FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 1 – Smalltalk

[7507/9502] Algoritmos y Programación III
Curso 2
Segundo Cuatrimestre de 2022

Alumno:	DIEM, Walter Gabriel
Número de padrón:	105618
Email:	wdiem@fi.uba.ar

Índice

1. Introducción

El presente informe reúne la documentación de la solución del primer trabajo práctico de la materia Algoritmos y Programación III, que consiste en desarrollar una aplicación que sirva como un sistema de penales de un partido de fútbol. Fue escrito en la implementación de Smalltalk de Pharo, versión 9.0, y se implementaron principios de diseño de programación orientada a objetos en la construcción de la solución.

2. Supuestos

A continuación se detallan algunas de las consideraciones hechas sobre los requerimientos para poder desarrollar el trabajo:

- De los comentarios de las pruebas-consigna se entendieron las siguientes fórmulas para el cálculo de la calificación de cada jugador:

  Para un JugadorTitular:

$$calificacion = (habilidad^2 * cansancio * coeficienteExperiencia) + estadoMoral$$

  Para un JugadorSuplente:

$$calificacion = (habilidad^{2+calidadDeSuplente} * coeficienteExperiencia) + estadoMoral$$

- No se puede crear un partido con un formato inadecuado, es decir, que no respete la estructura "*nombreEquipo1 - nombreEquipo2*".
- No se puede registrar un jugador en un equipo que no sea uno de los que "están jugando".
- No se puede elegir el pateador de un equipo que no tenga registrados jugadores.
- Es condición necesaria que una instancia de *AlgoPenales* haya recibido y procesado un partido.
- El retorno de *elegirPateadorDe* es un string.

3. Modelo de dominio

Se utilizarán los términos “dominio del problema” y “dominio de la realidad” indistintamente, ya que es razonable asumir que en este caso el problema en cuestión intenta modelar una situación de la realidad. Las asunciones del modelo de dominio vienen de interpretar el set de pruebas proveído por la cátedra y entender los requerimientos pedidos.
Dentro del modelo de dominio del problema se puede entender que existe un “objeto coordinador del partido” representado en el dominio de la solución por instancias de la clase *AlgoPenales*. El problema plantea la existencia de equipos, y hace entender que cada uno tiene relacionado a un set de jugadores, suplentes y titulares, cuyo reflejo en el modelo de la solución es una colección de instancias de clase *Jugador*, alojadas en una instancia de clase *Equipo*.
Si bien en un caso de un partido de fútbol real hay un marcador de goles, donde se puede deducir cuál equipo está ganando, perdiendo o si hay empate, eso escapa al dominio del problema que concierne este trabajo, esa restricción se refleja en que no se hizo falta un marcador de goles para hacer funcionar el trabajo práctico acorde a los requerimientos. Con decir esto, aclaro que lo que se está modelando no es un partido de fútbol, o penales de uno, completo, sino una versión reducida que se adapte al conjunto de pruebas. Otro ejemplo de esto es cómo afecta el resultado a un jugador, en el problema se entiende que el que estén ganando, perdiendo o empatando, afecta al equipo entero de la misma manera, siendo que en la realidad este efecto puede diferir de jugador a jugador.

4. Diagramas de clase

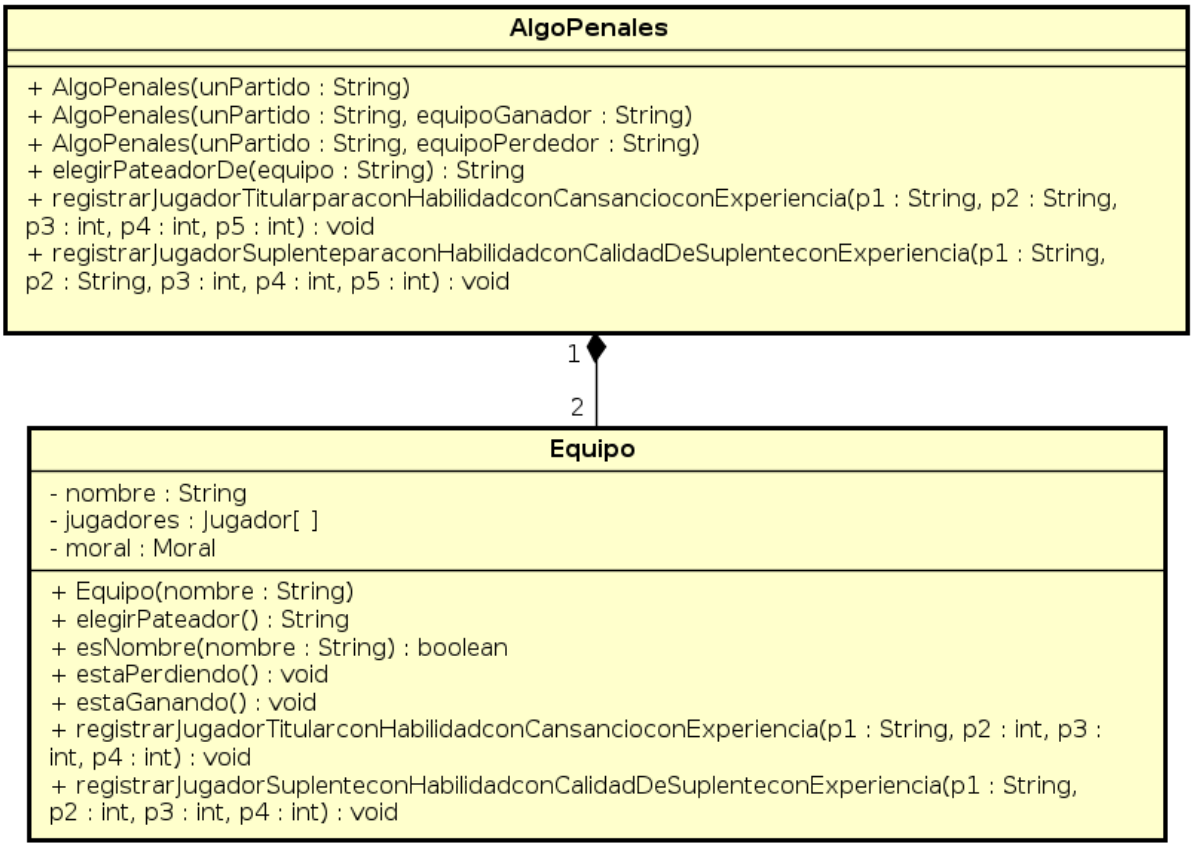


Figura 1. Composición de AlgoPenales con Equipo.

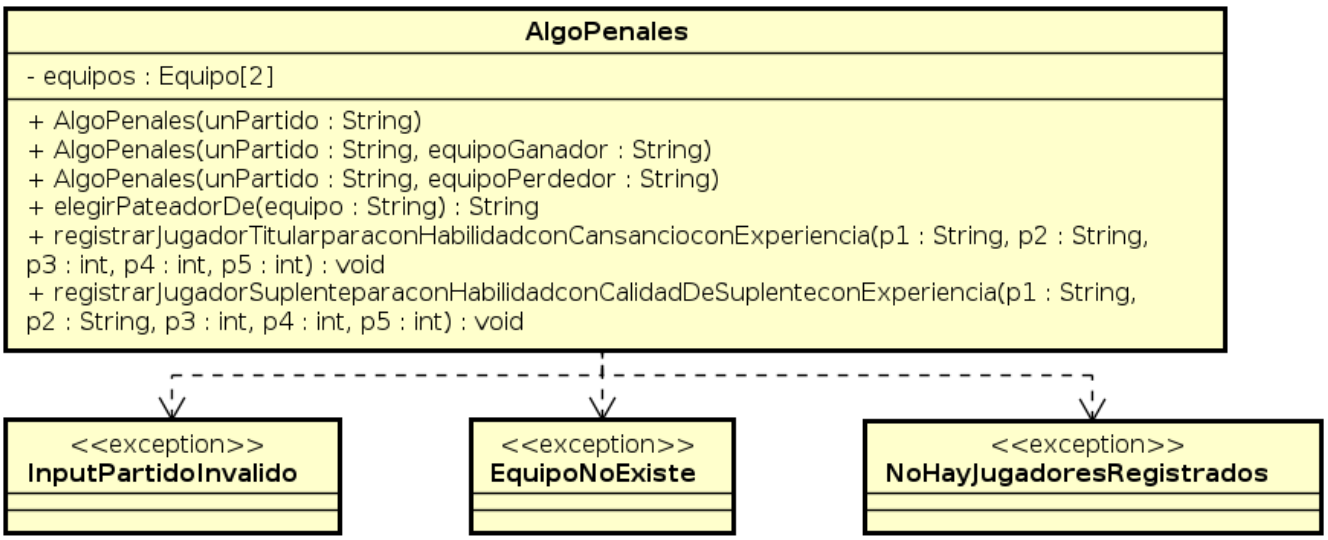


Figura 2. Excepciones de AlgoPenales

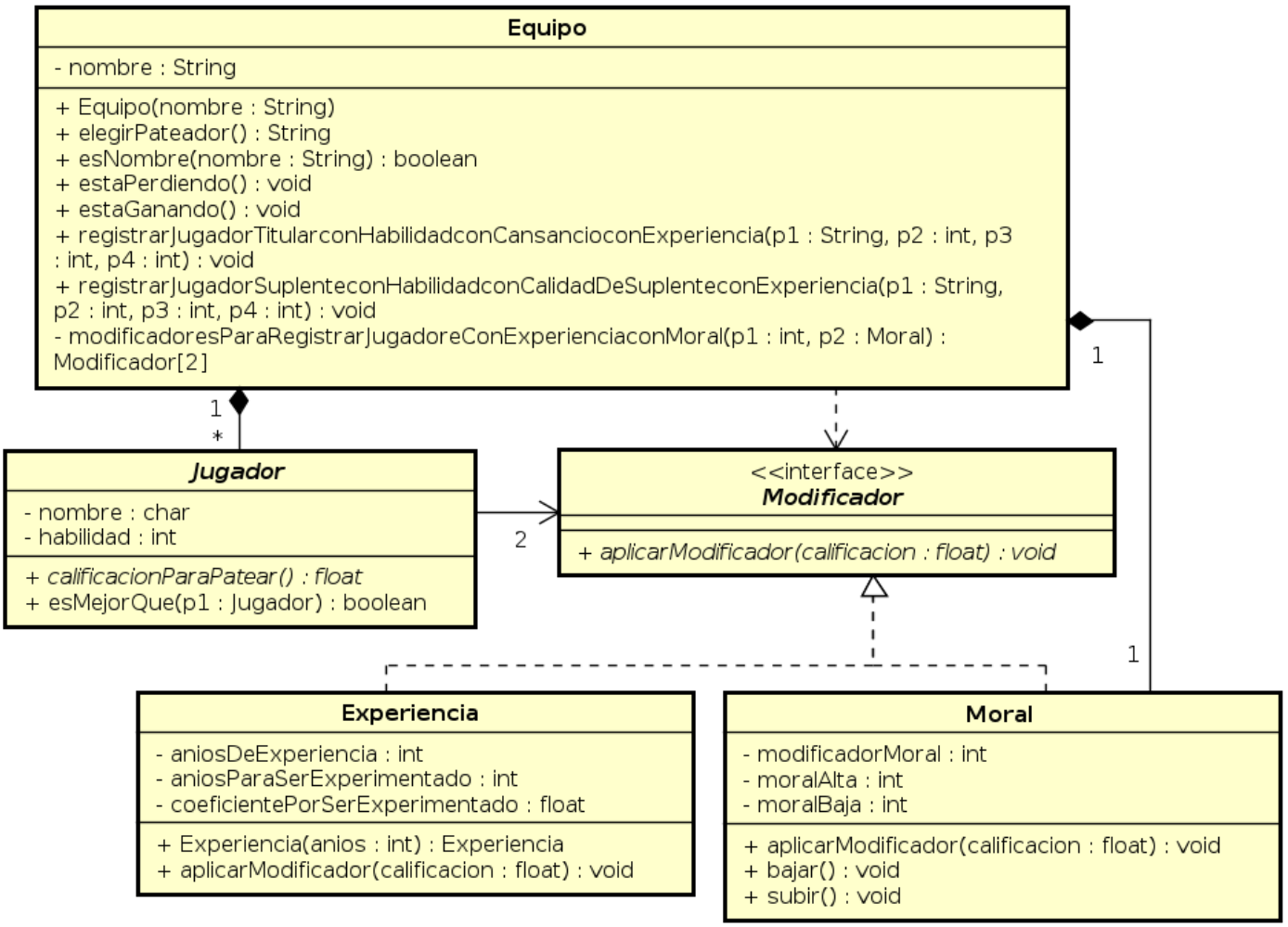


Figura 3. Relaciones de la clase Equipo.

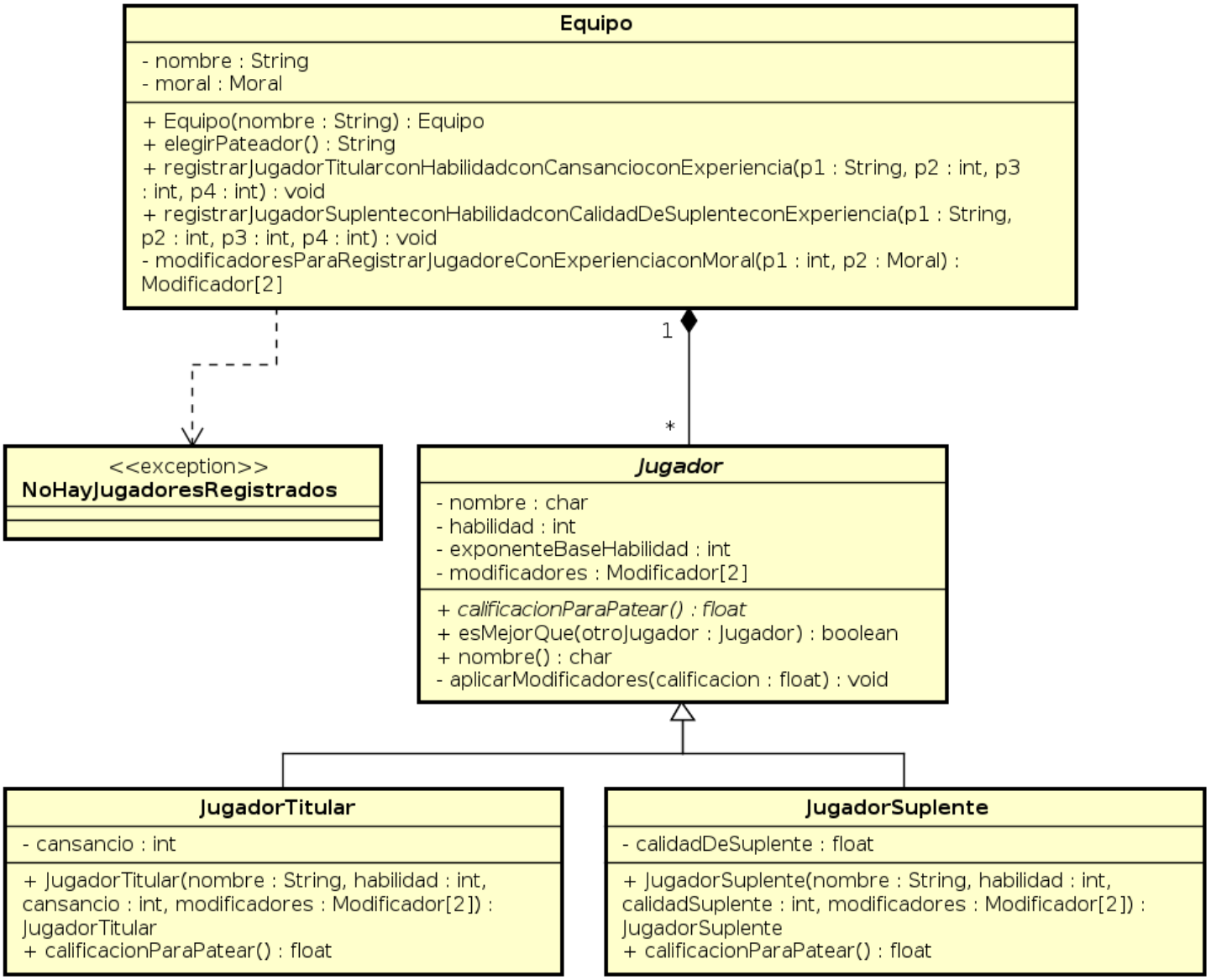


Figura 4. Relaciones de Equipo con Jugador.

5. Detalles de implementación

El principio guía que utilicé para el desarrollo de la solución fue el de *Tell, don't ask*. Me sirvió para cimentar la metodología del trabajo con objetos, dándole comportamiento a las clases y ordenándoles que hagan algo, en lugar de la errónea práctica de pedir información sobre el estado interno para luego tomar decisiones con ella desde otro lugar. A su vez fue útil tener en mente la ley de Demeter, bajo el mantra "que el objeto no hable con extraños" y manteniendo bajo control con quién puede comunicarse el objeto acorde a esta "ley".

Teniendo ese principio como punto de partida, se me hizo más fácil respetar los pilares de la POO, es decir: la abstracción, el encapsulamiento, el polimorfismo, la delegación y la herencia.
Un lugar donde encontré óptimo el uso del polimorfismo fue al notar que tanto la condición de experiencia como la condición de si el equipo ganaba o perdía modificaban la calificación del jugador, por lo que implementé el mismo método sintácticamente hablando, para dos clases polimórficas que creé: Experiencia y Moral, pudiendo agrupar así instancias de estas clases en una colección para luego ejecutar comportamiento polimórfico, esto lo expresé en el diagrama de clases como la interfaz "Modificador", aunque por el chequeo de tipos de Smalltalk no hizo falta implementar código para que fuera posible.
La delegación fue natural entendiendo quién era el *Information Expert*, es decir, quién era el objeto más capacitado y quien debería tener cierta responsabilidad, por lo que la utilicé en múltiples ocasiones.
La herencia la utilicé en el caso de mis clases JugadorTitular y JugadorSuplente, que son hijas de la clase abstracta Jugador. Se cumple la relación "es un", esto es: "un JugadorTitular es un Jugador" y "un JugadorSuplente es un Jugador". Ambos comparten comportamiento, código y un contrato que pude abstraer mediante la herencia, evitando así también repetición de código.
Finalmente, me gustaría dejar una nota sobre las pruebas, agrupé al principio las pruebas relacionadas con excepciones y luego casos "borde", como tener un sólo jugador. Se notarán pruebas con la leyenda "de la derecha", esto es así ya que noté que tenía errores inicialmente con que los equipos funcionaran correctamente, consecuentemente me mantuve probando los casos "del equipo de la derecha" para solidificar el buen comportamiento de la clase.

6. Excepciones

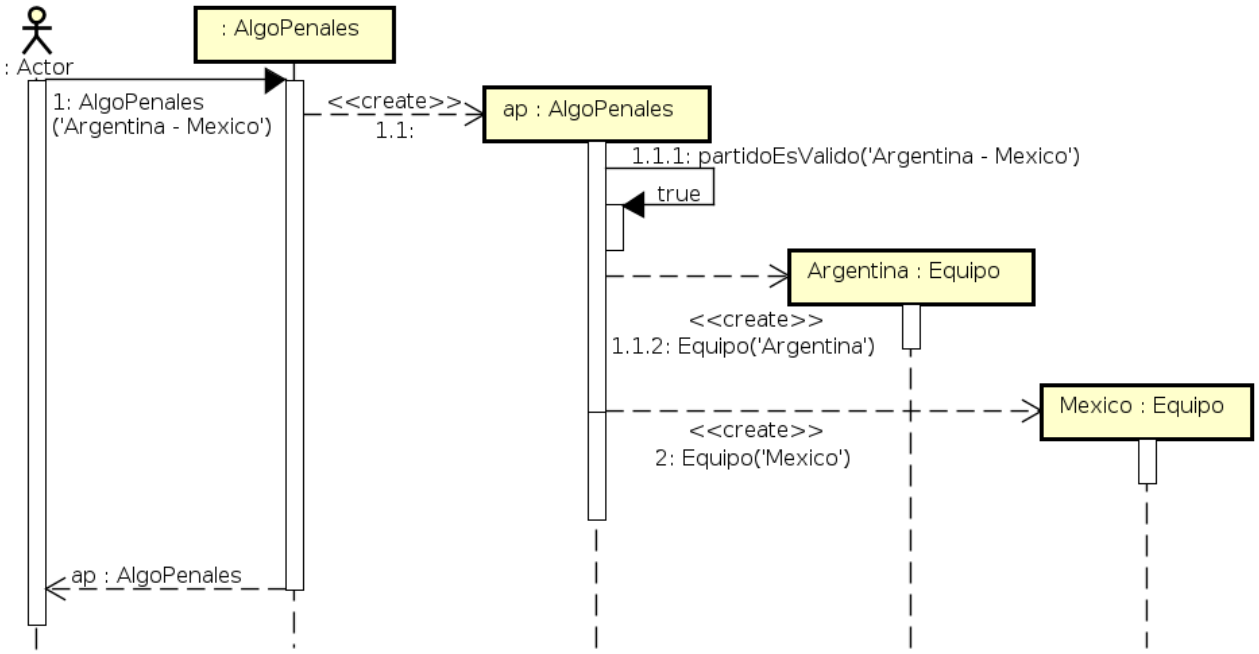
InputPartidoInvalido: Se lanza cuando el formato del string del partido recibido como parámetro para la creación de una instancia de *AlgoPenales* es incorrecto, esto es, no cumple con la estructura "*equipo1* - *equipo2*". Su razón de ser está dada por la necesidad de extraer dos substrings que serán los nombres de los equipos, fundamentales para la creación de los equipos.

EquipoNoExiste: Se lanza cuando el nombre del equipo solicitado no es el nombre de uno de los equipos que fueron cargados en *AlgoPenales*. Su razón de ser es que no se puede operar sobre un equipo solicitado que no existe.

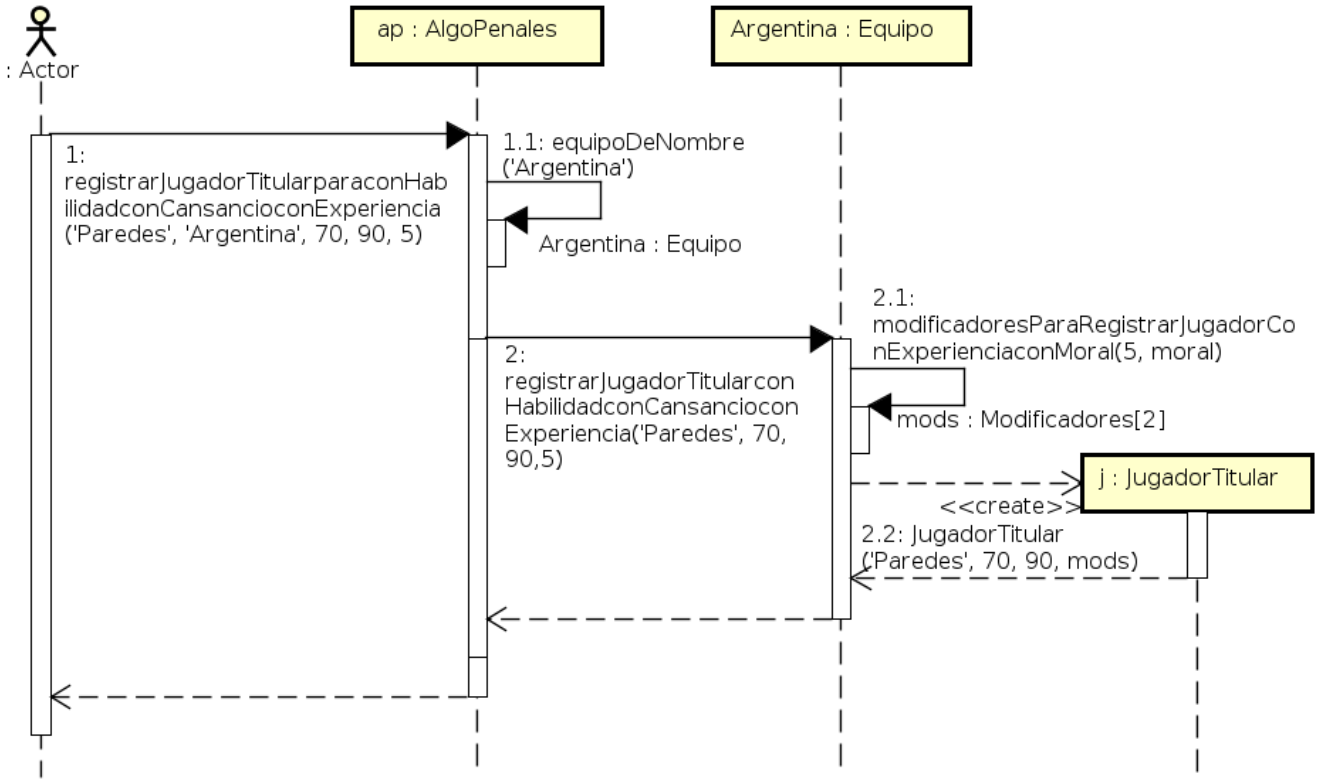
NoHayJugadoresRegistrados: Se lanza cuando se pide elegir un pateador de un equipo que no tiene jugadores. Su razón de ser es que no se puede elegir un jugador para que patee de un equipo que no tiene jugadores, fue necesario primero registrar algún jugador.

7. Diagramas de secuencia

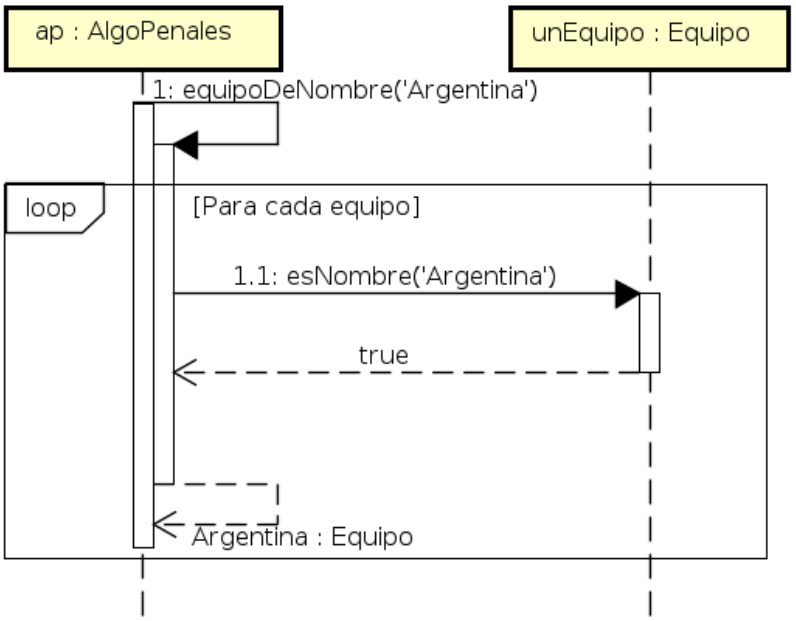
Para el primer diagrama de secuencia se esquematizarán las acciones que suceden en el caso de uso presente dentro del test01 proveído por la cátedra, de nombre *test01UnJugadorConCiertoCansancioYCiertaHabilidadEsElEncargadoDePatearElPenal.*



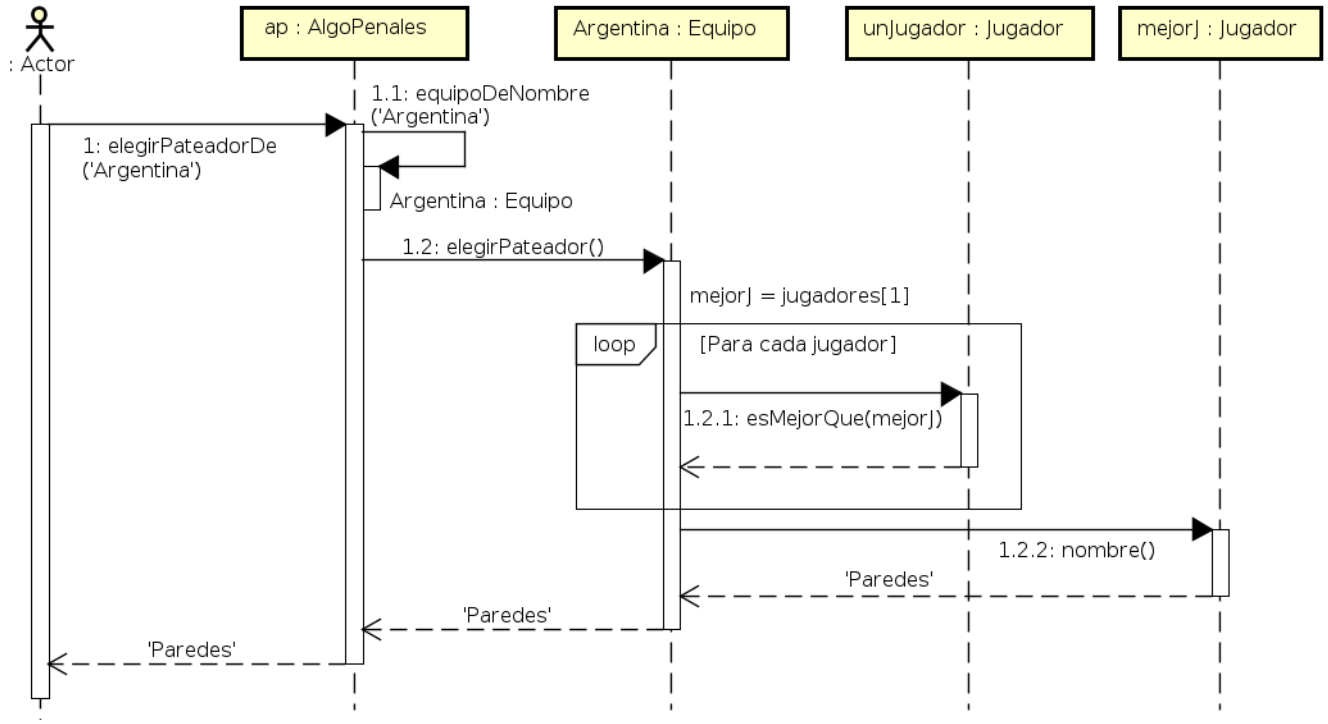
Luego:



Hago detalle en este último mensaje 1.1, lo expando para revelar el comportamiento interno del método:

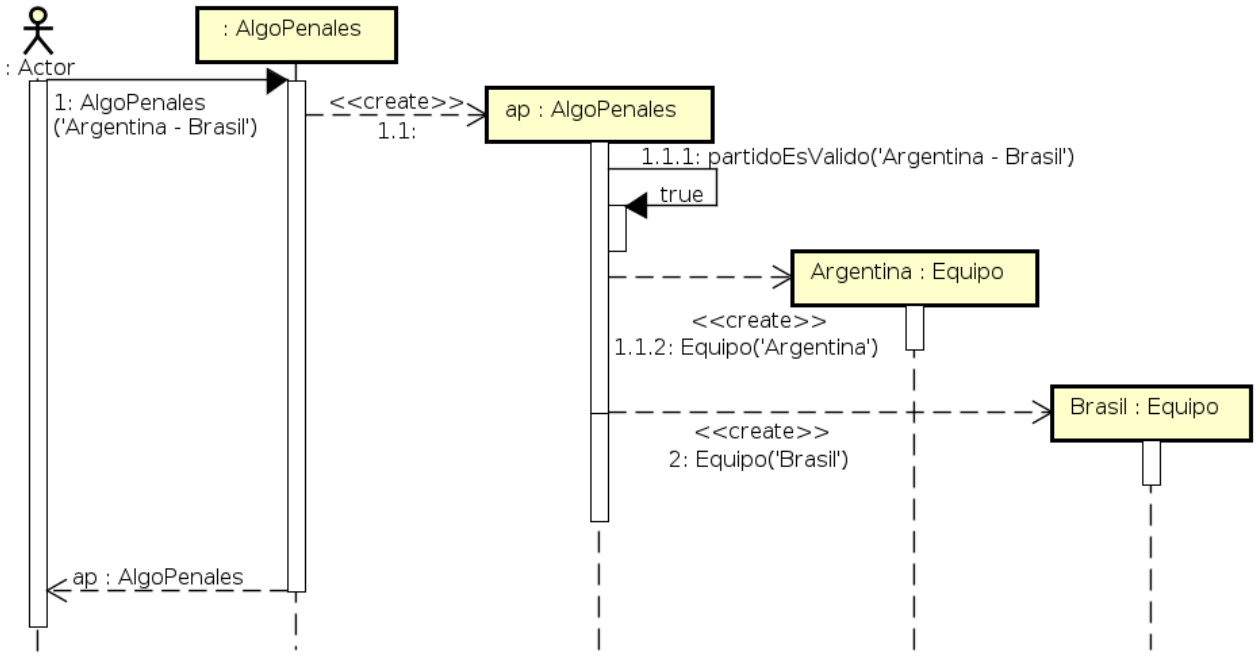


Este es el comportamiento que el método AlgoPenales>>equipoDeNombre tiene cada vez que es invocado, de ahora en más no se desglosará el mismo en un loop dedicado. Siguiendo:

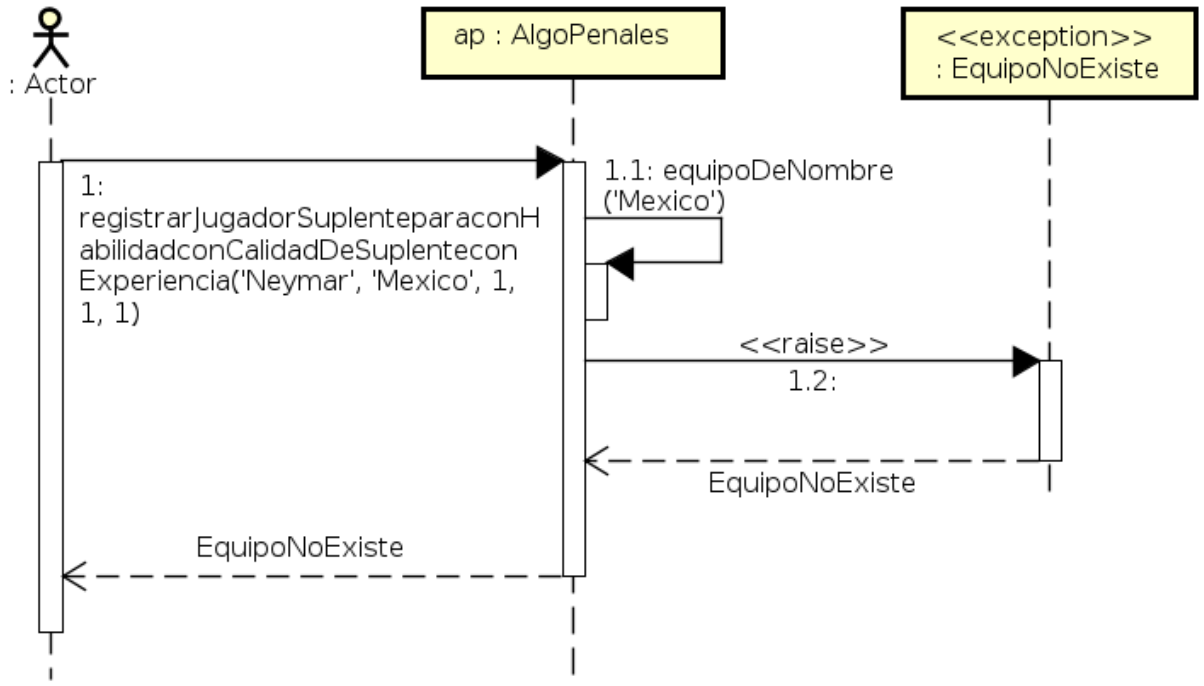


Y así es como se termina el caso de uso.

Para el siguiente caso de uso, se presenta un "caso no feliz", siguiendo el caso de uso del test19 de mi set de pruebas de AlgoPenalesTests, de nombre *test19NoPuedoRegistrarUnJugadorSuplenteEnEquipoInexistenteElevaExcepcionEquipoNoExiste*.



Siguiendo:



Terminando así la ejecución del código con una excepción del tipo EquipoNoExiste.