Programación 4 – solución práctico 3

Ejercicio 3:

Notar que en este ejercicio ya están identificadas las operaciones del sistema, que son:

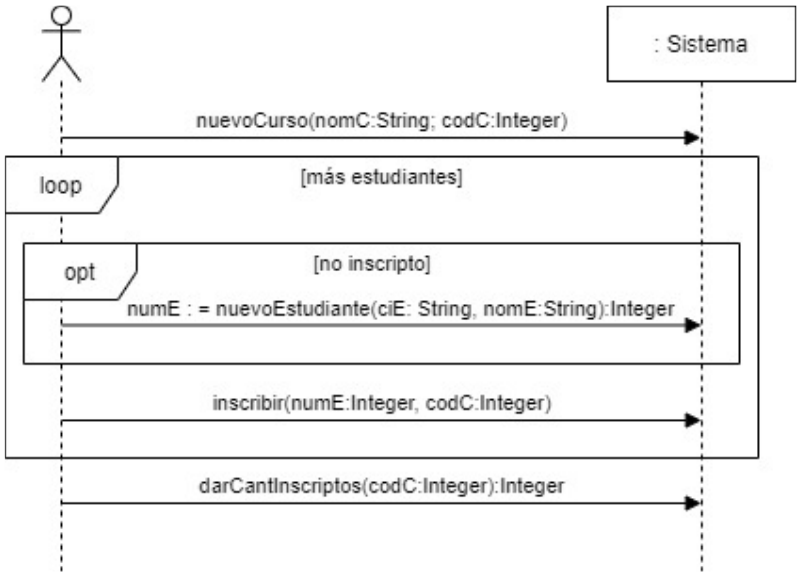
- `nuevoCurso`
- `nuevoEstudiante`
- `darCantInscriptos`
- `inscribir`

Esto no es el caso habitual en un proceso de análisis, donde se cuenta con una descripción textual del caso de uso (CU) y a partir de ahí se deben generar las operaciones del sistema necesarias. El objetivo de este ejercicio es discutir aspectos de sintaxis y semántica de la notación UML para Diagramas de Secuencia del Sistema (DSS). El modelado de un caso de uso de características realistas se verá en el ejercicio 8.

a)

Para realizar el DSS de este CU asumiremos el curso típico de eventos, es decir, asumimos que las precondiciones de las operaciones del sistema se cumplen con los datos ingresados, salvo que el texto del CU indique la ocurrencia de situaciones excepcionales e indique cómo se realiza la interacción en esos casos. Por ejemplo, para este CU se asume que el código del curso a ser ingresado no existe en el sistema. En cambio, para el estudiante se contempla el caso de que no exista en el sistema y se indica cómo proceder en esa situación.

El DSS es el siguiente:



Observar lo siguiente:

- En todas las operaciones se incluye el nombre y el tipo de cada parámetro.
- En las funciones se usa una variable para almacenar el resultado, si es necesario utilizarlo en una operación posterior. Para `nuevoEstudiante`, el resultado se almacena en `numE` porque se usa en la siguiente operación (`inscribir`). En cambio, al resultado de

darCantInscriptos no es necesario acceder en el diagrama, por lo tanto no es necesario utilizar una variable para almacenarlo.

Para modelar estas interacciones, siempre puede ayudar imaginarse cómo sería la interfase gráfica de usuario. Pero, recordar que el actor NO es el usuario, sino que es un objeto de la capa de presentación que realiza la comunicación entre el usuario y la capa lógica.

El CU se inicia cuando el usuario ingresa un nombre y un código para un nuevo curso. Luego se inicia una iteración (loop), donde el valor de verdad de la expresión booleana de la guarda [más estudiantes] dependerá de las acciones del usuario (si quiere o no agregar más estudiantes, lo que indicará mediante un botón o algún control similar en la interfase de usuario). Algo similar ocurre con la guarda [no inscripto]: el que sabe si está inscripto o no es el usuario. Imaginar que es un botón en la interfase, que da las opciones inscripto o no inscripto: si se selecciona la primera se va directamente a `inscribir`, si se selecciona la segunda se ingresa al bloque opt para invocar a `nuevoEstudiante`. Observar que la invocación a inscribir se hace con el parámetro numE, cuyo valor será el que devuelve `nuevoEstudiante` si se realizó la inscripción del estudiante o será un valor ingresado por el usuario si el estudiante ya estaba inscripto. Finalmente, la operación `darCantInscriptos` es una consulta (no modifica el estado del sistema) y está por fuera de la iteración loop.

Importante:

- Las construcciones de control de los DSS en UML (loop, opt, alt, ref) no están pensadas para representar un seudocódigo de la interacción del actor con el sistema. El principal objetivo del DSS es identificar las operaciones del sistema que son necesarias y que luego deberán diseñarse. Aclaraciones sobre la lógica pueden incluirse en notas, pero no conviene abusar del uso de notas. Si algo puede expresarse con los elementos sintácticos de UML, es incorrecto expresarlo mediante una nota.
- Las guardas (condiciones lógicas de las construcciones loop, opt y alt) son condiciones que deben ser posible de ser verificadas por parte del actor sin tener que consultar al sistema. Si alguna guarda necesita información del sistema para ser evaluada, entonces debe existir una operación del sistema que devuelva esa información en una variable, la cual será parte de la guarda. Existe el riesgo de abusar del uso de las guardas; vale la misma observación realizada al final del ítem anterior.

b)

El uso de memoria permite que durante el desarrollo del CU, el sistema recuerde información temporalmente para ser usada en operaciones que se invocan posteriormente en el mismo CU. Esto puede ser utilizado para reducir el tráfico de información entre el actor y el sistema, y también para hacer más eficientes algunas operaciones del sistema.

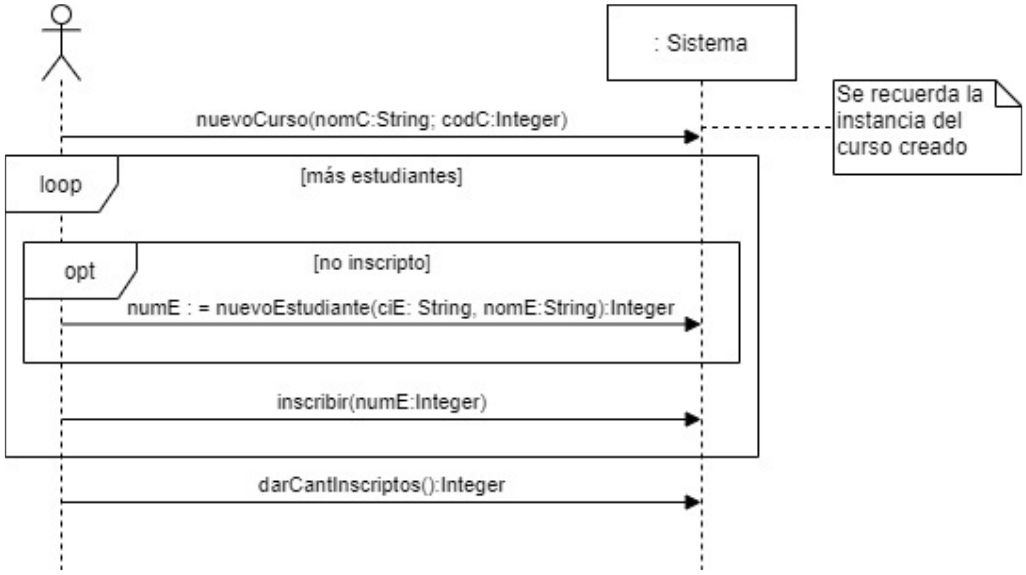
El DSS de la parte a) en su versión con memoria se presenta en la siguiente página.

A tener en cuenta:

- La nota asociada a la invocación de la operación `nuevoCurso` indica que el sistema mantendrá internamente una referencia (se puede pensar como un puntero en C++) al curso recién creado. Notar que se mantiene una referencia a la instancia y no su identificador.
- La operación `nuevoEstudiante` no cambia respecto a la de la parte a), pero las operaciones `inscribir` y `darCantInscriptos` ahora no reciben el parámetro `codC`,

dado que actúan sobre la instancia de curso recordada por el sistema. De esta forma, para cada estudiante que se inscribe en el curso (dentro del loop de este CU), el sistema no debe buscar nuevamente a la instancia del curso a partir de su identificador, lo que impacta positivamente en la eficiencia. Notar que si se recuerda el identificador del curso en lugar de la instancia, no hay mejora en la eficiencia pues para cada inscripción el curso debe buscarse nuevamente.

- Cuando se utiliza memoria en un DSS, sus operaciones del sistema son menos reutilizables en los DSS de otros CU. Por ejemplo, las operaciones `inscribir` y `darCantInscriptos` asumen que hay una instancia de curso en la memoria del sistema, por lo tanto no son reutilizables en otros CU donde no haya un curso en la memoria. Dicho lo anterior, si es necesario se podrían tener dos versiones de las operaciones: la que es eficiente y poco reutilizable (que asume memoria) y la que es menos eficiente pero reutilizable (que no asume memoria).
- Cuando finaliza el CU se asume que todos los identificadores locales dejan de existir, tal como ocurre en cualquier procedimiento o función de un lenguaje de programación. Para este DSS particular, esto incluye a la variable numE (que está almacenada del lado del actor) y a la referencia al curso recordado (que está almacenada del lado del sistema). Pero la instancia del curso creado, así como las inscripciones, permanecen almacenadas en el sistema y estarán disponibles en otros casos de uso. De la misma forma, el sistema debe registrar los números de estudiantes utilizados, para poder generar los nuevos al momento de ingresarse un nuevo estudiante. Esto también persiste entre los diferentes casos de uso.



c)

Primero haremos los contratos para la versión sin memoria. Haremos el contrato completo para `nuevoCurso` y `nuevoEstudiante`. Para las restantes operaciones haremos solo la parte correspondiente a las pre y post condiciones.

Firma	`nuevoCurso(nomC:String; codC:Integer)`
Parámetros	`nomC`: Nombre del nuevo curso. `codC`: Código del nuevo curso.
Responsabilidades	Crear un nuevo curso en el sistema a partir de los datos proporcionados.
Referencias cruzadas	Alta de curso con estudiantes inscriptos.

Salida	No aplica.
Precondiciones	- No existe en el sistema una instancia de `Curso` cuyo atributo `codigo` coincide con `codC`.
Postcondiciones	- Existe en el sistema una nueva instancia de `Curso` cuyo atributo `codigo` tiene el valor `codC` y cuyo atributo `nombre` tiene el valor `nomC`.

Firma	`nuevoEstudiante(ciE:String; nomE:String): Integer`
Parámetros	`ciE`: Cédula del nuevo estudiante. `nomE`: Nombre del nuevo estudiante.
Responsabilidades	Crear un nuevo estudiante en el sistema a partir de los datos proporcionados y devolver su número identificador en el sistema.
Referencias cruzadas	Alta de curso con estudiantes inscriptos.
Salida	Devuelve un valor que coincide con el atributo `nro` de la instancia de `Estudiante` recién creada.
Precondiciones	- No existe en el sistema una instancia de `Estudiante` cuyo atributo `CI` coincide con `ciE`.
Postcondiciones	- Existe en el sistema una nueva instancia de `Estudiante` cuyo atributo `CI` tiene el valor `ciE`, cuyo atributo `nombre` tiene el valor `nomE` y cuyo atributo `nro` tiene un valor generado por el sistema, diferente al de todas las demás instancias de la clase `Estudiante`.

Observaciones:

- La firma de cada operación debe corresponder con la del DSS.
- Los parámetros deben corresponder con los de la firma de la operación.
- Las responsabilidades son una descripción a alto nivel, no necesariamente en términos del modelo.
- Las referencias cruzadas deben incluir a todos los casos de uso donde se utiliza la operación.
- La salida aplica solamente en funciones (no en procedimientos) y debe estar expresada en términos del modelo, es decir, parámetros de las operaciones y elementos del modelo de dominio.
- Las precondiciones también deben estar expresadas en términos del modelo y deben ser lo más precisas posible. Lo mismo aplica para las postcondiciones. Notar que cuando se crean instancias o links, se debe indicar que son nuevos.

Firma	`inscribir(numE:Integer; codC:Integer)`
Precondiciones	- Existe en el sistema una instancia de `Estudiante` cuyo atributo `nro` coincide con `numE`. - Existe en el sistema una instancia de `Curso` cuyo atributo `codigo` coincide con `codC`. - No existe en el sistema un link de `inscripto` entre la instancia de `Estudiante` cuyo atributo `nro` coincide con `numE` y la instancia de `Curso` cuyo atributo `codigo` coincide con `codC`.
Postcondiciones	- Existe en el sistema un nuevo link de `inscripto` entre la instancia de `Estudiante` cuyo atributo `nro` coincide con `numE` y la instancia de `Curso` cuyo atributo `codigo` coincide con `codC`.

Firma	`darCantInscriptos(codC:Integer):Integer`
Salida	Devuelve la cantidad de links de `inscripto` entre la instancia de `Curso` cuyo atributo `codigo` coincide con `codC` e instancias de clase `Estudiante`.
Precondiciones	- Existe en el sistema una instancia de `Curso` cuyo atributo `codigo` coincide con `codC`.

Importante: Observar que para expresar las precondiciones, postcondiciones y salidas, debe tenerse en cuenta las mismas observaciones realizadas para la expresión de las restricciones del modelo de dominio (ver notas de la clase 3, página 3). En particular:

- Para la redacción tratamos de ser lo más precisos posibles, con las limitaciones del lenguaje natural.
- Se puede escribir algo más sencillo, por ejemplo, sin nombrar las asociaciones, dado que en algunos casos no hay más de una asociación entre cualquier par de clases.
- Se debe tener cuidado con la simplificación en la escritura de las expresiones. En particular, se deben evitar redacciones que no hacen referencia a ningún componente del modelo (clase, asociación, parámetros de operaciones) y redacciones que se parezcan demasiado a una parte de la letra del problema.

Versión de los contratos de las operaciones del DSS con memoria (se resaltan los cambios respecto a las versiones sin memoria):

Firma	`nuevoCurso(nomC:String; codC:Integer)`
Precondiciones	- No existe en el sistema una instancia de `Curso` cuyo atributo `codigo` coincide con `codC`.
Postcondiciones	- Existe en el sistema una nueva instancia de `Curso` cuyo atributo `codigo` tiene el valor `codC` y cuyo atributo `nombre` tiene el valor `nomC`. - El sistema recuerda la nueva instancia de `Curso`.

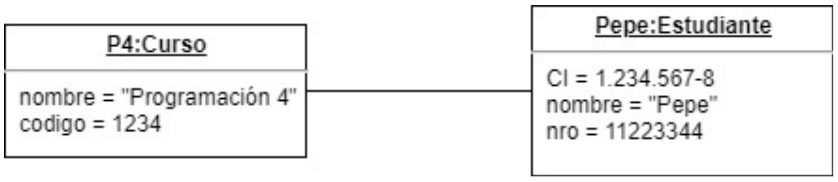
Firma	`inscribir(numE:Integer)`
Precondiciones	- Existe en el sistema una instancia de `Estudiante` cuyo atributo `nro` coincide con `numE`. - Existe en el sistema una instancia de `Curso` recordada. - No existe en el sistema un link de `inscripto` entre la instancia de `Estudiante` cuyo atributo `nro` coincide con `numE` y la instancia de `Curso` recordada.
Postcondiciones	- Existe en el sistema un nuevo link de `inscripto` entre la instancia de `Estudiante` cuyo atributo `nro` coincide con `numE` y la instancia de `Curso` recordada.

Firma	`darCantInscriptos():Integer`
Salida	Devuelve la cantidad de links de `inscripto` entre la instancia de `Curso` recordada e instancias de clase `Estudiante`.
Precondiciones	- Existe en el sistema una instancia de `Curso` recordada.

Notar que ahora se hace referencia también a las instancias recordadas en el sistema.

d)

Para esta parte utilizamos *snapshots*, entendidas como fotos del estado del sistema. En una snapshot participan entidades y links, que son instancias de las clases y asociaciones del modelo, respectivamente.



Las cajas representan instancias. En la parte superior se debe especificar el nombre de la clase precedido por : y opcionalmente el nombre de la instancia. En la parte inferior, se debe especificar el valor para cada uno de los atributos de la clase. El estado del sistema evoluciona de la siguiente manera:

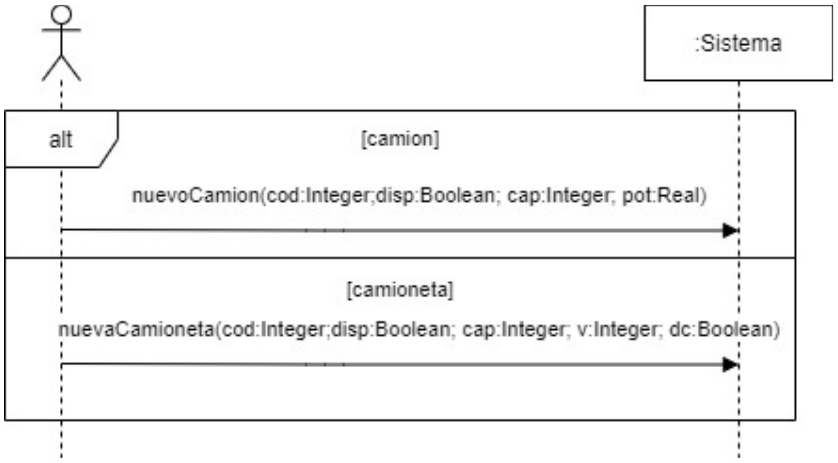
- Inicialmente estará vacío.
- Luego de la ejecución de `nuevoCurso`, contendrá la instancia P4.
- Luego de la ejecución de `nuevoEstudiante`, además contendrá la instancia Pepe.
- Luego de la ejecución de `inscribir`, además contendrá el link de la asociación `inscripto`, entre P4 y Pepe.

Notar que esta instancia del sistema cumple con las restricciones, en particular con las multiplicidades de la asociación `inscripto`.

Ejercicio 4:

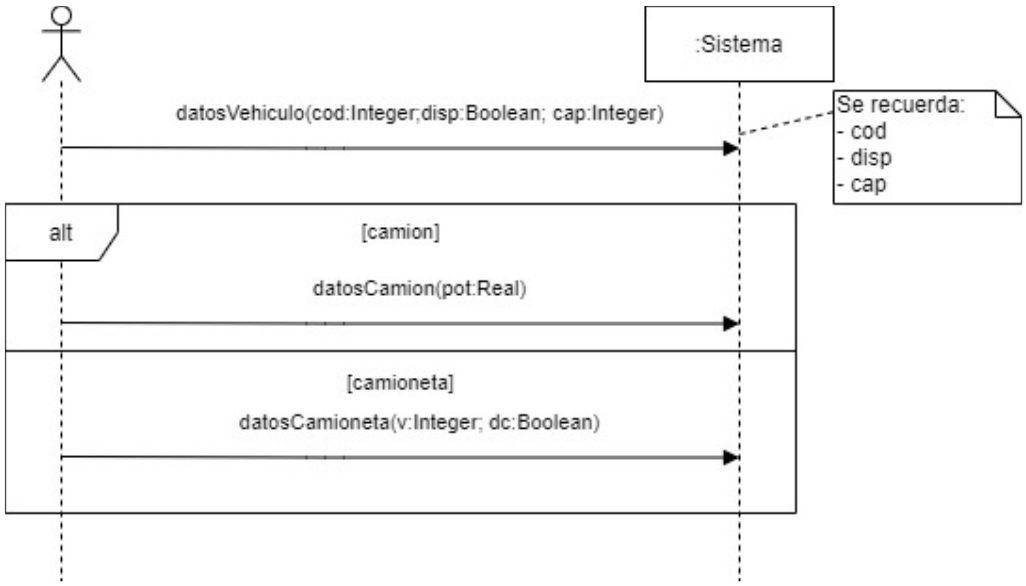
Veamos tres alternativas para modelar lo que pide la letra del problema.

Opción 1: Una operación de ingreso por cada clase concreta.



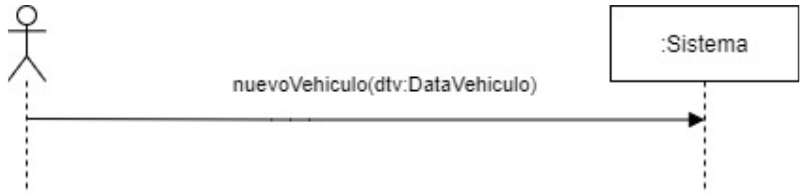
Notar que los valores de las guardas del alt (camion, camioneta, son expresiones booleanas) serán determinados en base lo que seleccione el usuario desde la interfase. Según lo que seleccione, se le pedirán datos de un camión o de una camioneta y se invocará la operación del sistema correspondiente.

Opción 2: Una operación común para los datos del vehículo y una para cada clase concreta, con los datos específicos.



Notar que esta opción permite que al principio se le pida al usuario desde la interfase solamente los datos del vehículo, luego se le pregunte de qué tipo es el vehículo y dependiendo de lo que seleccione, se le pedirá los datos de camión o de camioneta. En este DSS se utiliza memoria para recordar los datos de vehículo (se recuerdan los datos, no se crea una instancia de Vehiculo, dado que es una clase abstracta). Cuando el sistema reciba la invocación de las operaciones específicas de cada clase (Camion o Camioneta), sabrá de qué clase es el objeto que se debe crear.

Opción 3: Una sola operación con datavalue polimórfico.



En este caso se debe crear una jerarquía de datatypes, replicando exactamente la estructura de clases Vehiculo, Camion y Camioneta. La interacción con el usuario mediante la interfase será similar a las opciones 1 y 2. Cuando la capa de presentación sepa si el usuario va ingresar un camión o una camioneta, pide los datos específicos y crea un datavalue de tipo DataCamion o DataCamioneta. Cuando el sistema recibe la invocación a esta operación, deberá identificar cual es el tipo concreto del datavalue, para saber si debe crear una instancia de Camion o de Camioneta.

Resumen: Las tres opciones son válidas y las tres son relativamente flexibles ante el agregado de un nuevo tipo de vehículo, diferente a camión y camioneta.