Practicas de Arquitectura del Software

Video Club “La Esquina”

Realizado por

Mariano Martínez Cañada
Jesús Rodríguez Vicente

4º Ingeniería Informática – Junio de 2002

Indice

1. Introducción

El objetivo de esta práctica es aplicar el *Proceso UML* a la especificación de requisitos del Videoclub "La Esquina". A partir de dicha especificación se ha seguido el Proceso software basado en UML para sistemas de información. Caracteriza principalmente este tipo de procesos el estar basados en los *Casos de Uso* y por ser un proceso *iterativo e incremental*, centrándose en los aspectos críticos en las primeras iteraciones para minimizar los riesgos. El siguiente apartado explica de forma general las etapas de este proceso software.

1.1 Proceso software basado en UML

Las etapas del proceso son seis: Modelado del Negocio, Modelado de Requisitos, Modelado del Análisis, Modelado del Diseño, Implementación y Validación.

En la primera etapa del Modelado del Negocio identifican los procesos del negocio, se definen los casos de uso del negocio, se identifican los roles (diagrama de roles) y se modelan los flujos de tareas asociado a cada proceso de negocio, mediante escenarios y diagramas de procesos que muestran la interacción entre roles para conseguir el objetivo. Dado la pequeña magnitud de la aplicación a construir, no ha sido necesario realizar el Modelado del Negocio para esta práctica.

En la segunda etapa, por la que comenzamos directamente, se define el Modelado de Requisitos mediante los Casos de Uso (CdU) y el Modelo Conceptual.

Un CdU es creado para cada actividad que es soportada por el sistema, y proporciona un buen nivel de granuralidad para el desarrollo posterior. En ellos se describen principalmente los pasos que va a seguir la interacción con el sistema que el CdU representa.

En las etapas de Modelado del Análisis y Diseño se define para cada CdU un *diagrama de secuencia del sistema*, que muestra los eventos que un actor genera durante la interacción con el sistema. Cada evento da origen a una operación del sistema. Los *contratos* describen los efectos de cada una de las operaciones; suponen pues, una descripción más detallada sobre el comportamiento del sistema en términos de cambios de estado a los objetos del Modelo Conceptual tras la ejecución de una actividad con el sistema. Es por esto que la parte principal de un contrato sean las Pre y sobre todo, las Post condiciones, en las que se indican los objetos o asociaciones creadas o eliminadas, así como la modificación de atributos.

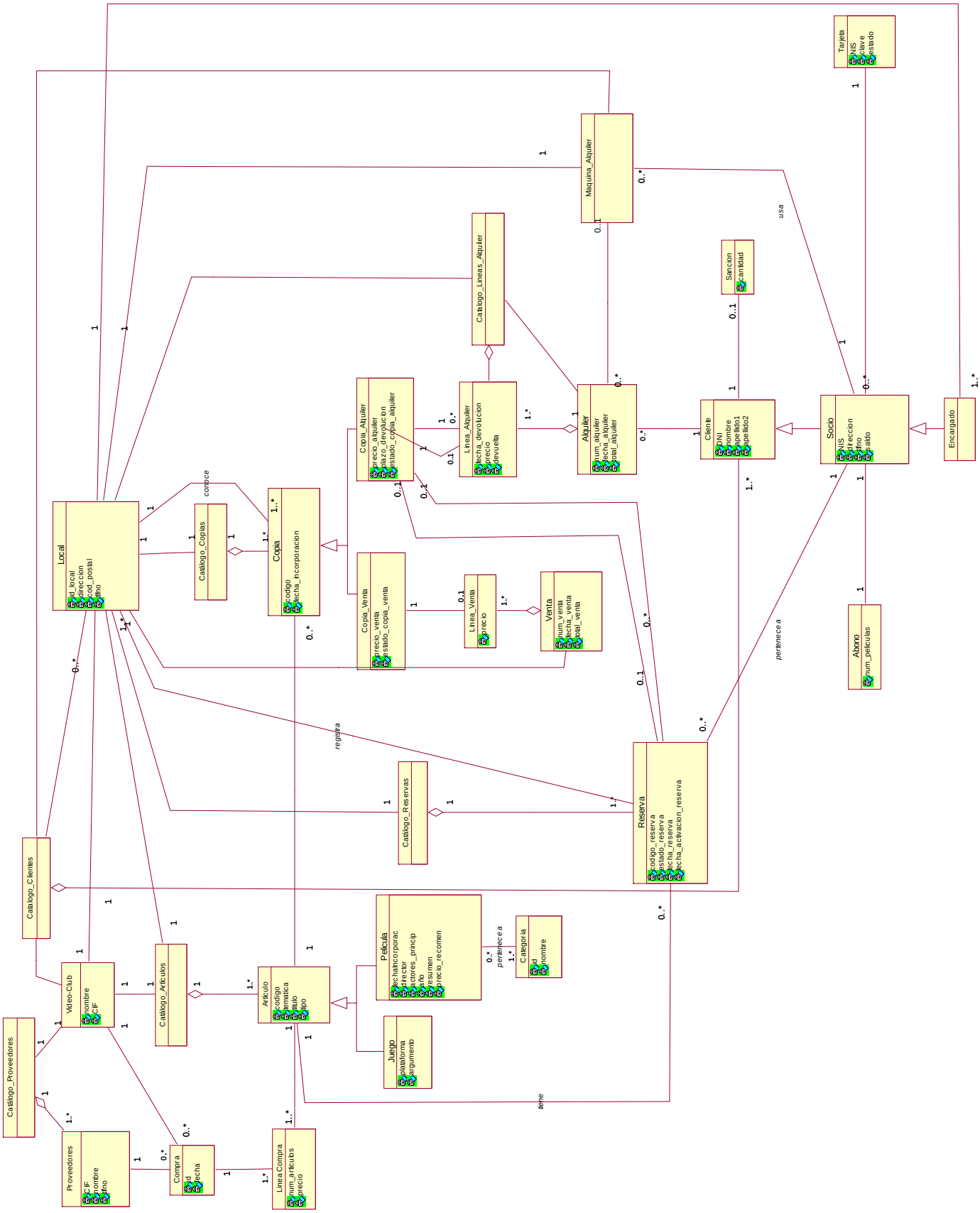
Hemos organizado el documento de manera que cada CdU se engloba en un apartado, en el que se muestra la siguiente información:

- La especificación textual del CdU, siguiendo la plantilla recomendada por Larman para la descripción de CdU[1]. Cualquier suposición al margen del enunciado original de la práctica se incluirá antes de cada especificación.
- Diagrama de secuencia del sistema para ese CdU, en el que, como ya hemos comentado, se muestra los eventos que el actor genera al interactuar con el sistema para ejecutar ese CdU.
- Los contratos de cada una de las operaciones que aparecen en el diagrama de secuencia, junto con sus correspondientes diagramas de colaboracion. Para todos ellos se ha intentado aplicar lo máximo posible los patrones GRASP, logrando así un mejor diseño.

Finalmente se realiza la etapa de generación de Código e Implementación. En esta práctica sólo realizamos la generación de Código de forma automática, pero no se llevó a cabo ninguna implementación concreta, ni por tanto, el proceso de Validación.

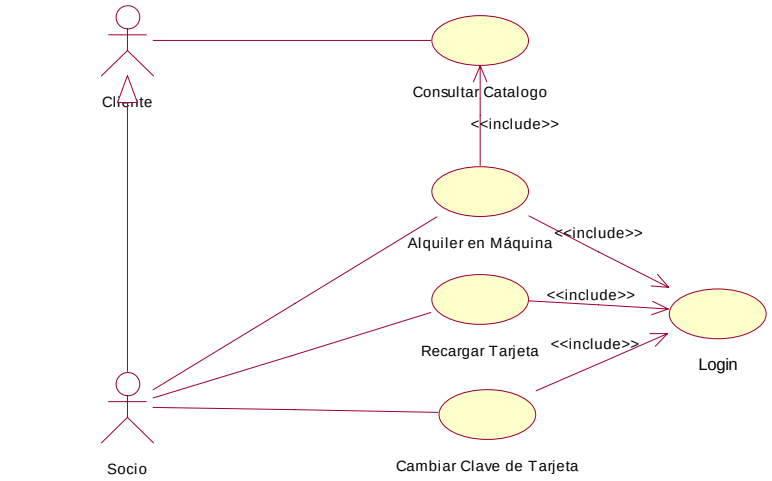
[1] Véase el libro: **C. Larman**, *"Applying UML and patterns"*, 2 edición, Prentice-Hall, 2002.

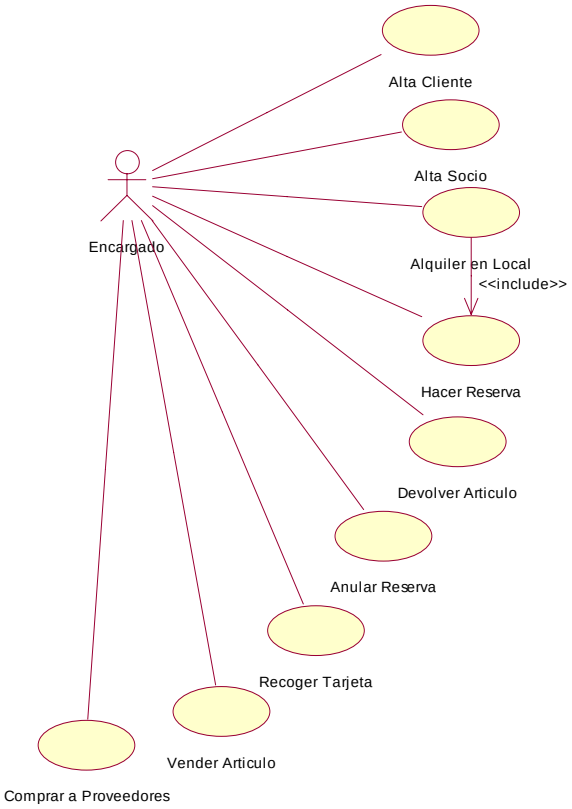
2. Diagrama Conceptual

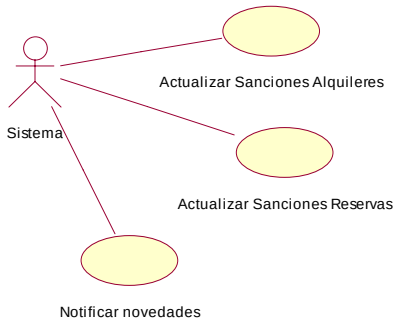


3. Identificación de los Casos de Uso

Se han identificado los siguientes casos de uso del sistema:







4. Alquiler en Local

4.1 Caso de Uso

Para describir este caso de uso hemos realizado ciertas suposiciones al margen de la especificación inicial de requisitos, las cuales exponemos a continuación:

- En el caso de que el cliente tenga una sanción económica anterior, ésta se cargará al alquiler actual.
- Si no hay copias disponibles del artículo que el cliente desea alquilar, el sistema le permitirá automáticamente realizar una reserva del artículo (sólo para socios).
- La fianza para los no socios se deberá abonar para cada artículo alquilado.

Caso de Uso: Alquiler en Local

Objetivo: Realizar el alquiler de una serie de artículos en el Video-Club.

Actores: Encargado(E)

Precondiciones:

Pasos:

1. E: El caso de uso se inicia cuando el CLIENTE llega al puesto del ENCARGADO con los ARTICULOS que desea alquilar.
2. E: Inicia el ALQUILER.
3. E: Introduce los datos de identificación del cliente.
4. S: Valida los datos del cliente.
5. E: Introduce el identificador del ARTICULO.
6. S: Registra la LINEA DE ALQUILER y muestra la descripción del ARTICULO, su precio y el total acumulado.
   El ENCARGADO repite los pasos 5-6 hasta introducir todos los articulos del alquiler.
7. S: Finaliza el ALQUILER.
8. S: Muestra el total del ALQUILER y pide confirmación de pago.
9. E: Confirma el pago cuando el cliente le facilita el dinero.
10. S: Registra el alquiler y extiende el recibo en el que figuran los cobros de los distintos ALQUILERES de los ARTICULOS.
11. E: Extiende los artículos junto al recibo.

Extensiones:

4.1 Los datos introducidos no son correctos.
  4.1.1 S: Indica el error.
  4.1.2 Finalizar caso de uso.
4.2 El cliente tiene pendiente una sanción económica.
  4.2.1 S: Carga la sanción al total del ALQUILER.
  4.2.2 Volver al flujo principal.

5.2 El SOCIO desea realizar el ALQUILER de una RESERVA pendiente de recoger.
  5.2.1 E: Introduce el identificador del artículo reservado.
  5.2.2 S: Registra que la RESERVA ha sido recogida.
  5.2.3 Volver al flujo principal (paso 6).

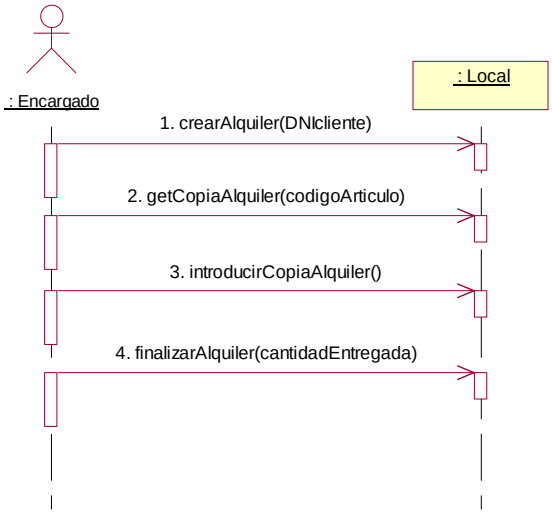
6.1 El cliente no es socio.
  6.1.1 S: Añade al precio asociado a cada artículo la fianza fijada por el local.
  6.1.2 Volver al flujo principal.
6.2 No hay ninguna copia disponible del artículo.
  6.2.1 S: Pide confirmación para reservar el artículo.
  6.2.2 E: Si el cliente lo desea, confirma la reserva.
  6.2.3 *Include* al caso de uso “Hacer Reserva”.

Cuestiones:

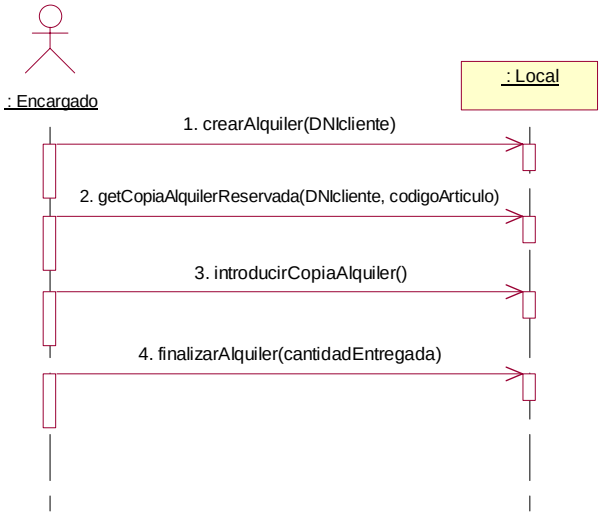
4.2 Diagrama de Secuencia del Sistema

Hemos identificado 2 escenarios diferentes para este CdU. El primero de ellos describe los pasos necesarios para el alquiler normal de un artículo. El segundo muestra la secuencia de acciones para el alquiler de un artículo reseervado. A pesar de ser prácticamente iguales (tan sólo se diferencian en una de las operaciones), hemos separado en 2 escenarios para una mayor claridad.

Escenario 1: Alquiler normal.



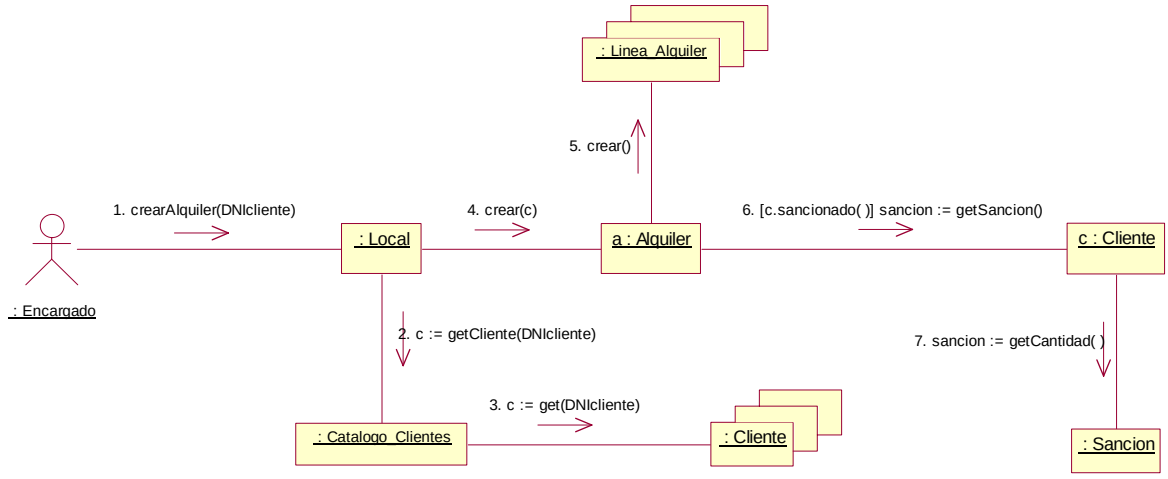
Escenario 2: Alquiler de un articulo reservado.



4.3 Contratos y Colaboraciones de las operaciones

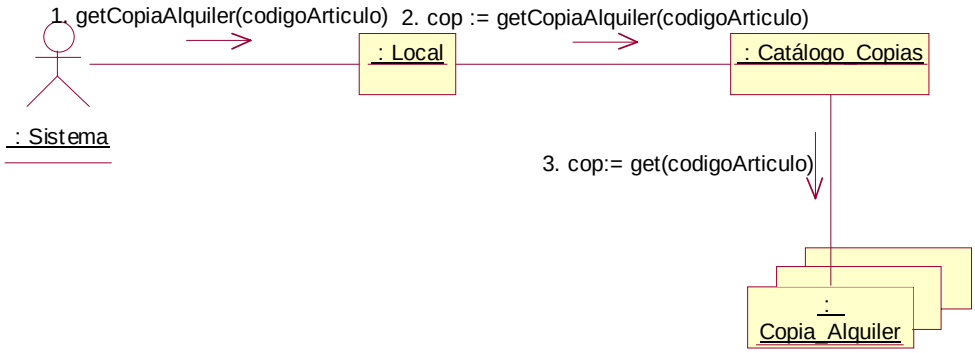
crearAlquiler(DNIcliente)

Operación: crearAlquiler(DNIcliente: tipoDNI)
Responsabilidades: Crear un nuevo alquiler , estableciendo su política de precios, asociándole su cliente y cargándole la sanción que este pudiese tener.
Caso de uso: Alquiler en Local
Controlador: Local
Precondiciones: - El sistema conoce DNIcliente.
Postcondiciones: - Se creó una nueva instancia de ALQUILER ‘a’ - ‘a’ se asoció con el CLIENTE ‘c’ cuyo DNI es ‘DNIcliente’. - Los atributos de ‘a’ se inicializaron. - Se creó el conjunto de LINEAS_ALQUILER del ALQUILER ‘a’. - ‘a’ se asoció con LOCAL (Controlador). - ‘c’ se asoció con LOCAL (Controlador). - Si el CLIENTE ‘c’ tiene una SANCION ‘s’ pendiente se cargará esta al total del alquiler de modo que: a.total_alquiler = a.total_alquiler + s.cantidad.



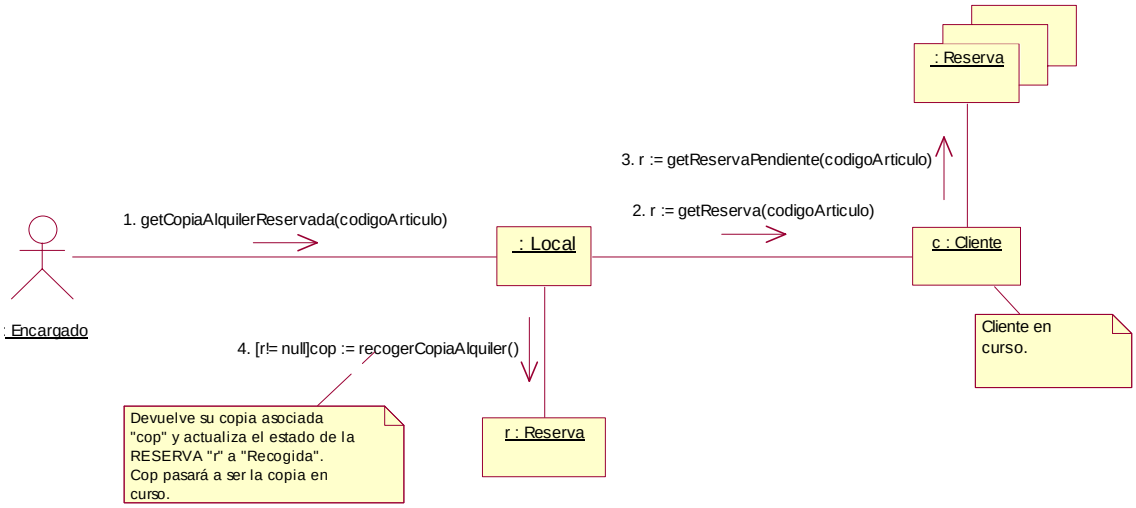
getCopiaAlquiler(codigoArticulo)

Operación: getCopiaAlquiler (codigoCopia: tipoCodigo)
Responsabilidades: Asociar una copia disponible con el Local para tenerla en curso.
Caso de uso: Alquiler en Local
Controlador: Local
Precondiciones: - El sistema conoce ‘codigoCopia’. - La COPIA_ALQUILER con código = ‘codigoCopia’, está disponible.
Postcondiciones: - Se asoció una COPIA_ALQUILER ‘cop’ al Controlador(Local).



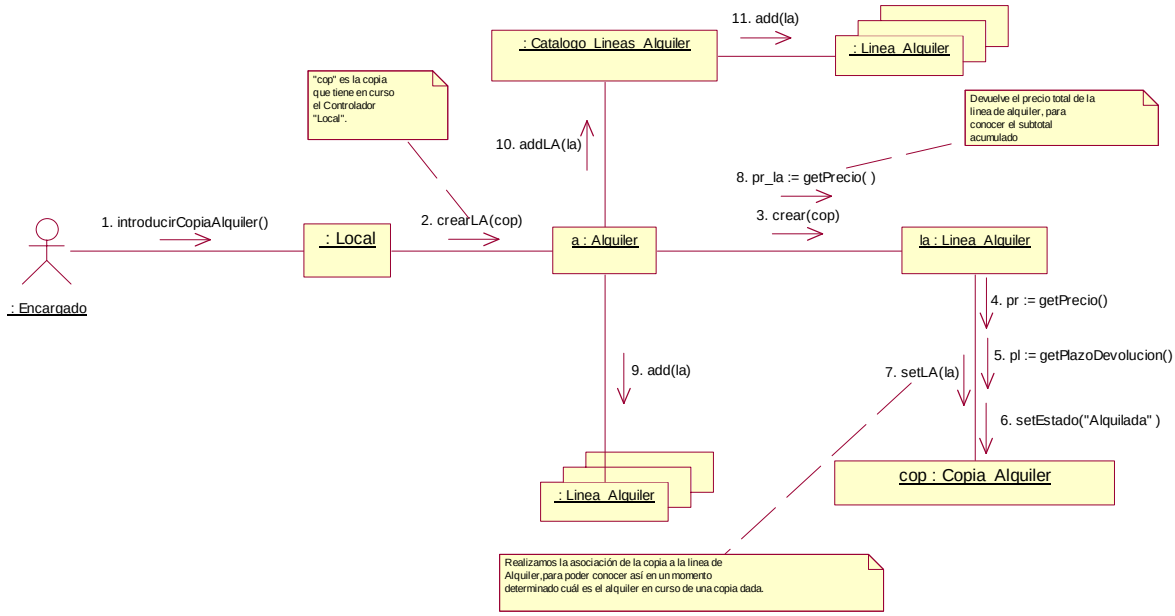
getCopiaAlquilerReservada(codigoArticulo)

Operación: getCopiaAlquilerReservada(codigoArticulo: tipoCodigo)
Responsabilidades: Asociar al Controlador una COPIA que fue reservada por el CLIENTE en curso sobre el ARTICULO con código ‘codigoArticulo’.
Caso de uso: Alquiler en Local
Controlador: Local
Precondiciones: - Se tiene en curso un CLIENTE ‘c’ a quien pertenece el ALQUILER ‘a’ también en proceso. - El sistema conoce ‘codigoArticulo’.
Postcondiciones: - Si el CLIENTE ‘c’ en curso es un socio que tenía una RESERVA ‘r’ de un ARTICULO que se identifica con ‘codigoArticulo’: • r.estado_reserva = “Recogida”. • La COPIA_ALQUILER ‘cop’ asociada a la RESERVA ‘r’ se asoció al Controlador (Local) como copia en curso.



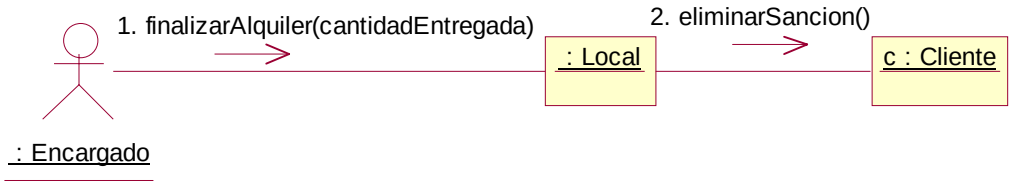
introducirCopiaAlquiler()

Operación: introducirCopiaAlquiler()
Responsabilidades: Introducir una nueva copia al alquiler en curso.
Caso de uso: Alquiler en Local
Controlador: Local
Precondiciones: - Se está procesando un ALQUILER ‘a’. - Se tiene en curso una COPIA_ALQUILER disponible ‘cop’ que se desea añadir al ALQUILER ‘a’ - Se tiene en curso un CLIENTE ‘c’ a quien pertenece el ALQUILER ‘a’ en proceso.
Postcondiciones: - Se creó una nueva instancia de LINEA_ALQUILER ‘la’. - Se asoció ‘la’ al ALQUILER en curso ‘a’. - Se asoció la COPIA_ALQUILER ‘cop’ a ‘la’. - Se asoció ‘la’ a la COPIA_ALQUILER ‘cop’ para conocer la LINEA_ALQUILER en curso de la copia. - la.precio = cop.precio_alquiler. - la.fecha_devolución = fecha actual + cop.plazoDevolucion. - la.devuelta = NO. - a.total_alquiler = a.total_alquiler + la.precio. - Se añadió ‘la’ a CATALOGO_LINEAS_ALQUILER.



finalizarAlquiler(cantidadEntregada: tipoDinero)

Operación: finalizarAlquiler(cantidadEntregada: tipoDinero)
Responsabilidades: Finalizar el alquiler actual una vez que el cliente facilita el dinero.
Caso de uso: Alquiler en Local
Controlador: Local (Sistema)
Precondiciones: - Se está procesando un ALQUILER ‘a’. - cantidadEntregada >= a.total_alquiler
Postcondiciones: - La instancia SANCION asociada al cliente fue eliminada (si existía).



5. Alquiler en Máquina

5.1 Caso de Uso

Antes de describir el CdU, explicaremos brevemente las suposiciones realizadas:

- A diferencia del alquiler en el local, el alquiler en máquina consta tan sólo de un artículo, es decir, está formado de una sóla línea de alquiler.
- La sanción del socio, si tiene, se cargará al crear el alquiler (igual que en el alquiler en local)
- El socio puede consultar el catálogo de artículos para seleccionar el artículo a alquilar.
- El socio puede recoger también una película reservada.

Caso de Uso: Alquiler en Máquina

Objetivo: Realizar el alquiler de un artículo en una de las máquinas automática que ofrece el Video-Club

Actores: Socio (Sc)

Precondiciones:

Pasos:

1. Sc: El caso de uso se inicia cuando el SOCIO desea alquilar un artículo en la MAQUINA DE ALQUILER que dispone el local.
2. *Include* a Caso de uso “Login”.
3. Sc: Inicia el ALQUILER.
4. Sc: Introduce identificador del ARTICULO.
5. S: Registra la LINEA DE ALQUILER y muestra la descripción del ARTICULO junto con su precio.
6. Sc: Finaliza (confirma) el ALQUILER.
7. S: Registra el alquiler y extiende el recibo.
8. S: Decrementa el saldo del SOCIO.
9. S: Extiende los artículos alquilados.

Extensiones:

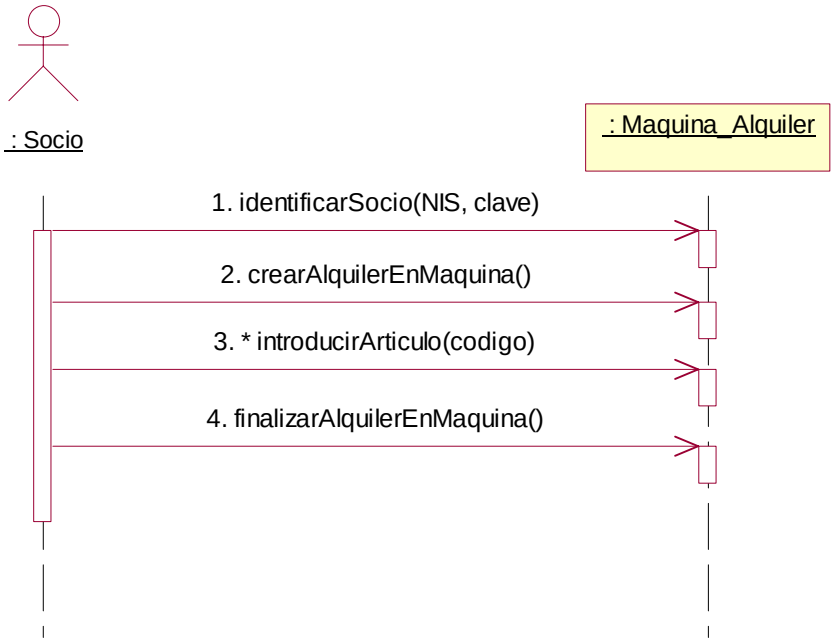
3.1 El SOCIO tiene pendiente una sanción económica.
  3.1.1 S: Carga la sanción al total del ALQUILER.
  3.1.2 Vorver al flujo principal (paso 4).

4.1 El SOCIO desea consultar el catalogo para seleccionar un ARTÍCULO.
  4.1.1 *Include* al caso de uso “Consultar Catálogo”.
4.2 El SOCIO desea realizar el ALQUILER de una RESERVA pendiente de recoger.
  4.2.1 Sc: Introduce el identificador del artículo reservado.
  4.2.2 S: Registra que la reserva ha sido recogida.
  4.2.3 Volver al flujo principal (paso 5).

7.1 No hay saldo suficiente en la tarjeta para realizar el alquiler.
  7.1.1 S: Indica al SOCIO de que no dispone de saldo suficiente.
  7.1.1 S: Anula el ALQUILER.

Cuestiones:

5.2 Diagrama de Secuencia del Sistema



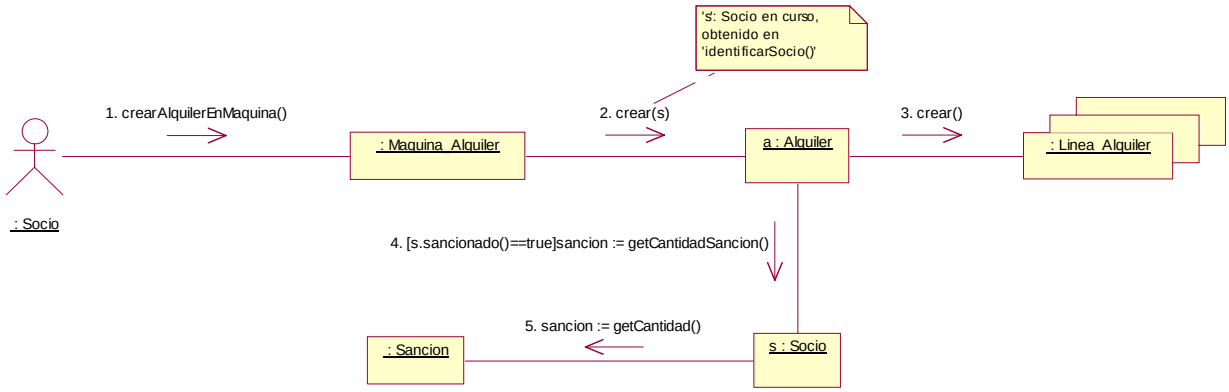
5.3 Contratos y Colaboraciones de las Operaciones

identificarSocio(id_tarjeta, clave)

Operación: identificarSocio(id_tarjeta, clave)
... (Misma operación que en el CdU “Login”) ...

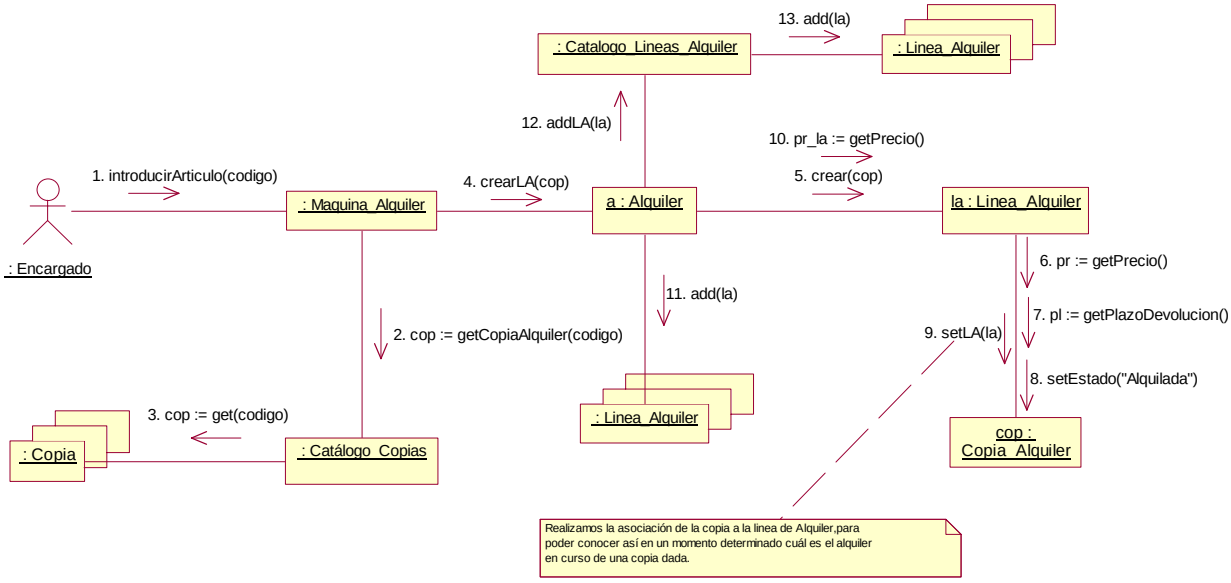
crearAlquilerEnMaquina()

Operación: crearAlquilerEnMaquina()
Responsabilidades: crear un nuevo alquiler en la Maquina de Alquiler
Caso de uso: Alquiler en Maquina
Controlador: Maquina de Alquiler (Sistema)
Precondiciones: - Un SOCIO ‘s’ se ha identificado con éxito en la MAQUINA_ALQUILER (mediante su tarjeta).
Postcondiciones: - Se creó una nueva instancia de ALQUILER ‘a’. - ‘a’ se asoció con el SOCIO ‘s’. - Los atributos de ‘a’ se inicializaron. - Se creó el conjunto de LINEAS_ALQUILER del ALQUILER ‘a’. - ‘a’ se asoció con la MAQUINA_ALQUILER(Controlador). - Si el CLIENTE ‘c’ tiene una SANCION ‘s’ pendiente se cargará esta al total del alquiler de modo que: a.total_alquiler = a.total_alquiler + s.cantidad.



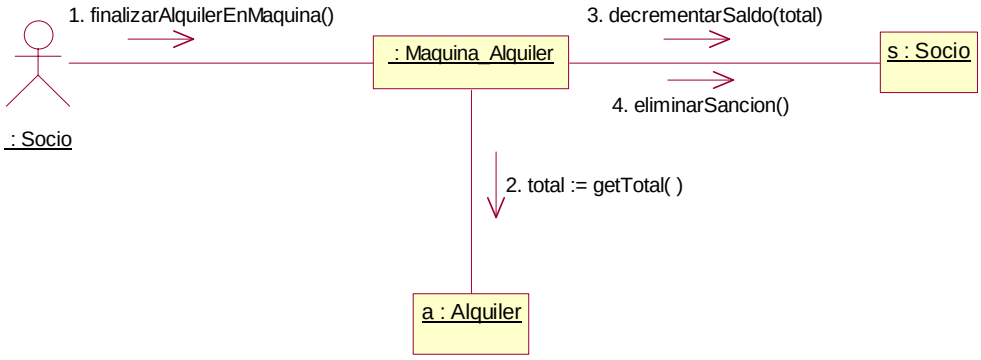
introducirArticulo(codigo)

Operación: introducirArticulo(codigo: tipoCodigo)
Responsabilidades: Introducir un nuevo artículo al alquiler en curso.
Caso de uso: Alquiler en Local
Controlador: Local (Sistema)
Precondiciones: - Se está procesando un ALQUILER ‘a’. - El sistema conoce ‘codigo’. - La copia con código = ‘codigo’,está disponible.
Postcondiciones: - Se creó una nueva instancia de LINEA_ALQUILER ‘la’. - Se asoció ‘la’ al ALQUILER en curso ‘a’. - Se asoció ‘la’ a la COPIA_ALQUILER ‘cop’ cuyo código es ‘codigo’. - la.precio = cop.precio_alquiler. - la.fecha_devolución = fecha actual + cop.plazoDevolucion. - la.devuelta = NO. - Se asoció la COPIA_ALQUILER ‘cop’ a ‘la’ para conocer la LINEA_ALQUILER en curso de la copia. - a.total_alquiler = a.total_alquiler + la.precio. - Se añadió ‘la’ a CATALOGO_LINEAS_ALQUILER.



finalizarAlquilarEnMaquina()

Operación: finalizarAlquilerEnMaquina()
Responsabilidades: Finalizar el alquiler actual en la máquina y decrementar el saldo del socio.
Caso de uso: Alquiler en Maquina
Controlador: Maquina de Alquiler
Precondiciones: - Se está procesando un ALQUILER ‘a’ por parte de un SOCIO ‘s’. - El saldo del SOCIO es mayor o igual al total del alquiler (s.saldo >= a.total_alquiler)
Postcondiciones: - s.saldo = s.saldo – a.getTotal(). - La instancia SANCION asociada al cliente fue eliminada (si existía).

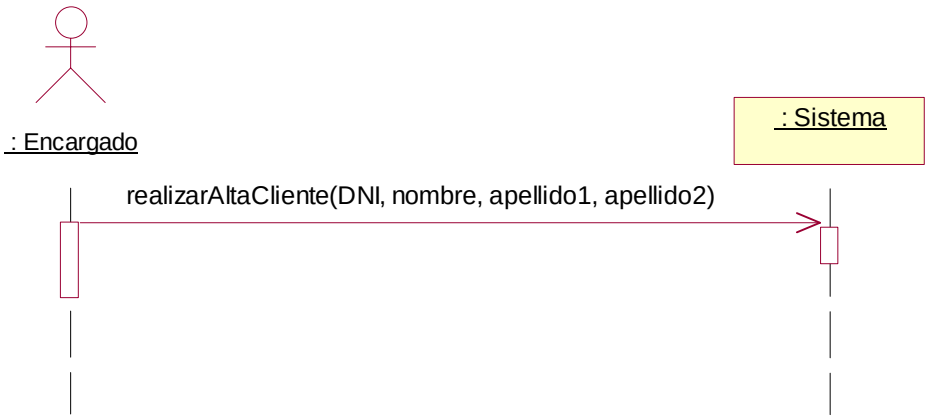


6. Alta Cliente

6.1 Caso de Uso

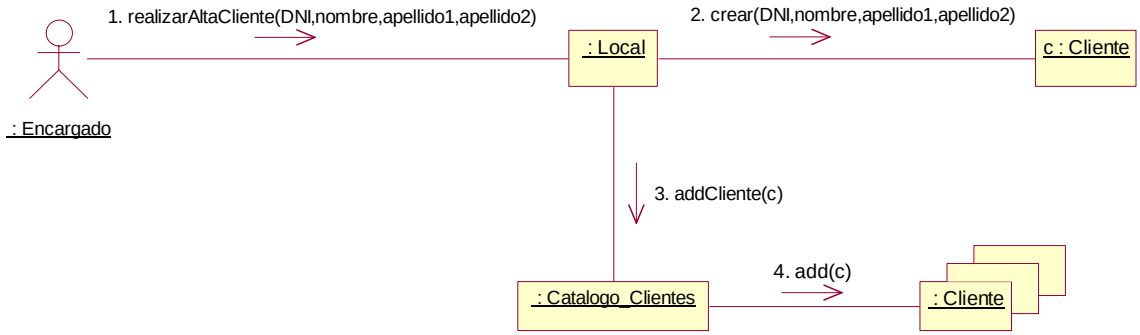
Caso de Uso: Alta Cliente
Objetivo: Registrar un nuevo cliente en la base de datos del Video-Club
Actores: Encargado (E)
Precondiciones:
Pasos: 1. El caso de uso se inicia cuando un cliente quiere usar por primera vez los servicios del Video-Club. 2. E: Introduce los datos de identificación del CLIENTE (DNI, nombre, apellidos) 3. S: Valida los datos del CLIENTE. 4. S: Registra al CLIENTE.
Extensiones: 3.1 Los datos introducidos son incorrectos. 3.1.1 S: Indica el error. 3.1.2 Finalizar el caso de uso.
Cuestiones:

6.2 Diagrama de Secuencia del Sistema



6.3 Contratos y Colaboraciones de la operación

Operación: realizarAltaCliente(DNI: tipoDNI, nombre: String, apellido1, apellido2: String)
Responsabilidades: Dar de alta a un cliente en la base de datos del Videoclub
Caso de uso: Alta Cliente
Controlador: Sistema
Precondiciones: - Los datos del cliente son válidos
Postcondiciones: - Se creó una nueva instancia de CLIENTE ‘c’. - ‘c’ se asoció (añadió) al CATALOGO_CLIENTES del videoclub. - c.DNI = DNI. - c.nombre = nombre. - c.apellido1 = apellido1. - c.apellido2 = apellido2.

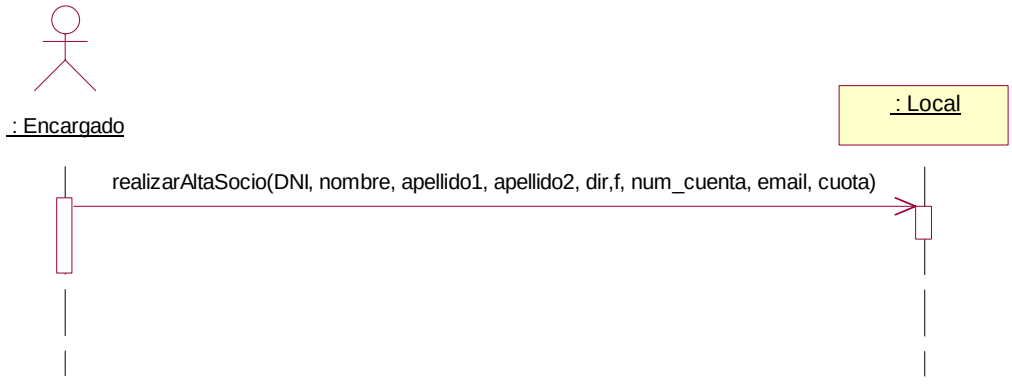


7. Alta Socio

7.1 Caso de Uso

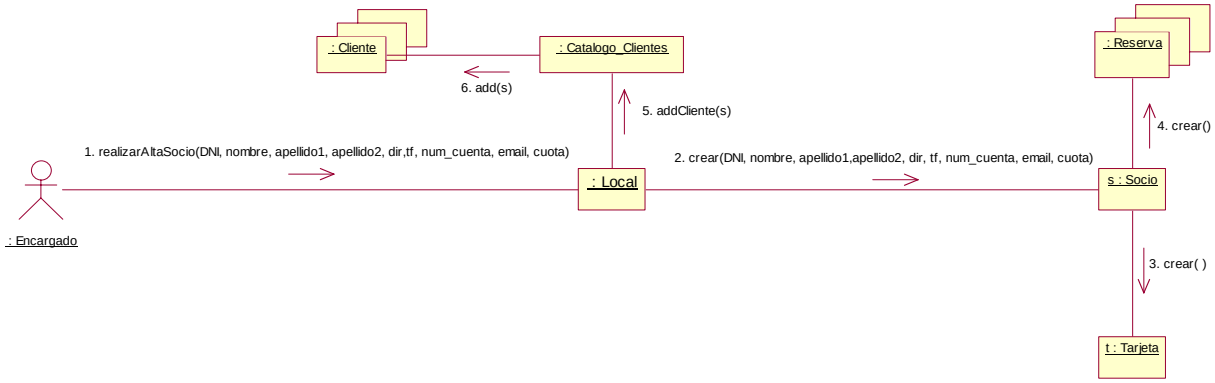
Caso de Uso: Alta Socio
Objetivo: Registrar un nuevo cliente como socio en la base de datos del Video-Club
Actores: Encargado (E)
Precondiciones:
Pasos: 1. El caso de uso se inicia cuando un cliente desea hacerse socio del videoclub. 2. E: Introduce los datos de identificación del SOCIO (DNI, nombre, apellidos, dirección, teléfono y, opcionalmente, número de cuenta y e-mail). 3. E: Introduce la cuota inicial desembolsada por el SOCIO, la cual será mayor o igual que la cuota mínima exigida. 4. S: Valida los datos introducidos. 5. S: Establece el saldo inicial de la TARJETA del socio a la cuota inicial desembolsada. La tarjeta inicialmente no estará disponible (se le entregará al socio más adelante). 6. S: Registra el alta del SOCIO y muestra su NIS (Número de Identificación de Socio). 7. E: Facilita al SOCIO su NIS.
Extensiones: 1.1 El actor de este caso de uso puede ser también el CLIENTE (en lugar del ENCARGADO) si el alta se realiza desde la web. 4.1 Los datos introducidos son incorrectos. 4.1.1 S: Indica error. 4.1.2 Finalizar el caso de uso.
Cuestiones:

7.2 Diagrama de Secuencia del Sistema



7.3 Contratos y Colaboraciones de la operación

Operación: realizarAltaSocio(DNI: tipoDNI, nombre,apellido1,apellido2: String, dir: String, tf: tipoTf, num_cuenta: tipoNumCuenta, email: String, cuota: Number)
Responsabilidades: Realizar el alta de un socio en el Videoclub.
Caso de uso: Alta Cliente
Controlador: Sistema
Precondiciones: - Los datos del cliente son válidos. - ‘cuota’ es mayor que la cuota inicial impuesta por el local.
Postcondiciones: - Se creó una nueva instancia de SOCIO ‘s’. - ‘s’ se asoció (añadió) al CATALOGO_CLIENTES del videoclub. - s.DNI = DNI. - s.nombre = nombre. - s.apellido1 = apellido1. - s.apellido2 = apellido2. - s.direccion = dir. - s.telefono = tf. - s.num_cuenta = num_cuenta. - s.email = email. - s.saldo = cuota. - Se creó la lista de RESERVAS del SOCIO ‘s’. - Se creó una nueva instancia de TARJETA ‘t’. - ‘t’ se asoció con el SOCIO ‘s’. - t.estado = “No disponible”. - El resto de atributos de ‘t’ se inicializaron a sus valores por defecto.



8. Consultar Catálogo

8.1 Caso de Uso

Para este caso de uso tan sólo se ha realizado la especificación textual, y de manera muy general. No se han incluido los contratos/colaboraciones porque en realidad mediante este caso de uso no se crea, se destruye o se modifica ningún objeto o asociación, tan sólo permite ver al cliente el catálogo de artículos del videoclub.

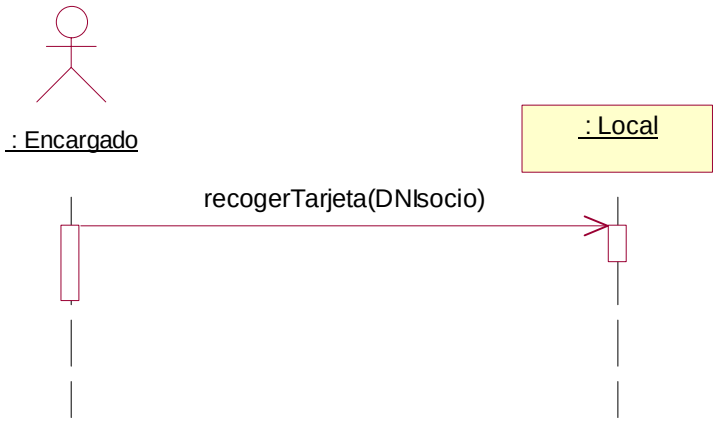
Caso de Uso: Consultar Catálogo
Objetivo: Llevar a cabo una consulta del catálogo de artículos del Video-Club.
Actores: Cliente (C)
Precondiciones:
Pasos: 1. El caso de uso se inicia cuando el CLIENTE, en un terminal de algún local o vía web, desea hacer una consulta sobre el CATÁLOGO de ARTICULOS. 2. C: introduce uno o varios campos de búsqueda. 3. S: muestra la relación de artículos encontrados en el CATALOGO. 4. El cliente selecciona uno de los artículos relacionados para ver su especificación detallada. *El CLIENTE repite los pasos 3-4 hasta acabar la consulta.* 5. Finalizar caso de uso.
Extensiones: 4.1 El CLIENTE desea ver la reproducción multimedia disponible para este ARTÍCULO. 4.1.1 S: Muestra la reproducción asociada al ARTICULO. 4.1.2 Volver al flujo principal de ejecución.
Cuestiones:

9. Recoger Tarjeta

9.1 Caso de Uso

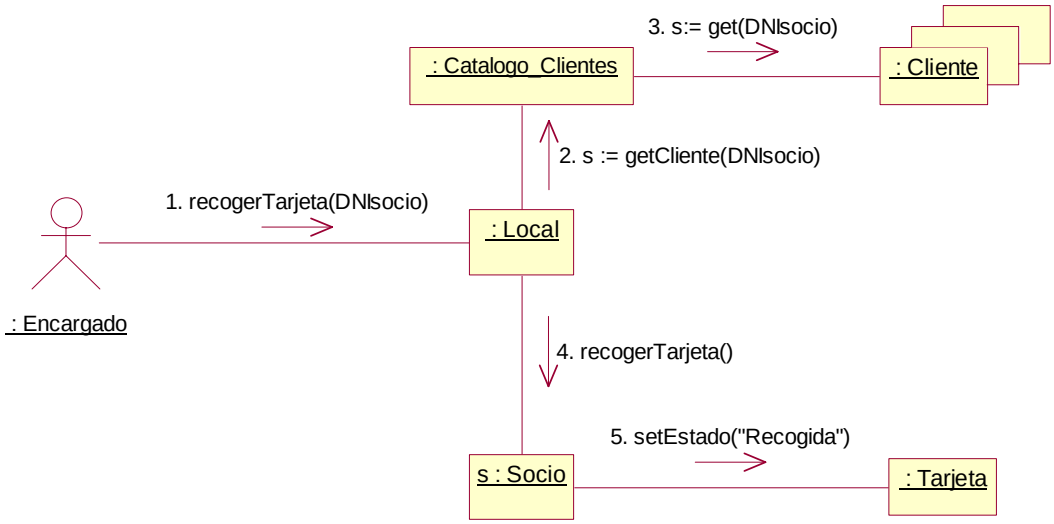
Caso de Uso: Recoger Tarjeta
Objetivo: Recoger la tarjeta de socio para el alquiler de artículos en las máquinas automáticas.
Actores: Encargado(E)
Precondiciones: El socio ya fue dado de alta.
Pasos: 1. El caso de uso se inicia cuando el SOCIO va a recoger su TARJETA al local después de haberse dado de alta. 2. S: Solicita los datos de identificación del SOCIO. 3. E: Facilita los datos del SOCIO. 4. S: Valida los datos introducidos. 5. S: Registra el cambio de estado de la TARJETA (recogida). 6. E: Entrega la TARJETA al SOCIO y le facilita su clave inicial.
Extensiones: 4.1 Los datos de identificación no son correctos. 4.1.1 Indicar error de identificación. 4.1.2 Volver al flujo principal (paso 2). 5.1 La tarjeta todavía no está disponible. 5.1.1 E: Indica este hecho al cliente 5.1.2 Finalizar caso de uso
Cuestiones:

9.2 Diagrama de Secuencia del Sistema



9.3 Contratos y Colaboraciones de la operación

Operación: recogerTarjeta(DNIsocio: tipoDNI)
Responsabilidades: Cambiar el estado de la tarjeta a “Recogido”.
Caso de uso: Recoger Tarjeta.
Controlador: Local (Sistema)
Precondiciones: - El sistema conoce ‘DNIsocio’.
Postcondiciones: - t.estado = “Recogida”, donde ‘t’ es la TARJETA asociada al SOCIO con DNI = ‘DNIsocio’.



10. Devolver Artículo

10.1 Caso de Uso

Estas son las suposiciones realizadas para este CdU:

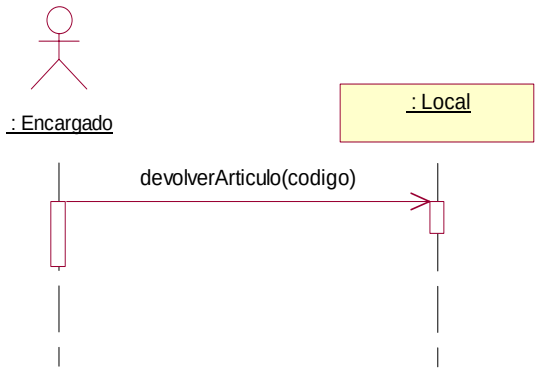
- Cuando un cliente devuelve un artículo, se comprobará si hay reservas pendientes para ese artículo, asignándole la copia devuelta a la más antigua de todas ellas.
- En el momento de la devolución el sistema comprobará el estado de la copia, y podrá cargar una sanción al cliente si ha devuelto la copia defectuosa o rota.
- La sanción por retraso en la devolución no se comprueba en este CdU, sino que es comprobado diariamente por el sistema (véase CdU “Actualizar Sanciones Alquileres”).

Caso de Uso: Devolver Artículo
Objetivo: Realizar la devolución de un artículo.
Actores: Encargado (E)
Precondiciones:
Pasos: 1. El caso de uso se inicia cuando un cliente accede al video-club con un artículo para realizar su devolución, ya sea mediante la máquina o en el local. 2. E: Facilita la copia. 3. S: Registra la devolución. 4. S: Comprueba si hay RESERVAS pendientes para el ARTICULO, asignando la COPIA a la primera reserva en espera.
Extensiones: 1.1 El actor de este caso de uso también puede ser un SOCIO que desea devolver un artículo mediante la MAQUINA DE ALQUILER. 1.1.1 *Include* al caso de uso “Login”. 3.1 La copia está defectuosa o rota 3.1.1 S: Informa del defecto en la copia y de la sanción impuesta al cliente. 3.1.2 S: Carga la sanción al cliente. 3.1.3 Volver al flujo principal.
Cuestiones:

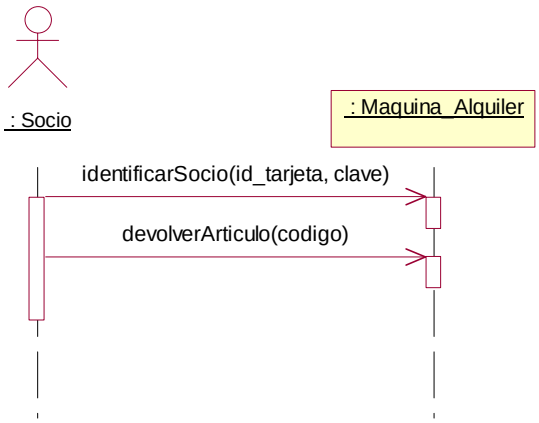
10.2 Diagrama de Secuencia

En este caso se pueden dar 2 escenarios diferentes según donde se realice la devolución, pero lo único que distingue a estos dos escenarios es la forma de acceder al sistema: en el local accedemos mediante la figura del *Encargado*, y en la máquina el propio *Socio* interactúa con el sistema tras la previa identificación.

Escenario 1: Devolución en Local.



Escenario 2: Devolución en Máquina.



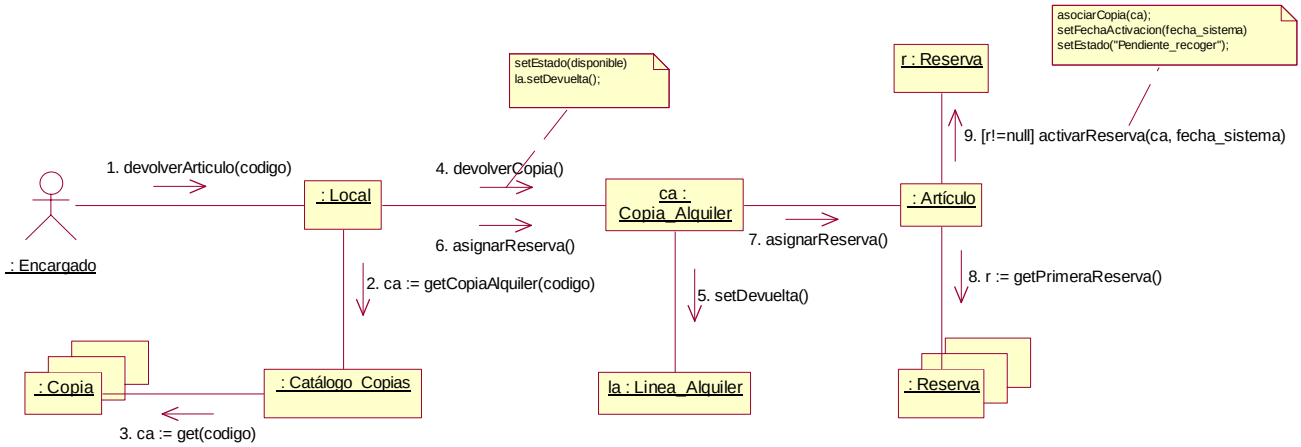
10.3 Contratos y Colaboraciones de las operaciónes

identificarSocio(id_tarjeta,clave)

Operación: identificarSocio(id_tarjeta,clave)
... (Misma operación que en “Login”) ...

devolverArticulo(codigoArticulo)

Operación: devolverArticulo(codigo: tipoCodigo)
Responsabilidades: Realizar la devolución de un artículo alquilado.
Caso de uso: Devolver Artículo
Controlador: Local (Sistema)
Precondiciones: - El sistema conoce ‘código’.
Postcondiciones: - ca.estado_copia_alquiler = “Disponible”, donde ‘ca’ es la COPIA_ALQUILER cuyo código es ‘código’. - la.devuelta = SI, donde ‘la’ es la LINEA_ALQUILER asociada a ‘ca’. - Si existía alguna RESERVA en espera para el ARTICULO, entonces: • ‘ca’ se asoció con la RESERVA ‘r’. • r.estado_reserva = “Pendiente_recoger”. • r.fecha_reserva_activa = fecha actual del sistema. • ca.estado_copia = “Reservada”.

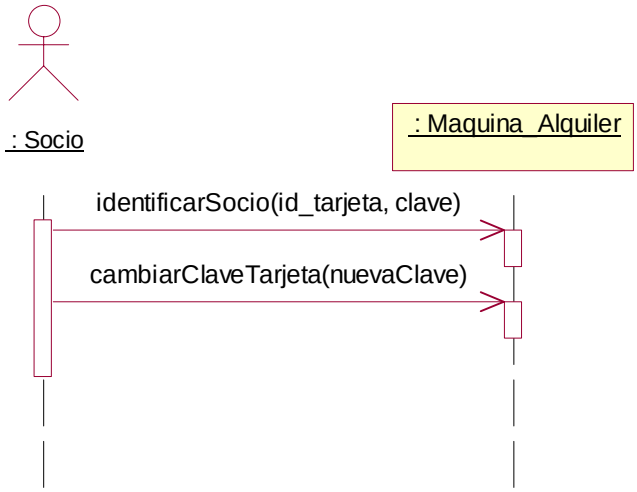


11. Cambiar Clave de Tarjeta

11.1 Caso de Uso

Caso de Uso: Cambiar Clave de Tarjeta
Objetivo: Cambiar la clave de la tarjeta de un socio.
Actores: Socio(Sc)
Precondiciones:
Pasos: 1. El caso de uso se inicia cuando el SOCIO desea cambiar la clave de su tarjeta. 2. *Include* caso de uso “Login”. 3. S: Solicita la nueva clave. 4. Sc: Introduce su nueva clave. 5. S: Registra la nueva clave.
Extensiones:
Cuestiones:

11.2 Diagrama de Secuencia del Sistema



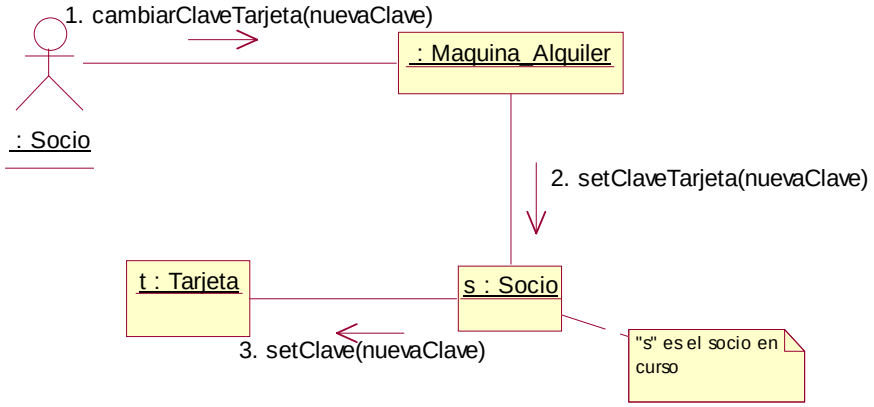
11.3 Contratos y Colaboraciones de las operaciones

introducirTarjeta()

Operación: introducirTarjeta()
... (Misma operación que en “Login”) ...

cambiarClaveTarjeta(nuevaClave)

Operación: cambiarClaveTarjeta(nuevaClave: tipoClave)
Responsabilidades: Cambiar la clave de la tarjeta del socio.
Caso de uso: Cambiar Clave de Tarjeta
Controlador: Maquina de Alquiler (Sistema)
Precondiciones: - Un SOCIO ‘s’ se ha identificado con éxito en la MAQUINA_ALQUILER (mediante su tarjeta).
Postcondiciones: - t.clave = nuevaClave, donde ‘t’ es la TARJETA asociada al SOCIO ‘s’.

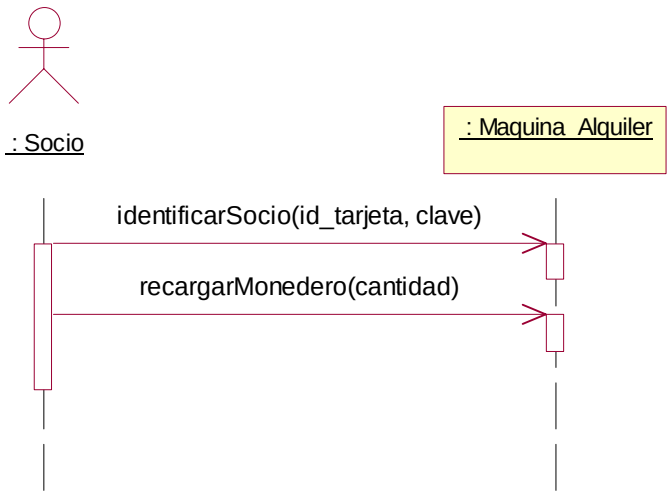


12. Recargar Tarjeta

12.1 Caso de Uso

Caso de Uso: Recargar Tarjeta
Objetivo: Recargar el saldo del socio mediante el uso de la máquina de alquiler
Actores: Socio (Sc)
Precondiciones:
Pasos: 1. El caso de uso se inicia cuando el SOCIO desea recargar su tarjeta en la máquina automática. 2. *Include* caso de uso “Login”. 3. S: Solicita el dinero. 4. Sc: Introduce el dinero. 5. Sc: Finaliza el ingreso. 6. S: Registra el ingreso.
Extensiones: 4.a El SOCIO desea realizar el ingreso en efectivo. 4.a.1 Sc: Introduce el dinero en efectivo. 4.a.2 Volver al paso 4. 4.b El SOCIO desea realizar el ingreso mediante una transferencia bancaria. 4.b.1 Sc: Introduce la cantidad a transferir. 4.b.2 S: Valida la transferencia. 4.b.3 S: Realiza la transferencia con la entidad bancaria. 4.b.4 Volver al paso 4.
Cuestiones:

12.2 Diagrama de Secuencia del Sistema



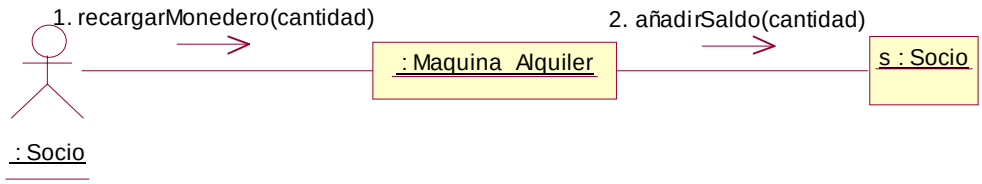
12.3 Contratos y Colaboraciones de las operaciones

identificarSocio()

Operación: identificarSocio()
... (Misma operación que en “Login”) ...

recargarMonedero()

Operación: recargarMonedero(cantidad)
Responsabilidades: Recargar el saldo disponible por el socio.
Caso de uso: Recargar tarjeta.
Controlador: Maquina de Alquiler.
Precondiciones: - Un SOCIO ‘s’ se ha identificado con éxito en la MAQUINA_ALQUILER (mediante su tarjeta).
Postcondiciones: - s.saldo = s.saldo + cantidad.

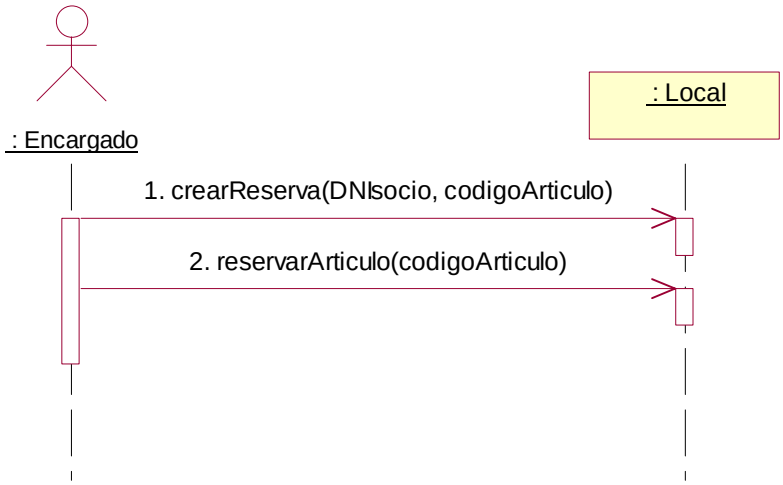


13. Hacer Reserva

13.1 Caso de Uso

Caso de Uso: Hacer Reserva
Objetivo: Realizar una reserva en el local o vía web
Actores: Encargado (E)
Precondiciones:
Pasos: 1. El caso de uso se inicia cuando el socio desea realizar una RESERVA de un ARTICULO. 2. E: Introduce el identificador del SOCIO. 3. S: Valida los datos. 4. E: Introduce el identificador del ARTICULO. 5. S: Registra la RESERVA del ARTICULO. 6. S: Una de las COPIAS disponibles del ARTICULO pasa a estar reservada (durante el tiempo estimado por la política del local). 7. S: Muestra el plazo válido de la RESERVA.
Extensiones: 1.1 El actor principal de este caso de uso también puede ser un SOCIO, si la RESERVA se realiza vía web. 6.1 Ninguna de las COPIAS del ARTICULO está disponible 6.1.1 S: Añade la RESERVA a la lista de espera del ARTICULO. 6.1.2 S: Informa de la situación de la RESERVA (“En espera”). 6.1.3 Finalizar caso de uso.
Cuestiones:

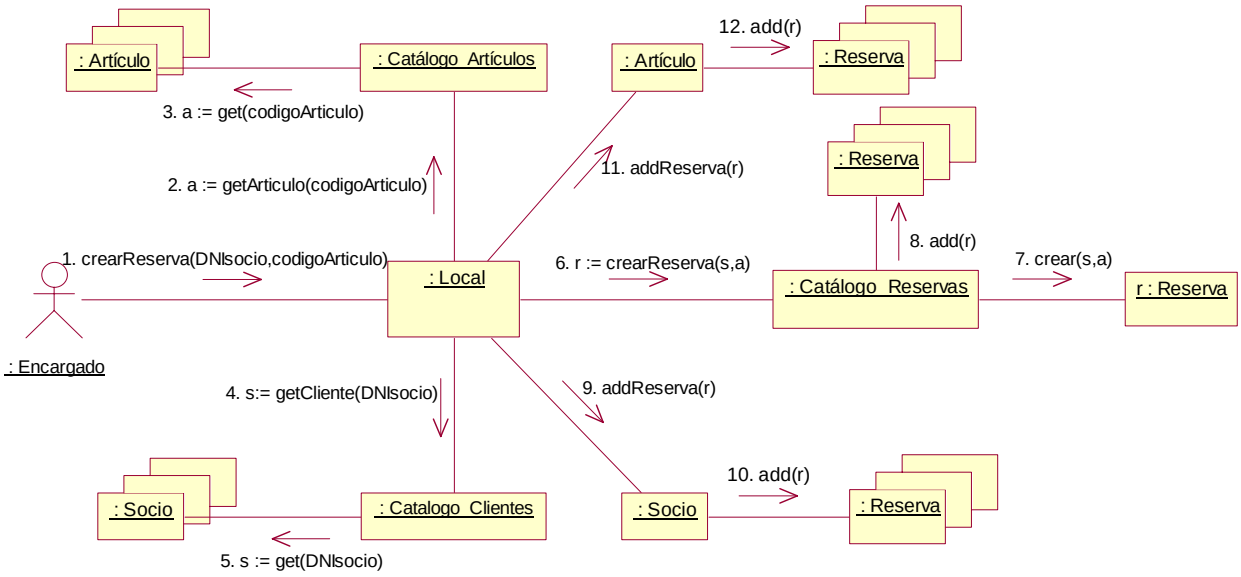
13.2 Diagrama de Secuencia del Sistema



13.3 Contratos y Colaboraciones de las operaciones

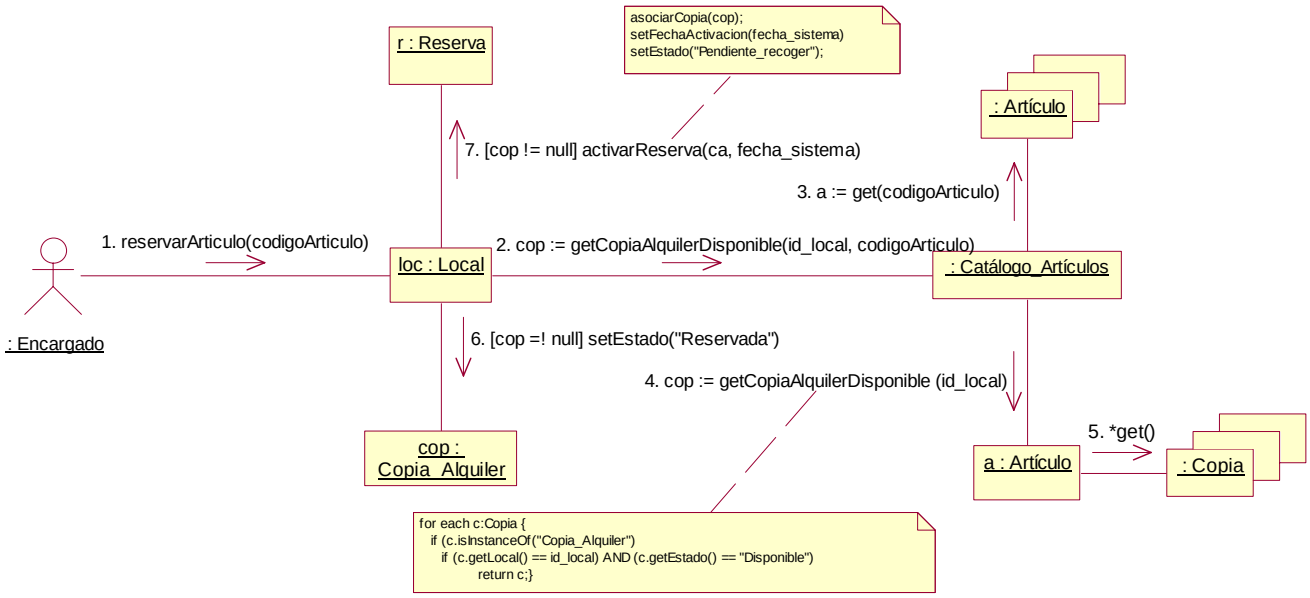
crearReserva (DNIsocio, codigoArticulo)

Operación: crearReserva (DNIsocio: tipoDNI, codigoArticulo: tipoCodigo)
Responsabilidades: Crear una nueva reserva de un artículo por parte de un socio.
Caso de uso: Hacer Reserva
Controlador: Sistema
Precondiciones: - El sistema conoce ‘DNIsocio’ y ‘codigoArticulo’
Postcondiciones: - Se creó una instancia de RESERVA ‘r’. - Se añadió ‘r’ a la lista de RESERVAS del SOCIO ‘s’ con DNI ‘DNIsocio’. - Se añadió ‘r’ a la lista de RESERVAS del ARTICULO ‘a’ cuyo código es ‘codigoArticulo’. - r.fecha_reserva = fecha actual del sistema. - r.codigo_reserva = siguiente código. - r.estado_reserva = ‘En_espera’. - Se añadió ‘r’ al CATÁLOGO_DE_RESERVAS. - ‘r’ se asoció con LOCAL (Controlador)



reservarArticulo(codigoArticulo)

Operación: reservarArticulo(codigoArticulo:tipoCodigo)
Responsabilidades: Formaliza la reserva de un artículo por parte de un socio.
Caso de uso: Hacer Reserva
Controlador: Sistema
Precondiciones: - Se está procesando una RESERVA ‘r’ - El sistema conoce ‘codigoArticulo’.
Postcondiciones: - Si existe una COPIA_ALQUILER ‘cop’ disponible del artículo que se identifica con ‘codigoArticulo’, entonces: • Se asoció la RESERVA ‘r’ con la COPIA_ALQUILER ‘cop’. • r.estado_reserva = “Pendiente_Recoger” • r.fecha_reserva_activa = fecha actual del sistema. • cop.estado_copia = “Reservada”



14. Anular Reserva

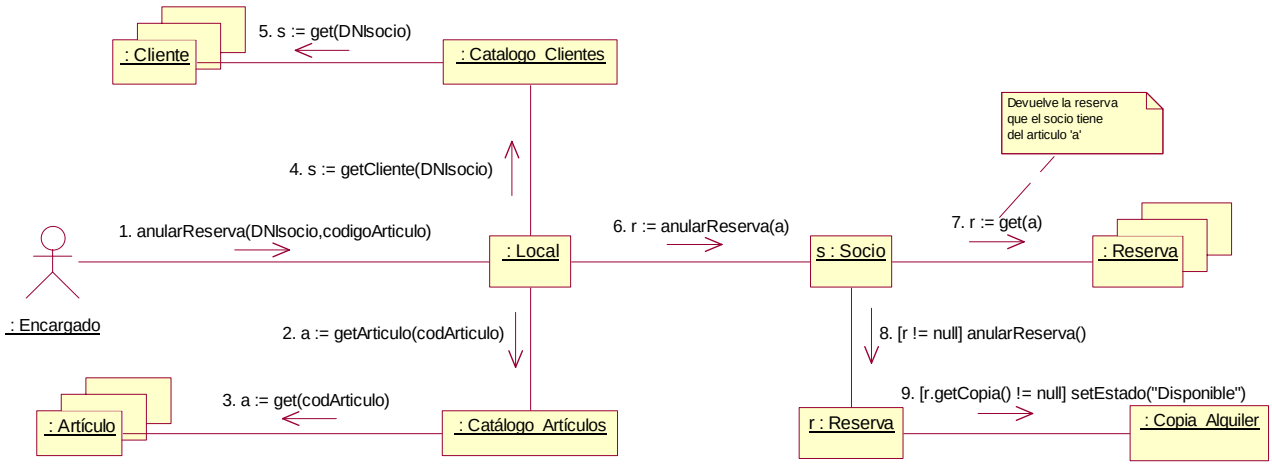
14.1 Caso de Uso

Caso de Uso: Anular Reserva
Objetivo: Realizar la anulación de una reserva hecha con anterioridad, vía web o en el local.
Actores: Encargado (E)
Precondiciones:
Pasos: 1. El caso de uso se inicia cuando el SOCIO accede al puesto del encargado de un local para anular una reserva activa o en espera. 2. S: Solicita datos de identificación del SOCIO. 3. E: Facilita los datos de identificación del SOCIO. 4. S: Valida los datos del SOCIO. 5. E: Introduce el artículo. 6. S: Realiza el registro de la anulación de la reserva.
Extensiones: 1. El caso de uso puede ser realizado también por un SOCIO, cuando se hace vía web.
Cuestiones:

14.2 Diagrama de Secuencia del Sistema

14.3 Contratos y Colaboraciones de la operación

Operación: anularReserva(DNIsocio: tipoDNI, codigoArticulo: tipoCodigo)
Responsabilidades: Anular una reserva de un artículo.
Caso de uso: Anular reserva
Controlador: Local
Precondiciones: - El sistema conoce ‘DNIsocio’ y ‘codigoArticulo’ - Existe una RESERVA ‘r’ por parte del SOCIO con DNI ‘DNIsocio’ del ARTICULO con codigo ‘codigoArticulo’.
Postcondiciones: - Si la RESERVA ‘r’ tiene asociada una COPIA_ALQUILER ‘c’, entonces • c.estado_copia = “Disponible”. - r.estado_reserva = ‘Anulada’.

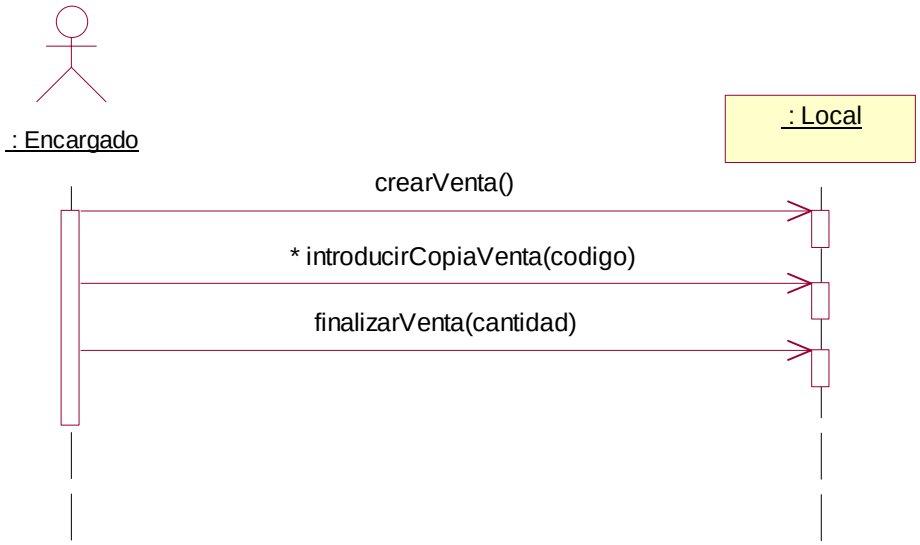


15. Vender Articulo

15.1 Caso de Uso

Caso de Uso: Vender Articulo
Objetivo: Relizar la venta de un artículo en algún local del Video-Club
Actores: Encargado(E)
Precondiciones:
Pasos: 1. El caso de uso se inicia cuando el cliente en un local del Videl-Club se dirige al ENCARGADO con los artículos a comprar. 2. E: Introduce el identificador del ARTICULO. 3. S: Registra la LINEA DE VENTA y muestra la descripción del ARTICULO, su precio y el total acumulado. *Repetir pasos 2-3 para todos los artículos que desea comprar el cliente.* 4. S: Muestra el total de la VENTA y pide confirmación de pago. 5. E: Confirma la venta (previo pago del cliente). 6. S: Registra la venta y extiende el recibo en el que figuran los precios de los distintos ARTICULOS y el total.
Extensiones:
Cuestiones:

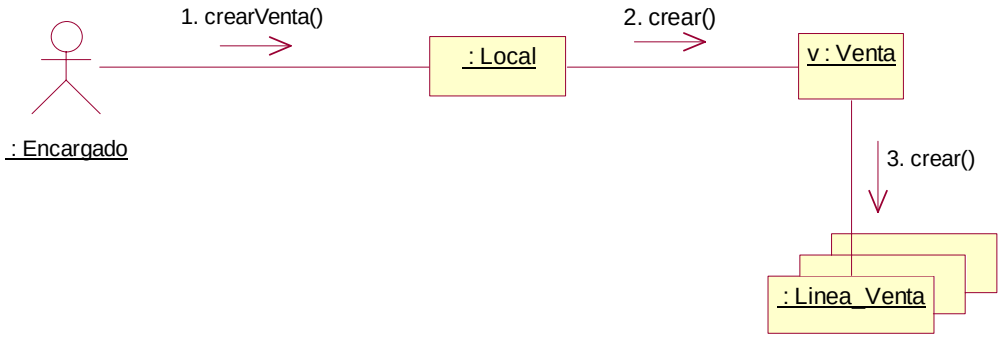
15.2 Diagrama de Secuencia del Sistema



15.3 Contratos y Colaboraciones de las operaciones

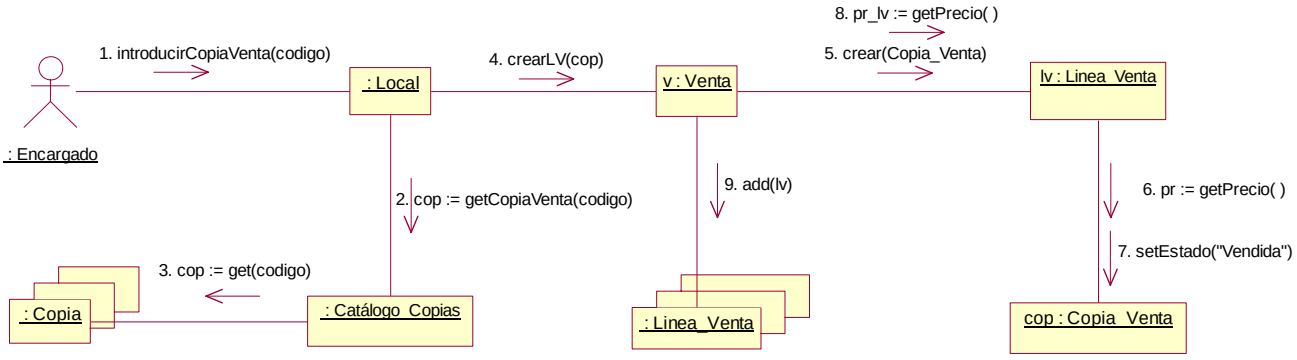
crearVenta()

Operación: crearVenta()
Responsabilidades: crear una nueva venta.
Caso de uso: Vender Articulo
Controlador: Local (Sistema)
Precondiciones:
Postcondiciones: - Se creó una nueva instancia de VENTA ‘v’. - Los atributos de ‘v’ se inicializaron. - Se creó el conjunto de LINEAS_VENTA de la VENTA ‘v’ - ‘v’ se asoció con LOCAL (Controlador).



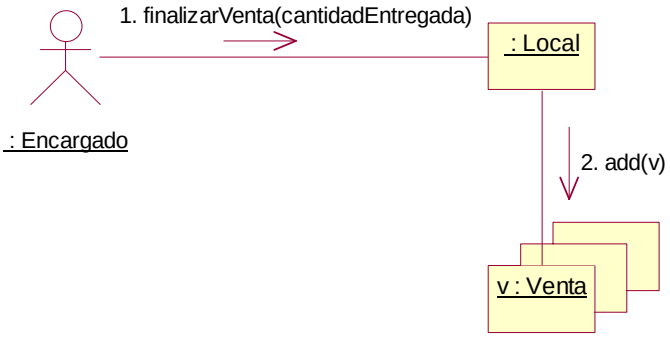
introducirCopiaVenta(codigo)

Operación: introducirCopiaVenta(codigo: tipoCodigo)
Responsabilidades: Introducir una nueva copia de venta a la venta en curso.
Caso de uso: Alquiler en Local
Controlador: Local (Sistema)
Precondiciones: - Se está procesando una VENTA ‘v’. - El sistema conoce ‘codigo’.
Postcondiciones: - Se creó una nueva instancia de LINEA_VENTA ‘lv’. - Se asoció ‘lv’ a la VENTA en curso ‘v’. - Se asoció ‘lv’ a la COPIA_VENTA ‘cop’ cuyo código es ‘codigo’. - lv.precio = cop.precio_venta. - cop.estado_copia_venta = “Vendida”. - v.total_venta = v.total_alquiler + lv.precio.



finalizarVenta(cantidadEntregada)

Operación: finalizarVenta(cantidadEntregada: tipoDinero)
Responsabilidades: Finalizar la venta actual.
Caso de uso: Venta Articulo.
Controlador: Local (Sistema)
Precondiciones: - Se está procesando una VENTA ‘v’. - cantidadEntregada >= v.total_venta.
Postcondiciones: - Se asoció ‘v’ al LOCAL (para registrar la VENTA).



16. Login

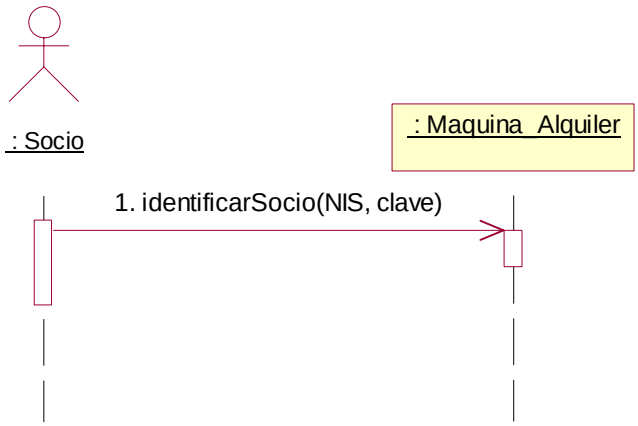
16.1 Caso de Uso

Para este CdU hemos considerado las siguientes suposiciones:

- El identificador de la tarjeta es el mismo que el del socio (Número de socio o NIS). De este modo sabremos de manera inmediata a qué socio pertenece la tarjeta.
- La propia tarjeta lleva almacenada en su interior el PIN o clave de acceso. La máquina lo único que hará será comprobar que esta clave coincide con la que ha introducido el usuario.

Caso de Uso: Login
Objetivo: Realizar el login del socio en la maquina automática de alquiler
Actores: Socio (Sc)
Precondiciones:
Pasos: 1. El caso de uso se inicia cuando el SOCIO quiere acceder a la MAQUINA DE ALQUILER. 2. S: Solicita que se introduzca la TARJETA de socio. 3. Sc: Introduce la TARJETA. 4. S: Solicita la clave de la TARJETA. 5. Sc: Introduce la clave. 6. S: Valida la clave introducida.
Extensiones: 6.1 La clave es introducida erróneamente menos de tres veces consecutivas 6.1.1 Indicar error. 6.1.2 Registra temporalmente la situación errónea. 6.1.3 Volver a paso 2. 6.2 La clave es introducida erróneamente por tercera vez 6.2.1: Indicar error. 6.2.2: Requisar TARJETA. 6.2.3: Finalizar caso de uso.
Cuestiones:

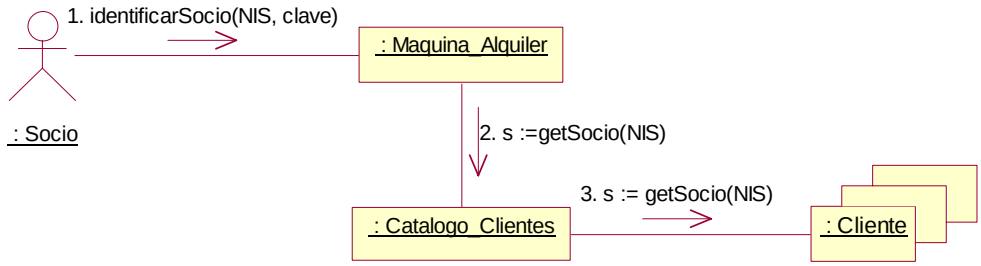
16.2 Diagrama de Secuencia del Sistema



16.3 Contratos y colaboraciones de la operación

indentificarSocio(id_tarjeta, clave)

Operación: indentificarSocio(NIS: tipoNIS, clave: tipoClave)
Responsabilidades: Identificar la tarjeta introducida para identificar al socio al que pertenece.
Caso de uso: Alquiler en Maquina
Controlador: Maquina de Alquiler (Sistema)
Precondiciones: - El sistema conoce NIS. - ‘clave’ (clave de la tarjeta) es correcta.
Postcondiciones: - El SOCIO cuyo NIS es *‘NIS’* se asoció con la MAQUINA_ALQUILER (Controlador).

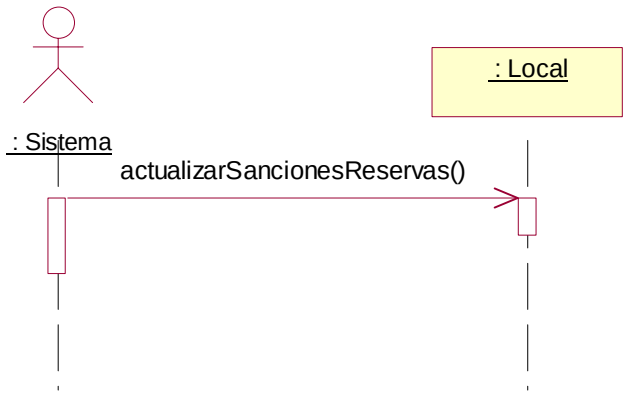


17. Actualizar Sanciones Reservas

17.1 Caso de Uso

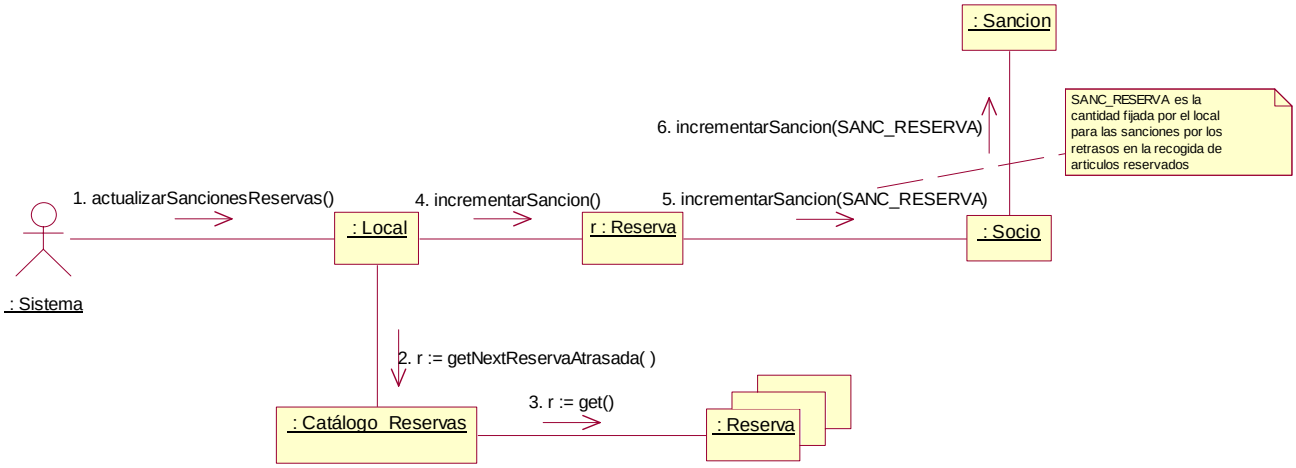
Caso de Uso: Actualizar Sanciones Reservas
Objetivo: Actualiza diariamente las sanciones de las reservas activas con retrasos en su recogida.
Actores:
Precondiciones:
Pasos: 1. S: Selecciona una RESERVA pendiente de recoger cuya fecha máxima de recogida haya expirado. 2. S: Incrementa la sanción del SOCIO que realizó la RESERVA, según la política vigente. *El Sistema repite los pasos 1-2 para todas las reservas activas.*
Extensiones:
Cuestiones:

17.2 Diagramas de Secuencia del Sistema



17.3 Contratos y Colaboración de la operación

Operación: actualizarSancionesReservas()
Responsabilidades: Actualizar las sanciones de los socios que tienen reservas activas con retrasos en su recogida.
Caso de uso: Actualizar Sanciones Reservas.
Controlador: Sistema
Precondiciones: - Esta es la única ejecución de la operación en el día.
Postcondiciones: - Para cada RESERVA ‘r’ con r.estado = “Pendiente_recoger” y con plazo de recogida expirado, • Se incrementó la sanción del SOCIO que realizó la RESERVA según la política del local.

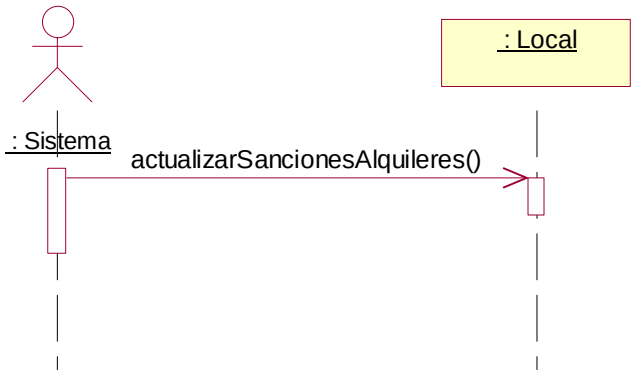


18. Actualizar Sanciones Alquileres

18.1 Caso de Uso

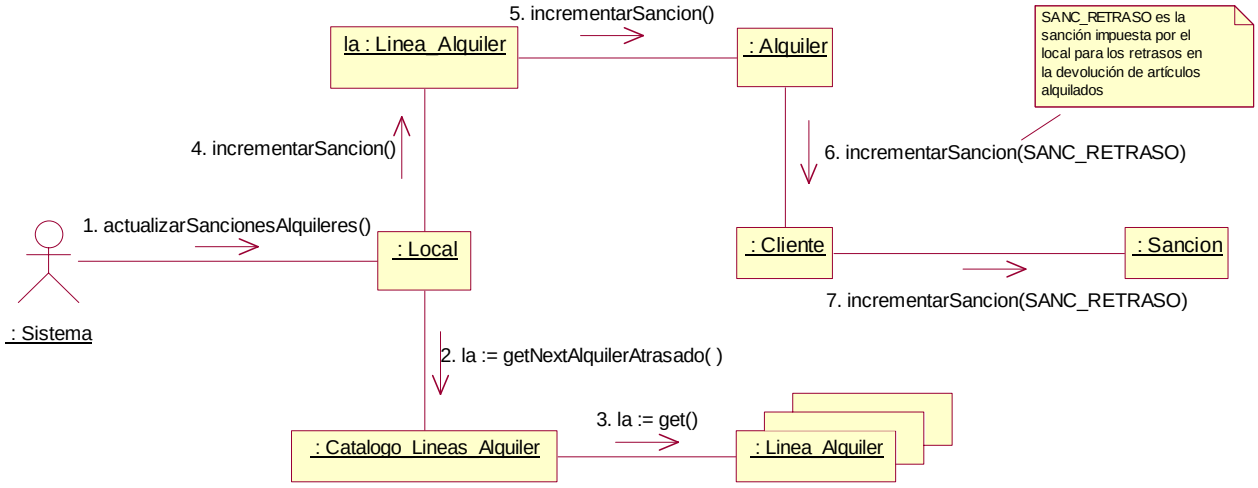
Caso de Uso: Actualizar Sanciones Alquileres
Objetivo: Actualiza diariamente las sanciones de los alquileres con retrasos en la devolución.
Actores:
Precondiciones:
Pasos: 1. S: Selecciona una LINEA_ALQUILER con plazo de devolución expirado. 2. S: Incrementa la sanción del CLIENTE que realizó el ALQUILER según la política vigente. *El Sistema repite los pasos 1-2 para todos los alquileres.*
Extensiones:
Cuestiones:

18.2 Diagrama de Secuencia del Sistema



18.3 Contratos y Colaboración de la operación

Operación: actualizarSancionesAlquileres()
Responsabilidades: Actualizar las sanciones de los socios que tienen alquileres retrasados.
Caso de uso: Actualizar Sanciones Alquileres.
Controlador: Sistema
Precondiciones: - Esta es la única ejecución de la operación en el día.
Postcondiciones: - Para cada LINEA_ALQUILER ‘la’ con ‘fecha_devolucion’ mayor que la fecha actual, • Se incrementó la sanción del SOCIO que realizó el ALQUILER, según la política del local.



19. Notificar novedades

19.1 Caso de Uso

Para este CdU sólo se ha realizado la especificación textual, ya que la práctica se ha centrado mayormente en el modelado de la gestión de los alquileres y ventas. Queda fuera de nuestro ámbito de estudio el sistema de notificación y no podríamos modelarlo bien con el estudio realizado en la práctica.

Caso de Uso: Notificar novedades
Objetivo: Realizar la notificación trimestral de las novedades incluidas por el Video-Club.
Actores:
Precondiciones:
Pasos: 1. S: Revisa las novedades incluidas por el Video-Club en los últimos tres meses. 2. S: Notifica a todos los socios de dichas novedades.
Extensiones:
Cuestiones:

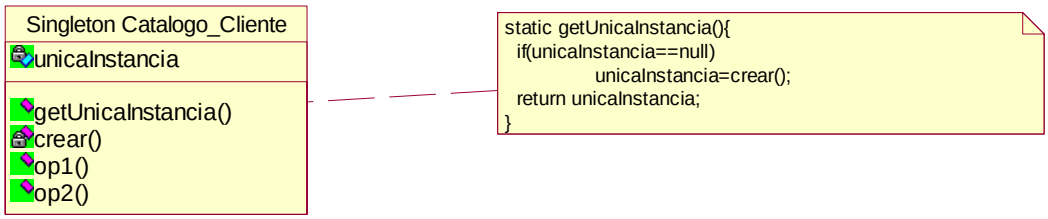
20. Patrones de Diseño

A continuación vamos a describir algunos patrones que podrían aplicarse para mejorar el diseño de la práctica.

20.1 Patrón Singleton

La aplicación del patrón Singleton es obvia para las clases *Catálogo* y *Video-Club*, puesto que tienen una única instancia y favorecemos el acceso global a dichas clases.

A continuación mostramos un ejemplo de la aplicación de este patrón para un catálogo de la práctica: *Catalogo_Clientes*.



De este modo, tenemos una instancia unica de la clase Singleton Catalogo_Cliente que es la variable de clase *unicaInstancia*, a la cual se accede mediante el método de clase *getUnicaInstancia()* cuyo código se muestra en la figura. Como se puede observar dicho código llama al método de creación de la clase que es protegido.

20.2 Patron State

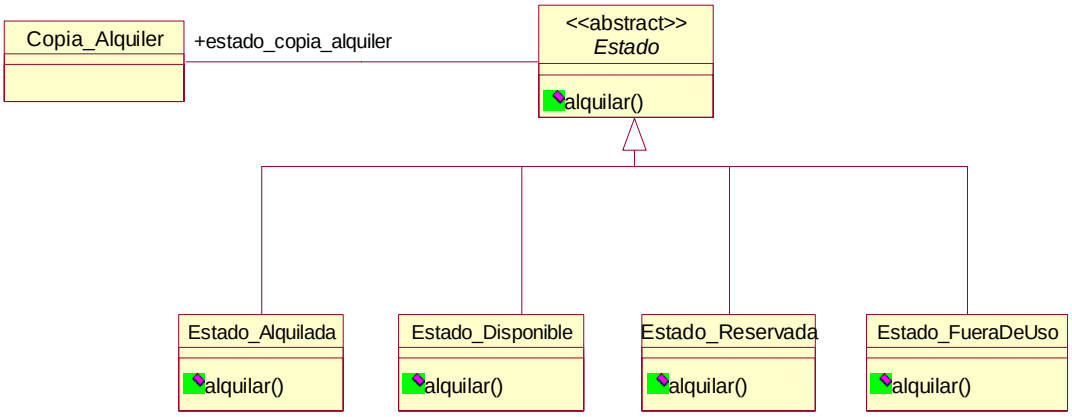
Hemos visto oportuno para las clases como *Reserva*, *Copia* y *Tarjeta* analizar la posible necesidad de aplicar el patrón estado.

La aplicación del patrón será oportuna en aquellas clases en las que el comportamiento de ciertas operaciones depende del estado de la clase, apareciendo estructuras CASE que debemos evitar en un buen diseño OO.

En concreto, en la clase Copia_Alquiler parece útil la aplicación del patrón. Podemos verlo cuando intentamos crear un método *alquilar()* en la clase *Copia_Alquiler*, cuyo código podría ser aproximadamente:

```
public alquilar() {
    case (estado_copia_alquiler) {
    “Disponible”: estado_copia_alquiler = “Alquilada”; break;
    “Reservada”:
       estado_copia_alquiler = “Alquilada;
       reserva.estado_reserva = “Recogida”; break;
    “Alquilada”: throw new PeliculaAlquiladaException(); break;
    “FueraDeUso”: throw new PeliculaFueraDeUsoException(); break;
    }
}
```

En lugar de esta estructura CASE, podríamos aplicar el patrón *estado* de la siguiente forma



Como vemos, cada subclase de la clase *Estado* hará efectivo el método *alquilar()*, dando un tratamiento específico al método según el estado en el que se encuentra la copia. De este modo, haciendo uso del polimorfismo, se resuelve de manera elegante el problema.

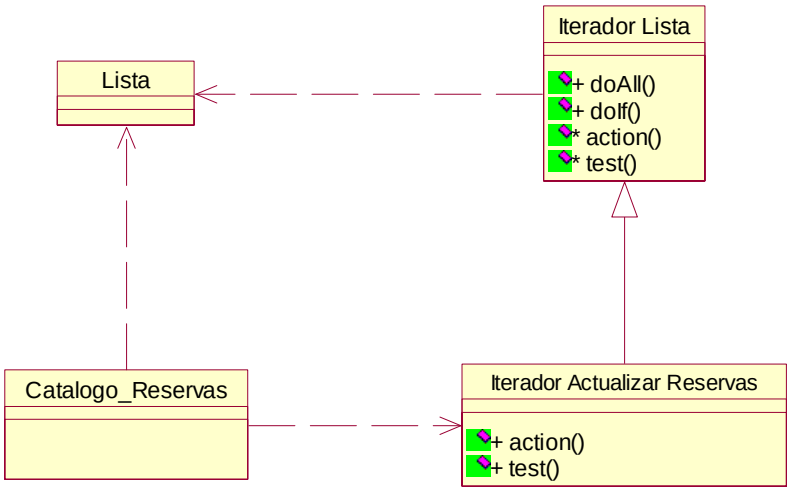
20.3 Patron Iterator

Para algunas operaciones, como las operaciones del sistema, es útil el uso de un patrón de iteración.

En este tipo de operaciones se realiza un recorrido sobre una estructura efectuando una acción si se cumple una determinada condición. Por ejemplo en el CdU “Actualizar Reservas” se realiza un recorrido sobre todo el “catálogo de reservas” y se intenta asignar una copia a aquellas reservas que están en espera.

Usaremos un Iterador Interno evitando así a la clase “catálogo de reservas” el control del recorrido sobre la lista de reservas.

La estructura del Iterador sería como se muestra a continuación:



De este modo la clase Iterador Actualizar Reservas tendría que definir los métodos *action()* y *test()* aproximadamente del siguiente modo:

```
public test(Reserva r) {
	return (r.estaEnEspera() == true);
}

public action(Reserva r) {
	SI hay una copia disponible ‘c’ del artículo reservado en ‘r’ ENTONCES
		r.asignarCopia(c);
		c.setEstadoReservada()
}
```

20.4 Patron Strategy / Abstract Factory

Sin duda, uno de los patrones más importante que debería aparecer en una aplicación como la de esta práctica sería el patrón Strategy (estrategia). También es uno de los patrones más difíciles de aplicar, como comprobaremos durante este apartado.

En un videoclub podemos aplicar numerosas políticas de precios para la venta o el alquiler de de artículos, así como distintas políticas de días de préstamo, todo esto dependiendo del tipo de cliente, del día de la semana o del producto elegido. Para mostrar la aplicación del patrón Strategy, tomaremos como ejemplo las posibles políticas en el alquiler de películas.

Políticas de precios

Intentaremos abordar en primer lugar el problema de las políticas de precios. Algunas políticas de precios que podríamos encontrarnos en un videoclub serían:

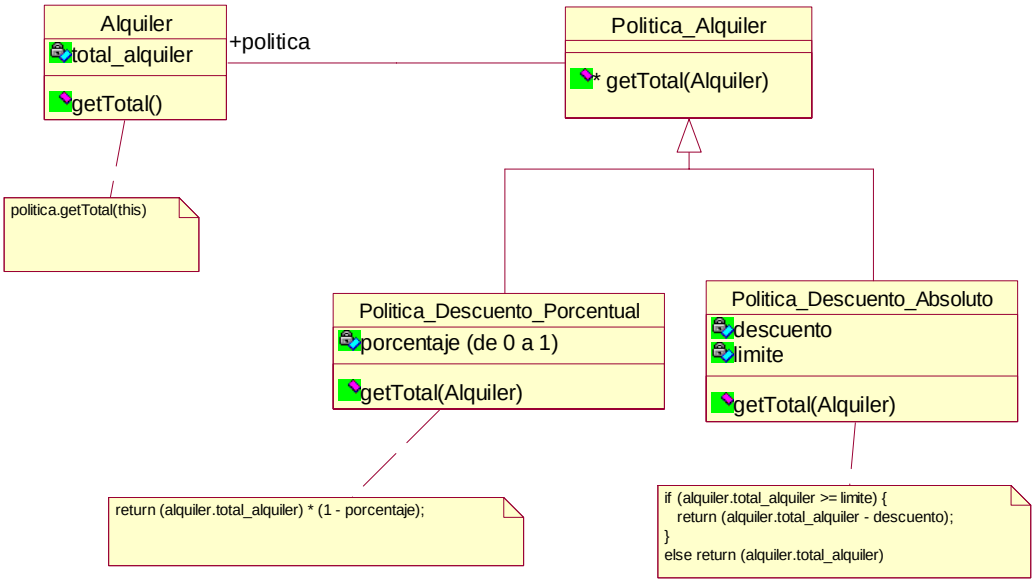
- Todos los socios tienen un 10% de descuento sobre todo el alquiler, y los encargados un 15%.
- De lunes a viernes, por un alquiler superior a 6 euros se descontará 1 euro al total.
- Los miércoles, las películas ordinarias valen 1 euro y las novedades 2 euros.
- Dos películas ordinarias por el precio de una.
- ...

A partir de los ejemplos anteriores podemos identificar dos tipos principales de políticas de precios:

1.- Políticas de todo el alquiler. A este grupo pertenecerían aquellas políticas que se aplican a todo el alquiler, como por ejemplo:

- Todos los socios tienen un 10% de descuento sobre todo el alquiler, y los encargados un 15%.
- De lunes a viernes, por un alquiler superior a 6 euros se descontará 1 euro al total.

Este tipo de políticas se pueden englobarse en 2 clases, a la vista de los ejemplos mencionados:



Politica_Descuento_Porcentual permite aplicar un descuento porcentual, determinado por el atributo *porcentaje*, sobre un alquiler completo.

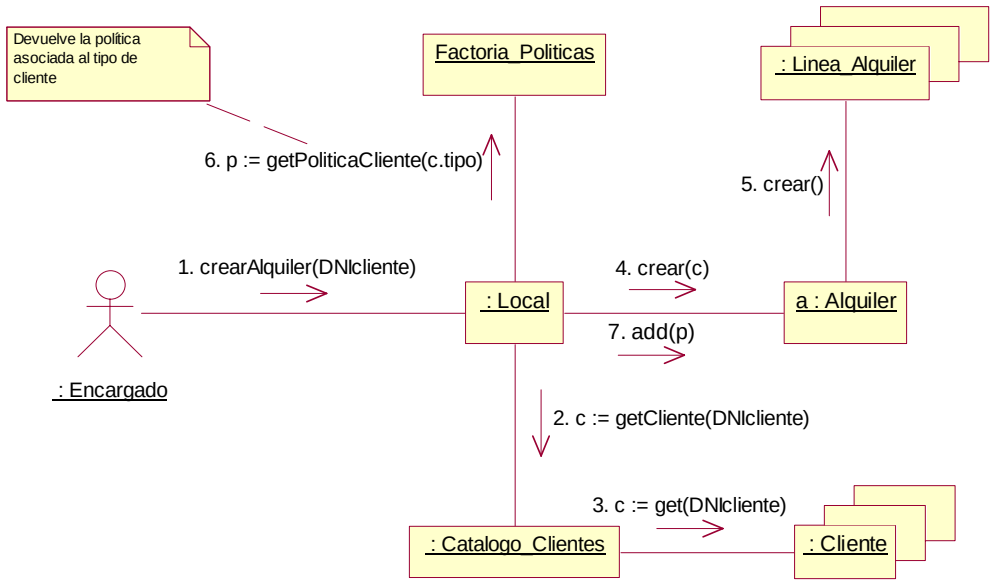
Politica_Descuento_Absoluto nos servirá para aplicar un descuento absoluto, indicado por *descuento*, si el total neto del alquiler supera la cantidad fijada por el atributo *límite*.

Puede observarse como estas clases se han definido de forma totalmente configurable, favoreciendo de este modo la extensibilidad. Los atributos configurables, como *porcentaje*, *descuento* o *límite* se introducirán como parámetros de creación al crear la política concreta.

Pero, ¿cómo y cuándo se crean estas políticas? *Alquiler* no debe conocer las políticas concretas del videoclub, así que usaremos una factoría de políticas (usando el patrón Abstract Factory). De este modo *Alquiler* sólo conocerá la interfaz de creación, pero no las estrategias concretas. A esta factoría le pasaríamos un parámetro como el tipo de cliente o la fecha actual y nos devolvería la política asociada a ese parámetro.

Ante la pregunta de cuándo se crean estas políticas solo cabe una respuesta: al inicio del alquiler. Este tipo de políticas no dependen de un producto concreto, sino de parámetros como el tipo de cliente o la fecha actual, que se conocen al crear el alquiler.

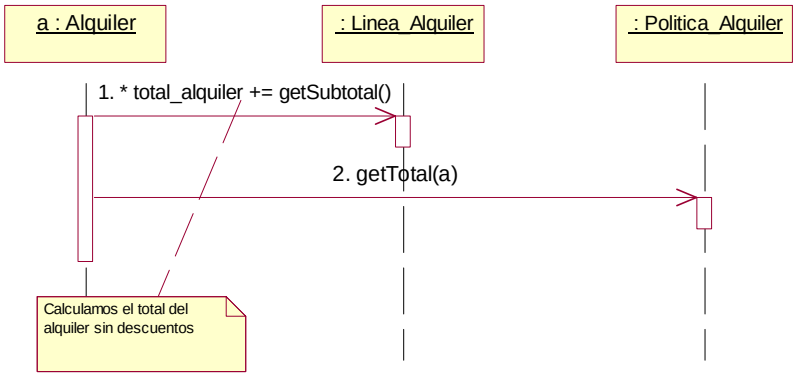
En el primer caso (según el tipo de cliente) la política se crearía una vez que identificamos al cliente, y se asociaría al alquiler. El diagrama de colaboraciones de la operación *crearAlquiler()* quedaría como sigue[2]:



Como podemos observar, tras crear el alquiler obtenemos la política asociada al tipo de cliente y la asociamos al alquiler[3]. Al finalizar el alquiler aplicaremos esta política al total "neto" (sin descuentos) del alquiler, para así obtener el precio real. El siguiente diagrama de secuencia mostraría cómo se calcularía el total del alquiler:

[2] Hemos simplificado la operación eliminando lo referente a las sanciones, para centrarnos en las políticas.

[3] Estamos suponiendo que al alquiler sólo tiene asociado una política en un momento determinado. El caso de que puedan aplicarse distintas políticas simultáneamente se tratará más adelante mediante el patrón *Composite*.



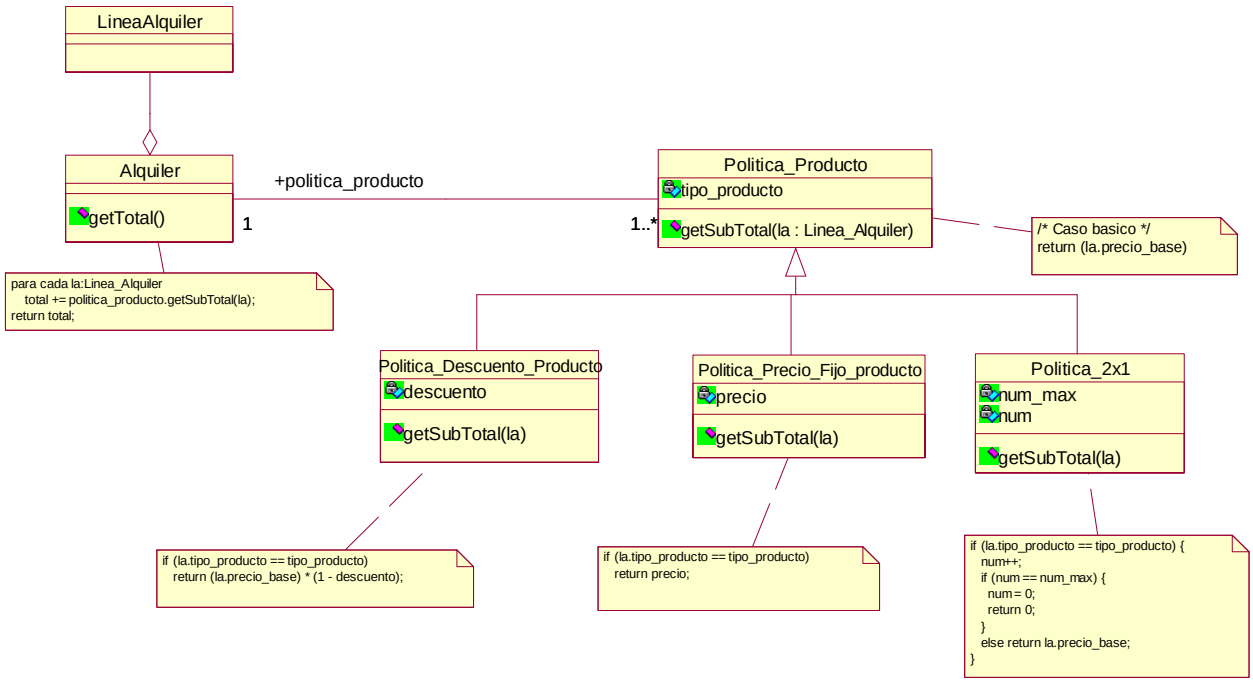
Algo similar ocurriría con la política referente a la fecha actual. La política se obtendría tras crearse el alquiler, y se asociaría al mismo.

2.- Políticas de línea de alquiler o de producto. Aquí encontraríamos las políticas que se aplican a una o varias líneas de alquiler, pero no a todo el alquiler, por ejemplo:

- Los miércoles, las películas ordinarias valen 1 euro y las novedades 2 euros.
- Los socios tienen un descuento del 20% en las novedades.
- Dos películas ordinarias por el precio de una.

Este caso es más complicado que resolver que el anterior. Una política de producto, aunque se aplique a una línea de alquiler individual, debe conocerse en todo el alquiler, puesto que puede haber varios productos relacionados. Pensemos en el caso del 2x1 (2 películas ordinarias por el precio de una). Si aparece una película ordinaria en el alquiler, debe existir algún objeto que registre este hecho y controle si se introducen más películas ordinarias, para así aplicar el descuento. La solución más natural es que sea la propia política de producto la que controle si efectivamente se cumplen las condiciones para aplicar el descuento. Lo que haremos será almacenar en *Alquiler* la lista de políticas de producto aplicables a las líneas de alquiler del alquiler actual.

Hemos identificado tres tipos generales para este tipo de políticas, que englobarían a los ejemplos mencionados:



Política_Descuento_Producto permite aplicar un descuento porcentual, determinado por el atributo *descuento*, a un producto. Vemos que estas políticas sólo se aplican si el tipo de producto coincide con el que buscamos. Con esta política podríamos, por ejemplo, asignar un 15% de descuento a las películas ordinarias.

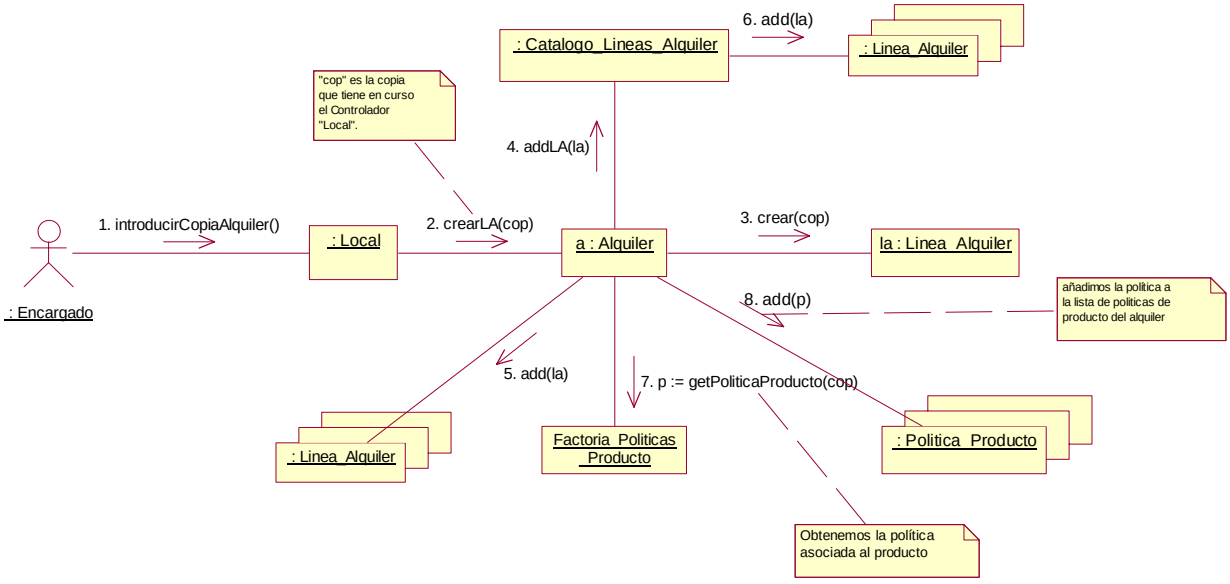
Politica_Precio_Fijo_Producto asigna un precio fijo (atributo *precio*) a un tipo de producto. Gracias a esta política podemos establecer el precio de las películas ordinarias a 1 euro, y el de las novedades a 2 euros, por poner un ejemplo.

Politica_2x1 permite realizar ofertas del tipo 2x1, 3x2, etc. El atributo *num* indica las unidades ya registradas del tipo de producto; *num_max* indica la cantidad de productos con oferta a la que hay que llegar para que el descuento se aplique. Por ejemplo, para una oferta de 2x1 en películas ordinarias, *num_max* será igual a 2, y *num* se incrementará con cada película ordinaria introducida. Mientras *num* sea menor que *num_max*, la política devolverá el precio normal del producto. Si *num* llega a *num_max*, el precio devuelto será 0 (ese artículo es gratis).

Al igual que en el caso de las políticas de todo el alquiler, tendríamos una factoría de políticas de productos, que dado un parámetro (puede ser el tipo de socio, la fecha actual o un producto concreto) devolverá la política asociada al parámetro.

¿Cuándo se crean las políticas? Habría varios posibles puntos de creación de este tipo de políticas:

a) Según la fecha actual: al inicio del alquiler. Por ejemplo, si es miércoles se asociará al alquiler actual la/s política/s de producto que correspondan a ese día (Películas ordinarias 1euro y novedades 2euros).
b) Según el tipo de cliente: al identificar al cliente, similar al caso de las políticas de todo el alquiler. Por ejemplo, si el cliente es socio, se asociarán a su alquiler la/s política/s de producto que le correspondan (20% de descuento en novedades).
c) Según el producto concreto: al registrar la linea de alquiler del producto. Por ejemplo, si se registra una película ordinaria, se asociará al alquiler actual la política referente a ese tipo de películas (2x1). Veamos de forma simplificada como sería para este último caso:



Fijémonos en las operaciones 7 y 8: obtenemos la política asociada al producto concreto y la añadimos a la lista de políticas de producto del alquiler.

Ahora bien, ¿cuándo se aplican estas políticas? Hasta ahora tan sólo las hemos creado y añadido a la lista. Será al final del alquiler, cuando se calcule el total, cuando se intentará aplicar cada política de la lista a cada linea de alquiler. Nos quedaremos siempre con el precio más bajo, favoreciendo así al cliente.

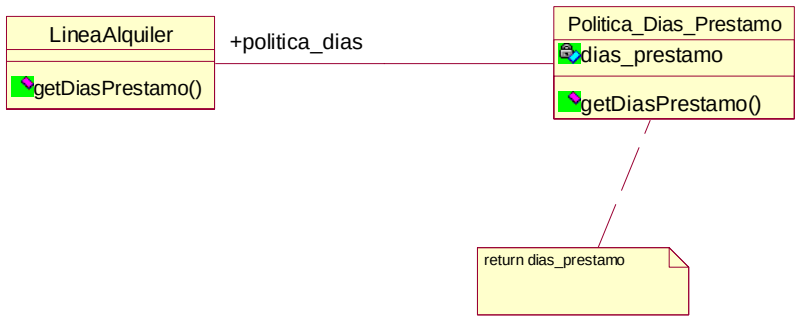
Cabe destacar que este tipo de políticas no son incompatibles con las políticas de todo el alquiler, pudiéndose aplicar conjuntamente ambos tipos de políticas para un mismo alquiler.

Políticas de días de préstamo

Por otro lado, el plazo de devolución de las películas también puede variar, dependiendo del día de la semana y del tipo de película. Tendríamos, pues, otro tipo de políticas, las políticas de días de préstamo:

- Las películas muy viejas tienen 7 días de plazo.
- Las novedades tienen 1 día de plazo, excepto los sábados, en el que el plazo se amplía a 2 días.
- ...

Para este caso nos bastaría con una clase que nos permita configurar los días de préstamos asignados a una línea de alquiler:



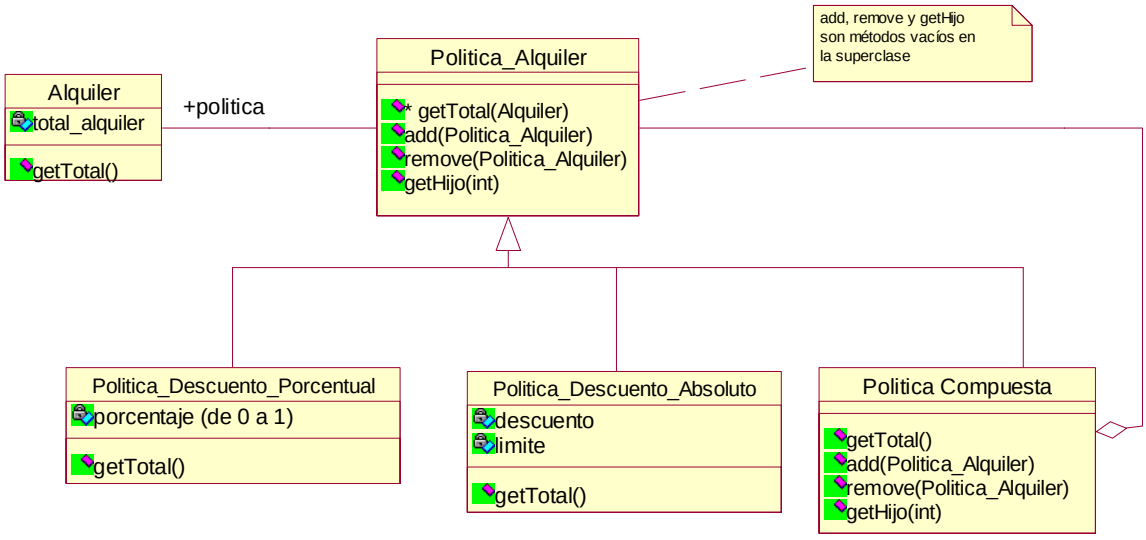
Al igual que las políticas de precios, mediante una factoría crearíamos las políticas concretas, según el día, el tipo de cliente o el producto. La creación y asignación de políticas a las líneas de alquiler sería totalmente análogo al caso de las políticas de precios:

a) Según la fecha actual: al iniciar el alquiler.
b) Según el tipo de cliente: al inicial el alquiler.
c) Según el producto: al registrar el producto.

20.5 Patron Strategy / Composite

Las mejoras introducidas en el apartado anterior no permiten todavía, por ejemplo, aplicar simultáneamente 2 políticas de todo el alquiler (10% de descuento y además 1 euro de descuento si el alquiler supera los 6 euros). Para solucionar esto podríamos tener una lista de políticas de alquiler, similar a la lista de políticas de producto que veíamos en el apardado anterior. Pero esta solución no sería muy acertada, puesto que *Alquiler* tendría que realizar el recorrido por la lista, aunque la mayoría de las veces sólo se aplique una de estas políticas.

La mejor solución está en aplicar el patrón *Composite* a las políticas de alquiler. Modificaremos la jerarquía de políticas para contemplar el caso de políticas compuestas. Veamos el diagrama de clases:

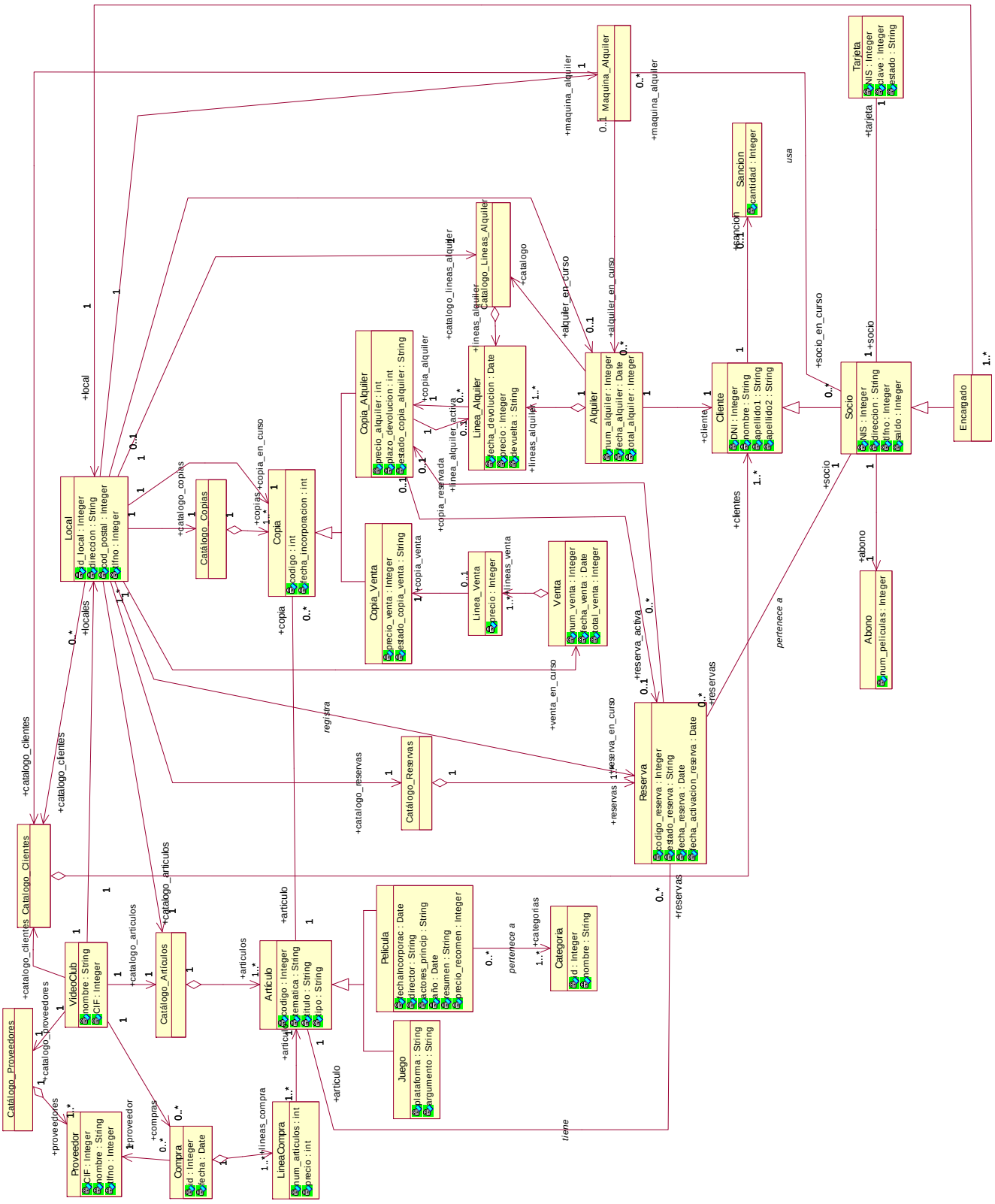


La clase *Política_Compuesta* estará formada por políticas simples. El método *getTotal()* recorrerá las políticas simples y devolverá la que dé menor resultado, favoreciendo así al cliente.

Los métodos *add*, *remove* y *getHijo* son exclusivos del manejo de la estructura compuesta. Puede verse que la superclase también incorpora los métodos *add*, *remove* y *getHijo*, pero en este caso están vacíos. Hacemos esto para favorecer la transparencia, ya que con este diseño la clase cliente (*Alquiler*) no se da cuenta si está manejando o no un objeto compuesto.

Este patrón puede aplicarse también a las políticas de producto. Así evitaríamos tener una lista de políticas de producto en el alquiler, sustituyéndola por un sólo objeto compuesto. Además, el recorrido de las distintas políticas ya no tendría que hacerse en *Alquiler*, sino que se haría en la propia clase *composite*. El diagrama de clases sería totalmente análogo al de las políticas de producto.

21. Diagrama de clases



22. Listado de Código Generado

```
public class Abono
{
   private Integer num_peliculas;

   public Abono()
   {
   }
}


public class Alquiler
{
   private Integer num_alquiler;
   private Date fecha_alquiler;
   private Integer total_alquiler;
   public Linea_Alquiler lineas_alquiler[];
   public Catalogo_Lineas_Alquiler catalogo;
   public Cliente cliente;

   public Alquiler()
   {
   }

   public void crear(Cliente c)
   {
   }

   public void crearLA(Copia_Alquiler cop)
   {
   }

   public void getTotal()
   {
   }

   public void incrementarSancion()
   {
   }
}


public class Artículo
{
   private Integer codigo;
   private String tematica;
   private String titulo;
   private String tipo;
   public Copia copia[];
   public Reserva reservas[];

   public Artículo()
   {
   }

   public void getCopiaAlquilerDisponible(Integer id_local)
   {
   }

   public Integer getCodigo()
   {
   }
}


public class Catalogo_Lineas_Alquiler
{
   public Linea_Alquiler lineas_alquiler;

   public Catalogo_Alquileres()
   {
```

```
    }

    public void addLA(Linea_Alquiler linea_alquiler)
    {
    }

    public Linea_Alquiler getNextAlquilerAtrasado()
    {
    }
}


public class Catalogo_Clientes
{
    public Cliente clientes[];

    public Catalogo_Clientes()
    {
    }

    public Cliente getCliente(Integer DNI)
    {
    }

    public void addCliente(Cliente c)
    {
    }
}



public class Categoria
{
    private Integer id;
    private String nombre;

    public Categoria()
    {
    }
}



public class Catálogo_Artículos
{
    public Artículo articulos[];

    public Catálogo_Artículos()
    {
    }

    public void getArticulo(Integer codigo)
    {
    }

    public void getCopiaAlquilerDisponible(Integer id_local, Integer
codigoArticulo)
    {
    }
}



public class Catálogo_Copias
{
    public Copia copias[];

    public Catálogo_Copias()
    {
    }

    public void getCopiaAlquiler(int codigoArticulo)
    {
```

```
    }

    /**
    @roseuid 3D14C67701E1
    */
    public Copia_Venta getCopiaVenta(int codigo)
    {
    }
}


public class Catálogo_Proveedores
{
    public Proveedor proveedores[];

    public Catálogo_Proveedores()
    {
    }
}


public class Catálogo_Reservas
{
    public Reserva reservas[];


    public Catálogo_Reservas()
    {
    }

    public void crearReserva(Socio socio, Artículo_Alquiler articulo)
    {
    }

    public void getNextReservaEn_espera()
    {
    }

    public Reserva getNextReservaAtrasada()
    {
    }
}


public class Catálogo_Tarjetas
{
    public Tarjeta tarjetas;

    public Catálogo_Tarjetas()
    {
    }

    public void getTarjeta()
    {
    }

    public void addTarjeta(Tarjeta tarjeta)
    {
    }
}




public class Cliente
{
    private Integer DNI;
```

```
   private String nombre;
   private String apellido1;
   private String apellido2;
   public Sancion sancion;

   public Cliente()
   {
   }

   public Integer getSancion()
   {
   }

   public Reserva getReserva()
   {
   }

   public void eliminarSancion()
   {
   }

   public void crear(Integer DNI, String nombre, String apellido1, String
apellido2)
   {
   }

   public void incrementarSancion(Integer cantidad)
   {
   }
}
```

```
public class Compra
{
   private Integer id;
   private Date fecha;
   public Proveedor proveedor;
   public LineaCompra lineas_compra[];

   public Compra()
   {
   }
}
```

```
public class Copia
{
   private int codigo;
   private int fecha_incorporacion;
   public Artículo articulo;

   public Copia()
   {
   }
}
```

```
public class Copia_Alquiler extends Copia
{
   private int precio_alquiler;
   private int plazo_devolucion;

   /**
     estado_copia_alquiler = {Disponible, Alquilada, Reservada, FueraDeUso}
   */
   private String estado_copia_alquiler;
   public Linea_Alquiler linea_alquiler_activa;
   public Reserva reserva_activa;
```

```
    public Copia_Alquiler()
    {
    }

    public int getPrecioAlquiler()
    {
    }

    public int getPlazoDevolucion()
    {
    }

    public void setEstado(int nuevoEstado)
    {
    }

    public void setLA(Linea_Alquiler linea_alquiler)
    {
    }

    public void devolverCopia()
    {
    }

    public Copia_Alquiler getCopia()
    {
    }
}


public class Copia_Venta extends Copia
{
    private Integer precio_venta;

    /**
      estado_copia_venta = {Vendida, En_venta}
    */
    private String estado_copia_venta;

    public Copia_Venta()
    {
    }

    public void setEstado(String nuevoEstado)
    {
    }

    public Integer getPrecioVenta()
    {
    }
}



public class Encargado extends Socio
{
    public Local local;

    public Encargado()
    {
    }
}


public class Juego extends Artículo
{
    private String plataforma;
    private String argumento;

    public Juego()
    {
    }
}
```

```
public class Linea_Alquiler
{
   private Date fecha_devolucion;
   private Integer precio;

   /**
    devuelta = {SI, NO}
   */
   private String devuelta;
   public Copia_Alquiler copia_alquiler;

   public Linea_Alquiler()
   {
   }

   public void crear(Copia_Alquiler cop)
   {
   }

   public Integer getPrecio()
   {
   }

   public void setDevuelta()
   {
   }

   public void incrementarSancion()
   {
   }
}
```

```
public class Linea_Venta
{
   private Integer precio;
   public Copia_Venta copia_venta;

   public Linea_Venta()
   {
   }

   public void crear(Copia_Venta copia)
   {
   }

   public Integer getPrecio()
   {
   }
}
```

```
public class LineaCompra
{
   private int num_articulos;
   private int precio;
   public Artículo articulo;

   public LineaCompra()
   {
   }
}
```

```
public class Local
{
   private Integer id_local;
```

```
    private String direccion;
    private Integer cod_postal;
    private Integer tlfno;
    public Venta venta_en_curso;
    public Catalogo_Clientes catalogo_clientes;
    public Catálogo_Artículos catalogo_articulos;
    public Catálogo_Copias catalogo_copias;
    public Copia copia_en_curso;
    public Maquina_Alquiler maquina_alquiler;
    public Reserva reserva_en_curso;
    public Catálogo_Tarjetas catalogo_tarjetas;
    public Catálogo_Reservas catalogo_reservas;
    public Alquiler alquiler_en_curso;
    public Catalogo_Alquileres catalogo_alquiler;

    public Local()
    {
    }

    public void crearAlquiler(int DNIcliente)
    {
    }

    public void getCopiaAlquilerReservada(Integer codigoArticulo)
    {
    }

    public void getCopiaAlquiler(Integer codigoArticulo)
    {
    }

    public void introducirCopiaAlquiler()
    {
    }

    public void finalizarAlquiler(Integer cantidad)
    {
    }

    public void realizarAltaCliente(Integer DNI, String nombre, String
apellido1, String apellido2)
    {
    }

    public void realizarAltaSocio(Integer DNI, String nombre, String apellido1,
String apellido2, String direccion, Integer tf, Integer num_cuenta, String
email, Integer cuota)
    {
    }

    public void devolverArticulo(Integer codigo)
    {
    }

    public void crearReserva(Integer DNIsocio, Integer codigoArticulo)
    {
    }

    public void obtenerCopiaAlquilerDisponible(Integer codigoArticulo)
    {
    }

    public void finalizarReserva()
    {
    }

    public void anularReserva(Integer DNIsocio, Integer codigoArticulo)
    {
    }

    public void actualizarReservaEn_espera()
    {
    }

    public void actualizarSancionesAlquileres()
    {
    }
```

```
    public void actualizarSancionesReservas()
    {
    }

    public void recogerTarjeta(Integer DNIsocio)
    {
    }

    public void crearVenta()
    {
    }

    public void introducirCopiaVenta(Integer codigo)
    {
    }
}



public class Maquina_Alquiler
{
    public Alquiler alquiler_en_curso[];
    public Socio socio_en_curso[];
    public Catálogo_Tarjetas catalogo_tarjetas;

    public Maquina_Alquiler()
    {
    }

    public void identificarSocio(Integer id_tarjeta, Integer clave)
    {
    }

    public void crearAlquilerEnMaquina()
    {
    }

    public void finalizarAlquilerEnMaquina()
    {
    }

    public void introducirArticulo(Integer codigo)
    {
    }

    public void cambiarClaveTarjeta(Integer nuevaClave)
    {
    }

    public void recargarMonedero(Integer cantidad)
    {
    }
}



public class Pelicula extends Artículo
{
    private Date fechaIncorporac;
    private String director;
    private String actores_princip;
    private Date año;
    private String resumen;
    private Integer precio_recomen;
    public Categoria categorias[];

    public Pelicula()
    {
    }
}
```

```
public class Política_Precio
{
   private Integer id;
   private String descripcion;
   private Integer descuentoSocio;
   private Integer sancionPerdida;
   private Integer sancionReserva;
   private Integer sancionRetraso;
   private Integer fianzaNoSocios;

   public Política_Precio()
   {
   }

   public Integer getSancionRetraso()
   {
   }

   public Integer getSancionReserva()
   {
   }
}
```

```
public class Proveedor
{
   private Integer CIF;
   private String nombre;
   private Integer tlfno;

   public Proveedor()
   {
   }
}
```

```
public class Reserva
{
   private Integer codigo_reserva;

   /**
     Valores posibles: {Recogida, En_espera, Pendiente_Recoger, Anulada}
   */
   private String estado_reserva;

   /**
Fecha en la que la reserva fue efectuada (de cara a dar prioridad a las
reservas más antiguas)
   */
   private Date fecha_reserva;

   /**
Fecha en la que la reserva pasó a estar 'activa', es decir, fecha en la que el
artículo reservado pasó al estado 'Pendiente_recoger'
   */
   private Date fecha_activacion_reserva;
   public Socio socio;
   public Artículo articulo;
   public Copia_Alquiler copia_reservada;

   public Reserva()
   {
   }

   public Copia_Alquiler recogerCopiaAlquiler()
   {
   }

   public void crear(Socio socio, Artículo_Alquiler articulo)
   {
   }
```

```
    public void setEstado(String nuevoEstado)
    {
    }

    public void setFechaActivacionReserva(Date fecha)
    {
    }

    public void asociarCopia(Copia_Alquiler copia)
    {
    }

    public void anularReserva()
    {
    }

    public Integer getCodigoArticulo()
    {
    }

    public void incrementarSancion()
    {
    }
}



public class Sancion
{
    private Integer cantidad;

    public Sancion()
    {
    }

    public Integer getCantidad()
    {
    }

    public void incrementarSancion(Integer cantidad)
    {
    }
}



public class Socio extends Cliente
{
    private Integer NIS;
    private String direccion;
    private Integer tlfno;
    private Integer saldo;
    public Tarjeta tarjeta;
    public Reserva reservas[];
    public Abono abono;
    public Maquina_Alquiler maquina_alquiler[];
    public Catálogo_Tarjetas catalogo_tarjetas;

    public Socio()
    {
    }

    public void decrementarSaldo(Integer cantidad)
    {
    }

    public void crear(Integer DNI, String nombre, String apellido1, String
apellido2, String direccion, Integer tf, Integer num_cuenta, String email,
Integer cuota)
    {
    }

    public void setClaveTarjeta(Integer nuevaClave)
    {
    }
```

```
    public void añadirSaldo(Integer cantidad)
    {
    }

    public void anularReserva(Artículo_Alquiler articulo)
    {
    }

    public void recogerTarjeta()
    {
    }
}
public class Tarjeta
{
    private Integer id_tarjeta;
    private Integer clave;
    private String estado;
    public Socio socio;

    public Tarjeta()
    {
    }

    public Socio getSocio()
    {
    }

    */
    public void crear()
    {
    }

    public void setClave(Integer nuevaClave)
    {
    }

    public void setEstado(String nuevoEstado)
    {
    }
}


public class Venta
{
    private Integer num_venta;
    private Date fecha_venta;
    private Integer total_venta;
    public Linea_Venta lineas_venta[];

    public Venta()
    {
    }

    public void crear()
    {
    }

    public void crearLV(Copia_Venta copia)
    {
    }
}


public class VideoClub
{
    private String nombre;
    private Integer CIF;
    public Local locales[];
    public Compra compras[];
    public Catálogo_Artículos catalogo_articulos;
    public Catálogo_Proveedores catalogo_proveedores;
    public Catalogo_Clientes catalogo_clientes;
    public Catálogo_Tarjetas catalogo_tarjetas;

    public VideoClub()
    {
```

```
    }
}
```