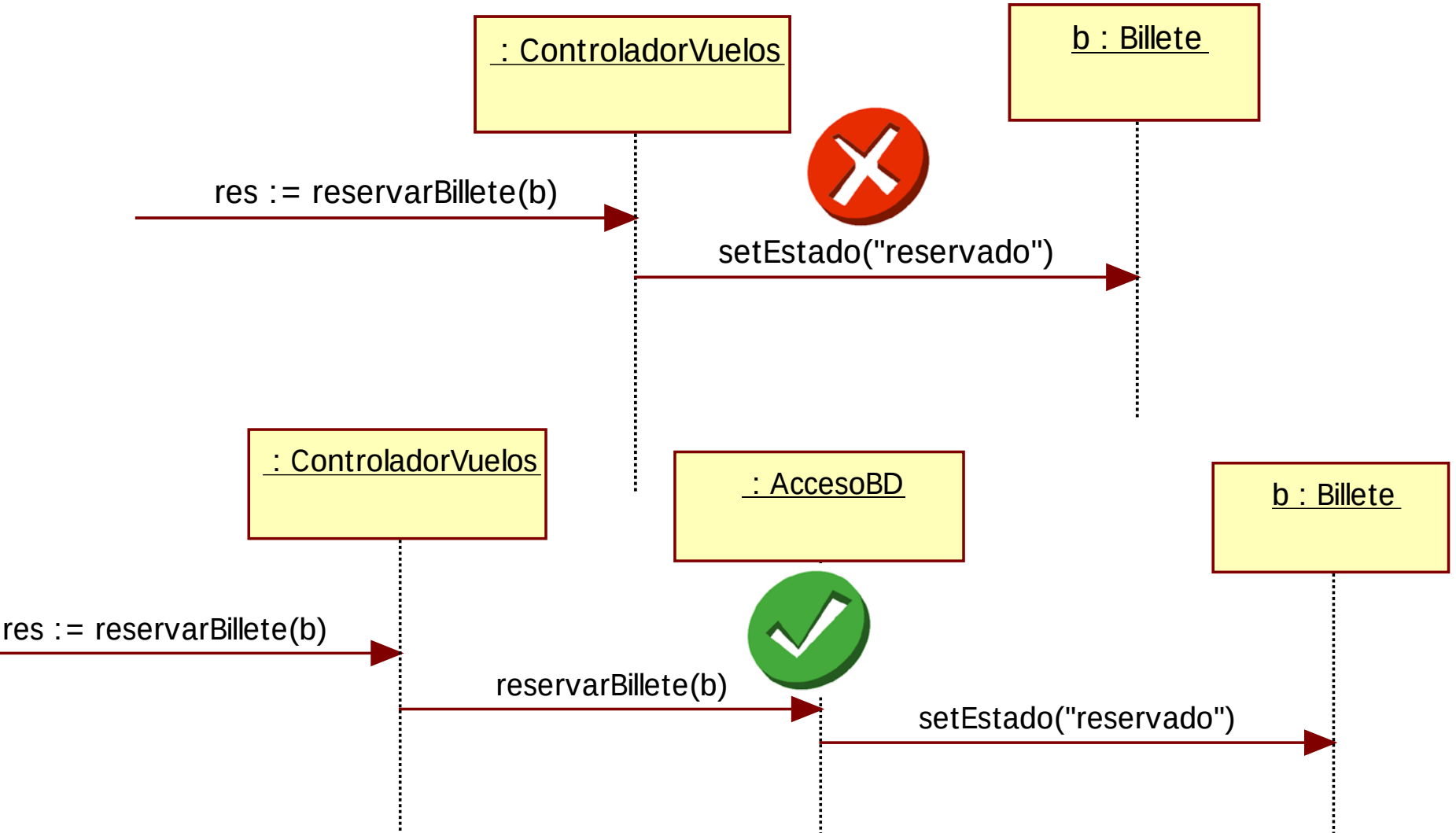


Error típico: solicitar que un actor ejecute un método

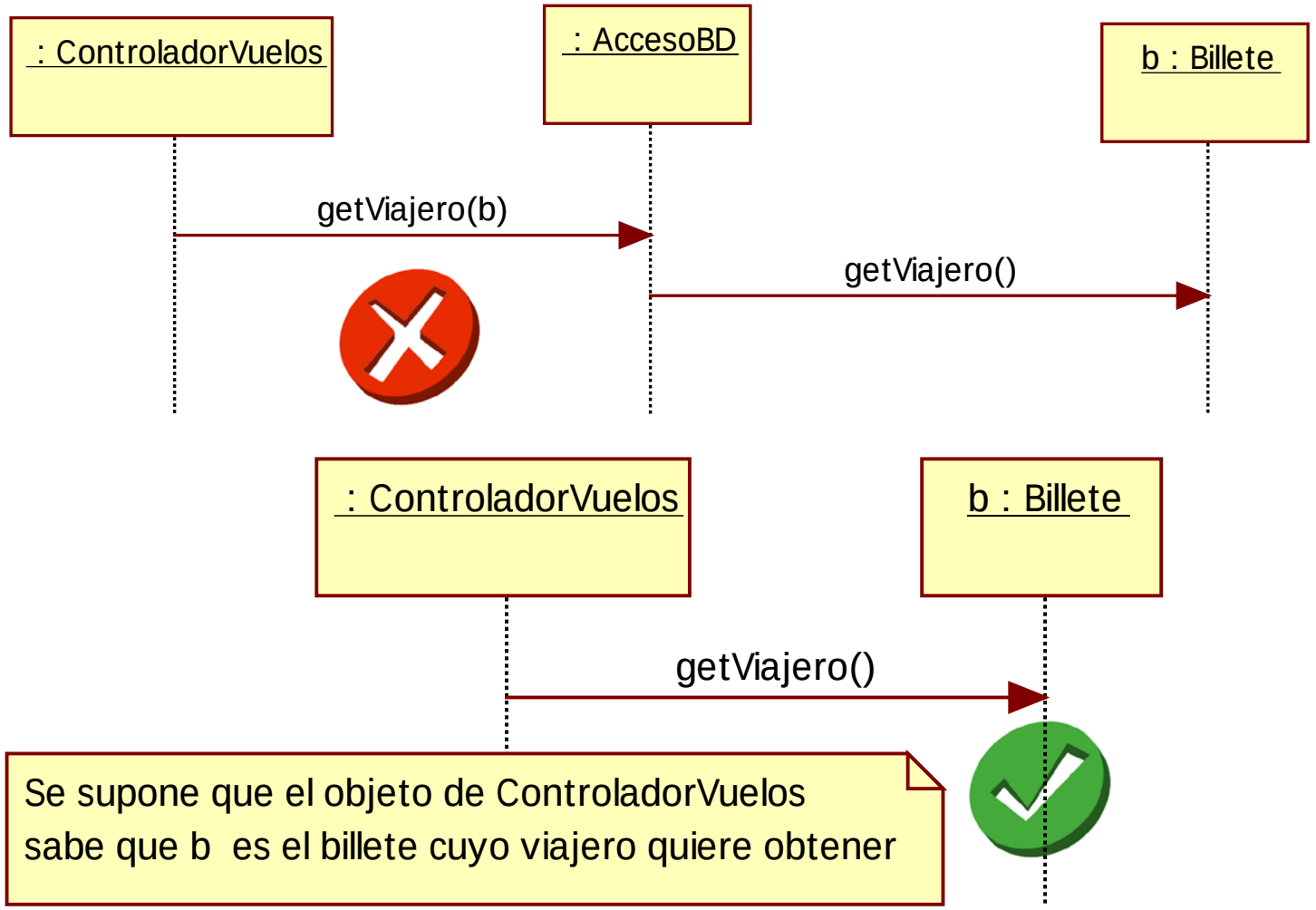
Flujo de Eventos:
1.- El usuario proporciona un número de vuelo
2.- El sistema obtiene el horario de dicho vuelo y se le muestra al usuario



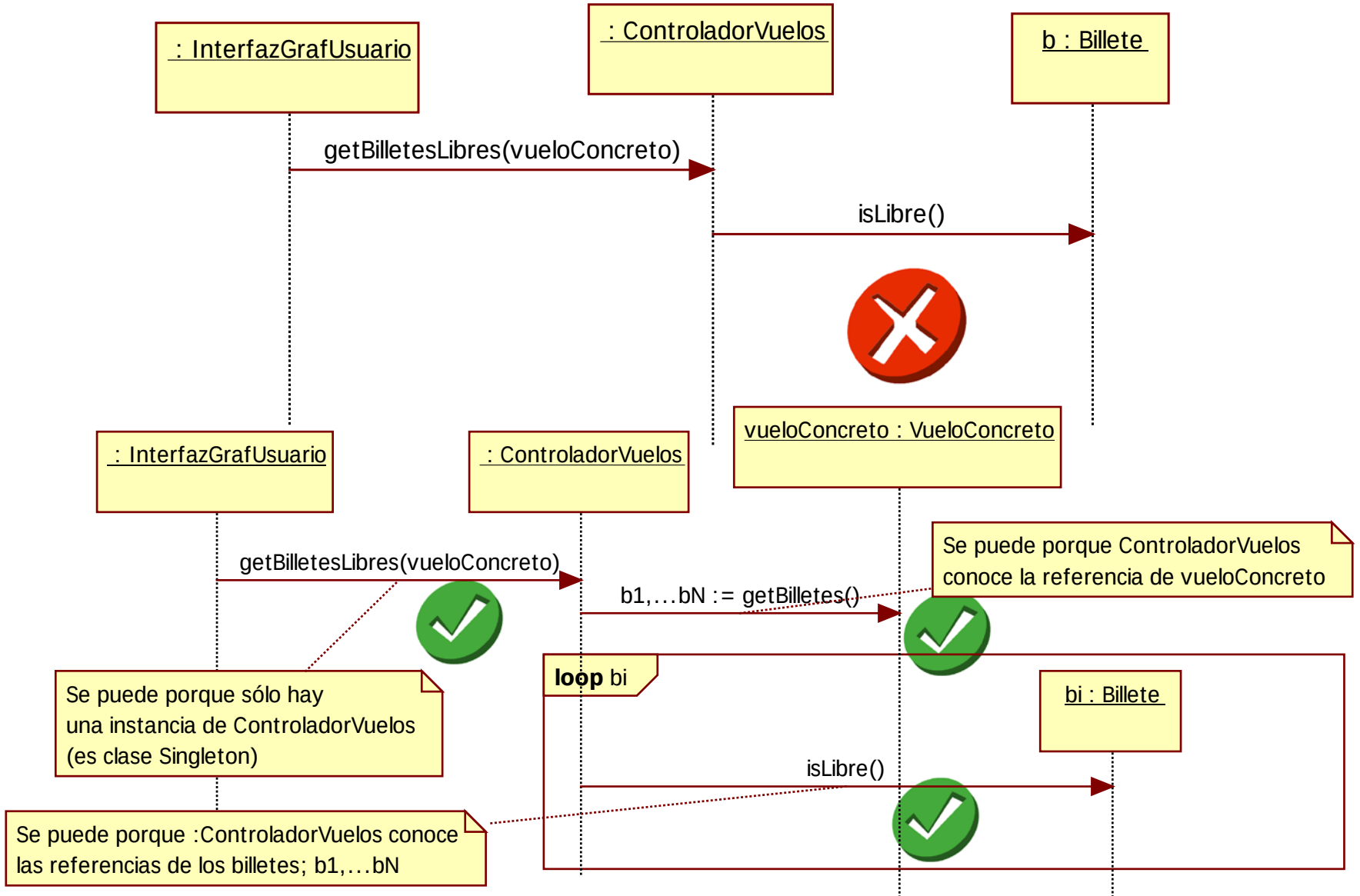
Error típico: realizar operaciones de actualización/borrado/inserción directamente en los objetos del dominio



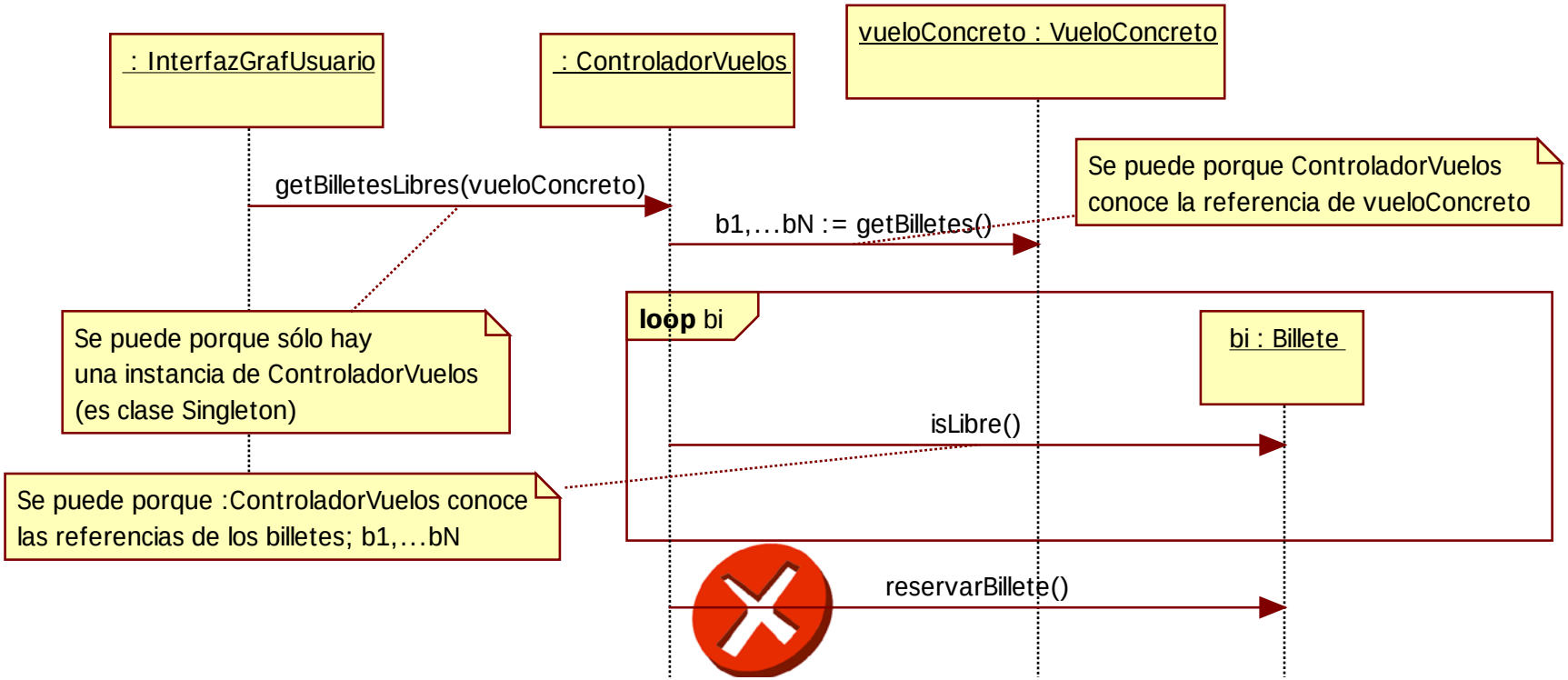
Error típico: consultar datos a través de la BD, conocido el objeto del dominio



Error típico: enviar un método a un objeto cuya referencia no es conocida



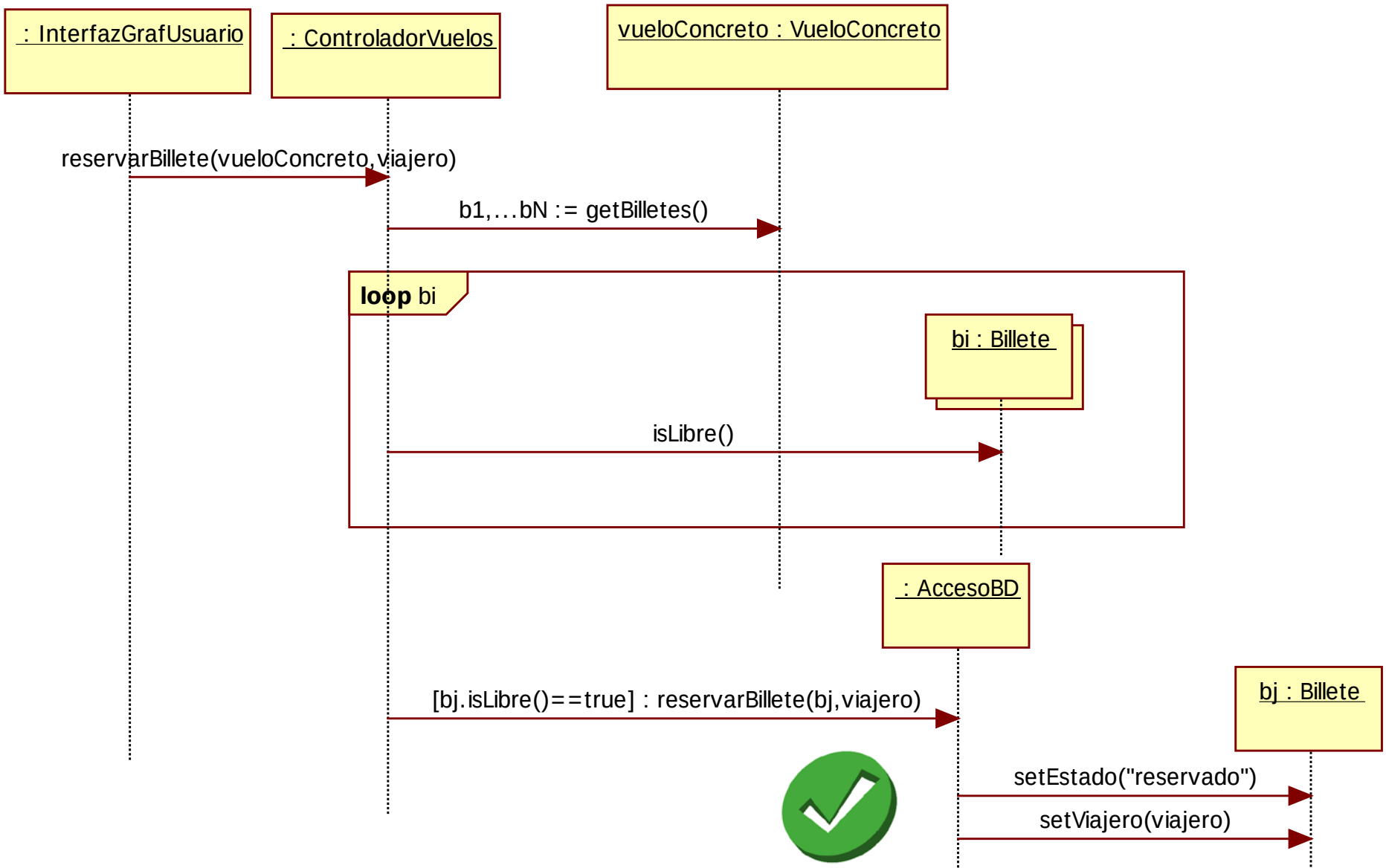
Error típico: realizar más acciones que la responsabilidad asignada



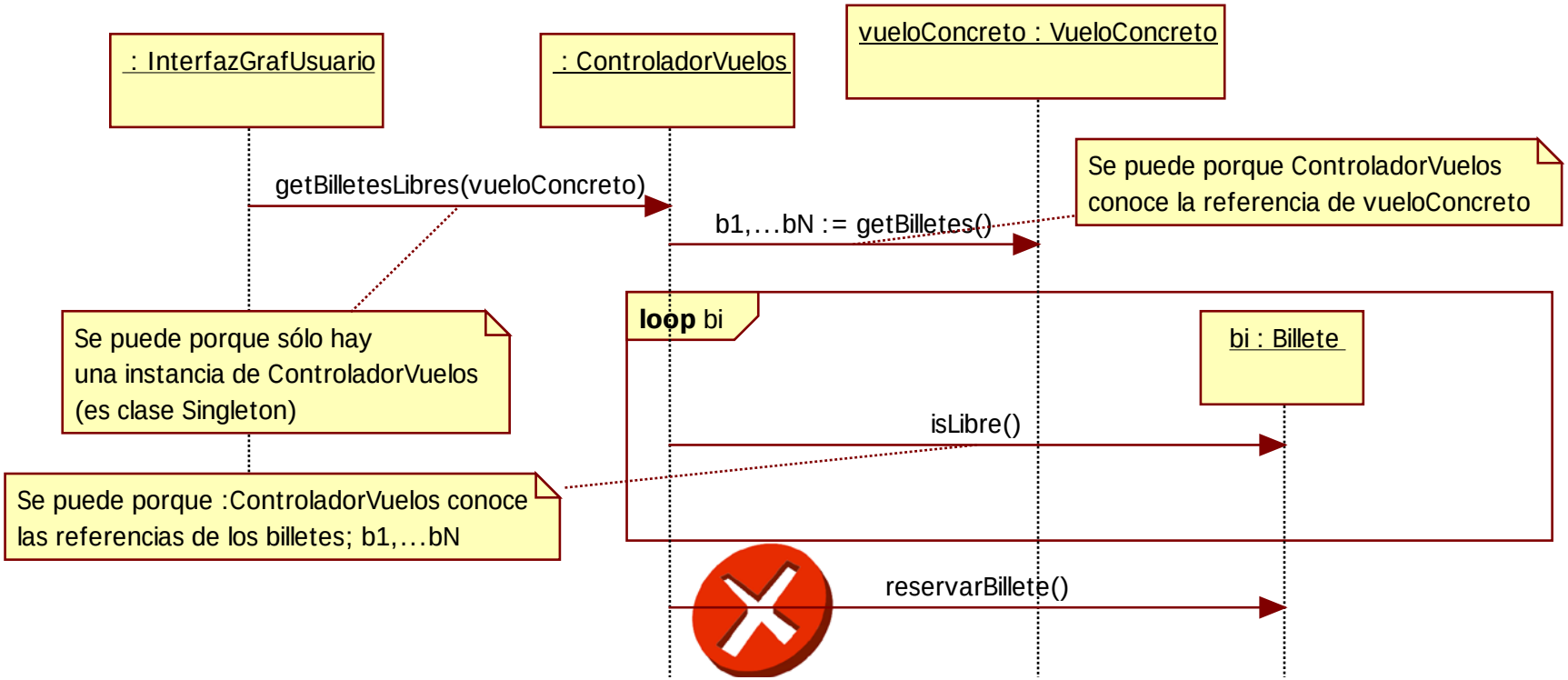
La responsabilidad era getBilletesLibres(vueloConcreto) NO es coherente que dentro de ella se reserve un billete.

Nota: por otro lado, tampoco se haría así reservar billete (actualizando directamente un objeto del dominio)

Error típico: realizar más acciones que la responsabilidad asignada

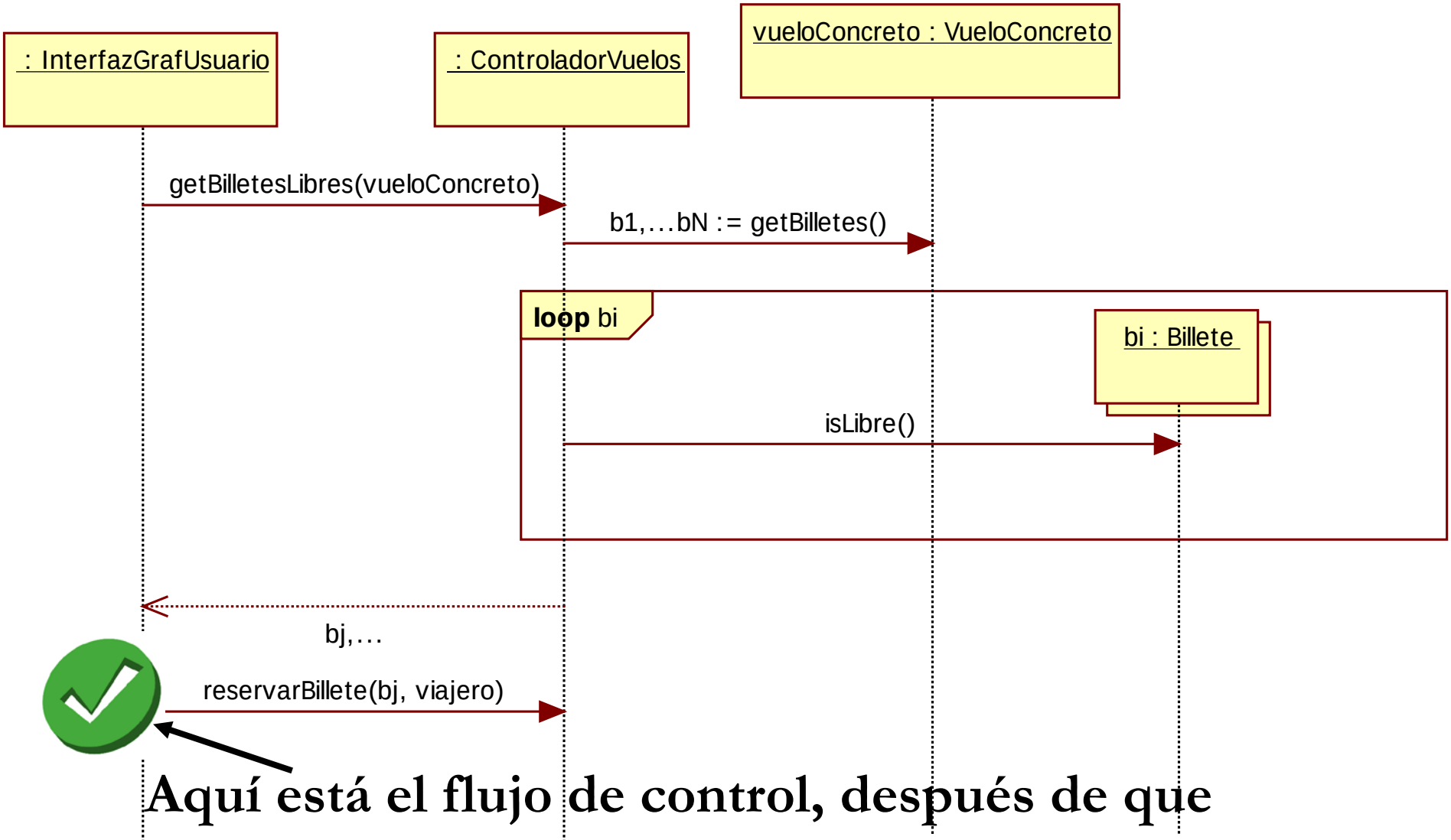


Error típico: no respetar el flujo de control en el diagrama de secuencia



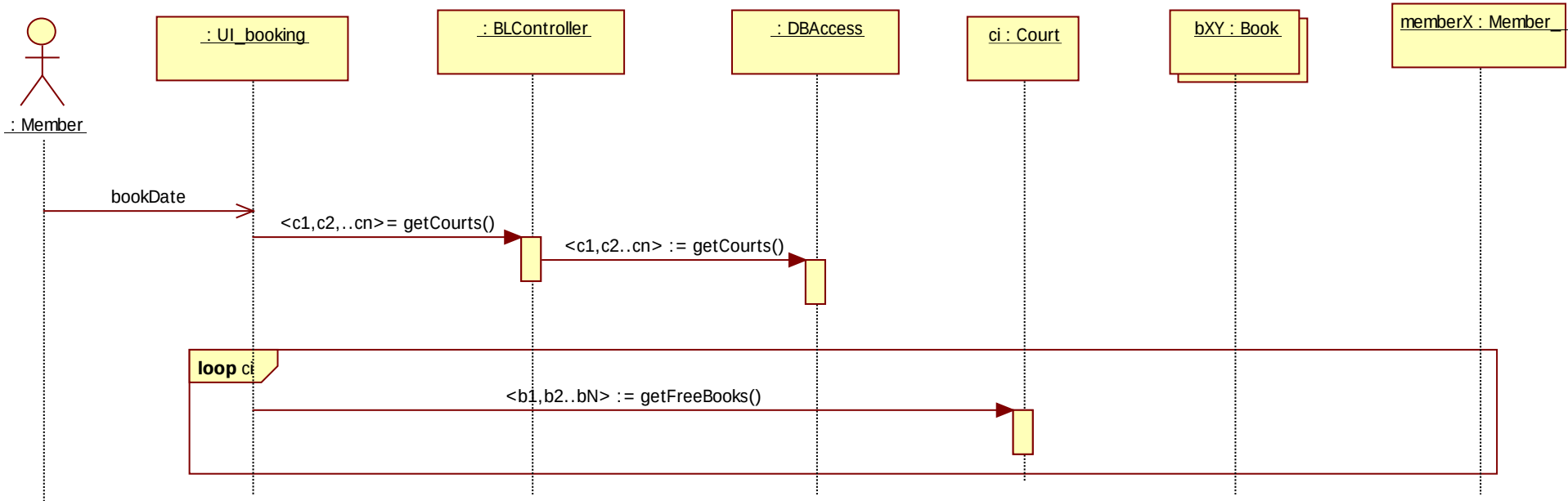
Tal vez la idea del diseño era, una vez obtenidos los billetes libres, entonces reservar un billete, pero en ese caso, el FLUJO DE CONTROL estaría en "InterfazGrafUsuario", y NO en el objeto "ControladorVuelos"

Error típico: no respetar el flujo de control en el diagrama de secuencia



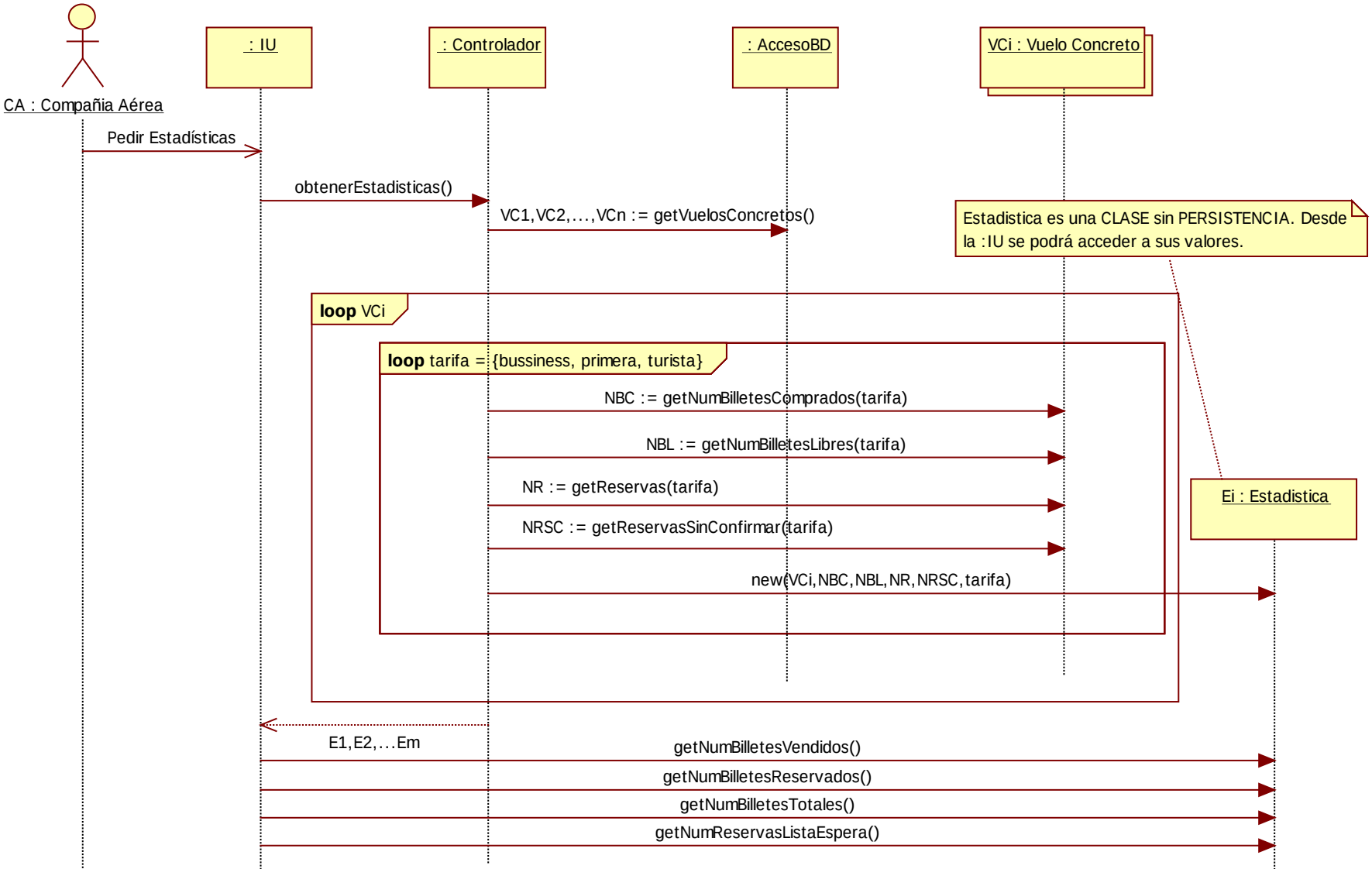
Aquí está el flujo de control, después de que "getBilletesLibres(vueloConcreto)" termine

Se puede ACCEDER a objetos del dominio directamente desde la interfaz, solamente para CONSULTAR datos



NOTA: en realidad, en la implementación veremos que se accede a COPIAS de dichos objetos del dominio y no a los objetos reales del dominio que realmente se encuentran en otra máquina remota (servidor de datos)

Se pueden CREAR objetos NO persistentes, [1) para enviar resultados a la interfaz]



[o 2) para diseñar algoritmos más complejos]