Trabajo Práctico 1 — Smalltalk

[7507/9502] Algoritmos y Programación III
Curso X
Primer cuatrimestre de 2018

Alumno:	Valentino Ceniceros
Número de padrón:	111054
Email:	vceniceros@fi.uba.ar

Índice

1. Introducción

El presente informe reune la documentación de la solución del primer trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar una aplicación de un sistema de una agencia de viajes en Pharo utilizando los conceptos del paradigma de la orientación a objetos vistos hasta ahora en el curso.

2. Supuestos

Al ser una consigna muy entretenida pero lo suficientemente ambigua como para que no sea paso 1, 2, 3 se me ocurrieron múltiples supuestos a lo largo del tp, aquí enumeraré los principales.

- La clase algo rastreador es concreta.
- Esta tiene entre sus atributos un modelo de rastreador.
- El ki obtenido por la clase no puede ser negativo.
- Criterio es una colección de distintos rastreos (lecturas).
- No es el mismo el peleador que recibe algo rastreador que el que se registra en el rastreo.
- La mejor manera de implementar polimorfismo va a ser mediante un diccionario, porque el usuario solo pone una clave y este devuelve como valor la clase buscada.
- Seguido de lo anterior, es evidente que esto debe ser una clase completamente aparte.

3. Modelo de dominio

3.1. Pilares utilizados

Los pilares que utilicé fueron:

3.2. Encapsulamiento

Este consiste en que los atributos y comportamientos internos de la clase sean utilizados y entendidos solamente por la clase misma. Esto trae como ventaja la independencia de una clase respecto a otras en cuanto al control de los estados de la misma. Si tuviera atributos de una clase siendo modificados por otra externa, esta perdería el control sobre sí misma, lo que haría el código menos mantenible, ya que un cambio en una clase podría romper otra. Este pilar fue la base fundamental de todo mi código. Cada vez que quise acceder a un atributo de una clase desde fuera de ella, terminé creando métodos encargados de manejar ese comportamiento o estado.

3.3. Herencia

Este consiste en que una clase comparta comportamientos y estados de otra con leves o nulas modificaciones en cuanto a la implementación, basándose en la relación de .^{es}to es unçomo pilar fundamental. Esto permite tener múltiples clases que hagan lo mismo sin necesidad de tener una implementación completamente idéntica en otra clase, evitando así repetir código. La mayor desventaja de la herencia es que es una estructura muy rígida que limita la cantidad de cosas que se pueden hacer en una clase heredera, sin mencionar lo problemático que puede ser que un cambio en una clase padre termine modificando y/o rompiendo a sus clases hijas. Este pilar lo implementé en BuscarPorDiccionario, Criterio, Transformación y Rastreador, simplificando mucho la implementación de múltiples partes de mi código.

3.4. Polimorfismo

El polimorfismo, en principio, permite que varios objetos reaccionen al mismo mensaje. Suena bastante trivial, pero cuando se aplica correctamente, permite crear un código muy escalable donde la reacción de una clase a los mensajes recibidos se puede definir perfectamente con otra clase en vez de tener que poner una estructura de condicionales que maneje cómo se debe comportar esta según qué parámetro reciba como mensaje. El mayor problema que tiene es que, si se abusa de él, se termina haciendo un código prácticamente ilegible, donde el ocultamiento de implementación llega al grado de perder el sentido. Este concepto lo apliqué, por ejemplo, en Transformación, donde en vez de poner un "si Transformación entonces ki es igual a", simplemente le pasé el ki a la Transformación para que esta calcule el ki según qué transformación sea.

3.5. Delegación

Esta consiste en que mediante los mensajes una clase le designa a otra una tarea, lo cual es excelente para simplificar el código, ya que en vez de que una clase haga demasiadas cosas, esta solo tiene que darle esa tarea a otra clase. La mayor desventaja de delegar es que, mal implementado, puede terminar rompiendo el encapsulamiento. En mi código, lo implementé, por ejemplo, en transformar ki, donde para calcular el ki de un peleador, este solo le daba su ki base a Transformación para que esta se encargue de calcular el ki real del mismo.

3.6. Abstracción

Este pilar permite ocultar los detalles de la implementación al llevar comportamientos complejos a simplemente un nombre, referenciando un comportamiento del código. Esto permite un código no solo legible sino también muy escalable, ya que una clase abstracta puede implementarse en una infinita cantidad de instancias. Todas las clases de mi código que son heredables son abstractas, permitiéndome así simplificar muchas funciones de mi código, como, por ejemplo, Criterio.

3.7. Delegación sobre Herencia

La herencia me permitió, por ejemplo, simplificar y polimorfar Transformación, haciendo que multiplicar el ki dependa exclusivamente de dicho atributo de Transformación. También utilicé delegación en Transformar, ya que un peleador delega la tarea de transformar su ki a su transformacion en cuestión.

4. Diagramas de clase

Este es el diagrama principal de como funciona el programa.

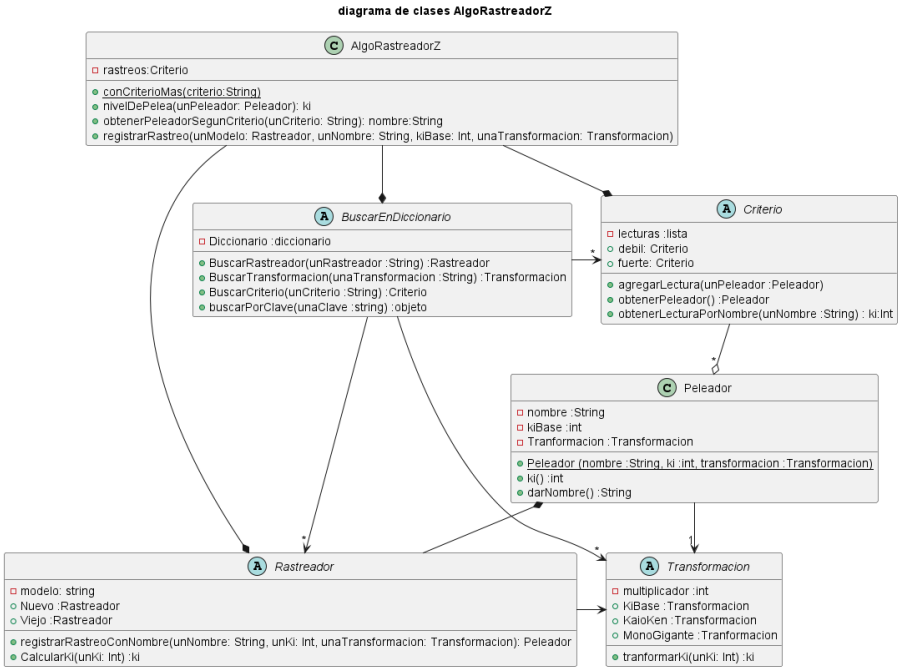

Figura 1: Diagrama del algo rastreador z.

Aca los diagramas de las herencias.

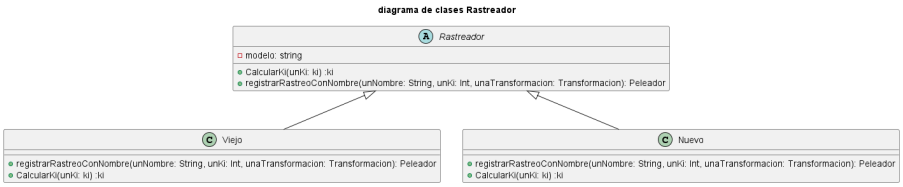

Figura 2: Diagrama de rastreador.

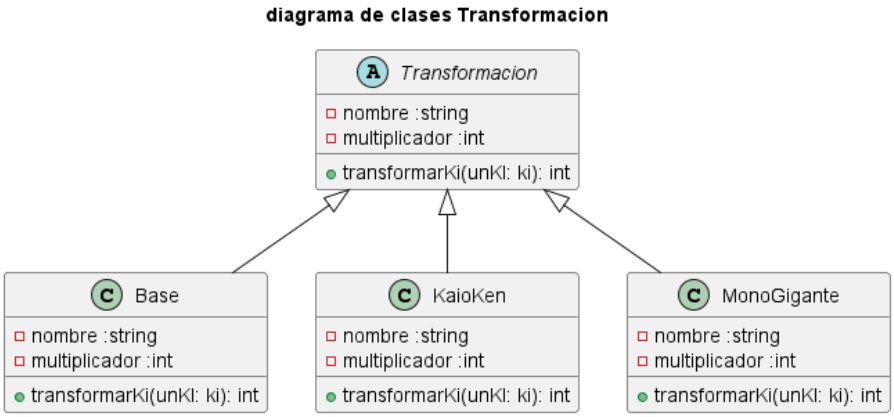

Figura 3: Diagrama de transformaciones.

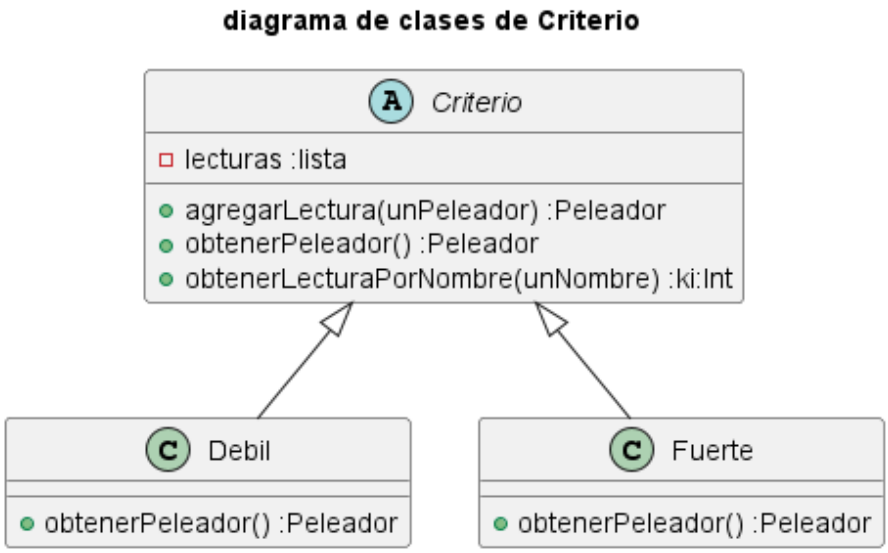

Figura 4: Diagrama de criterio.

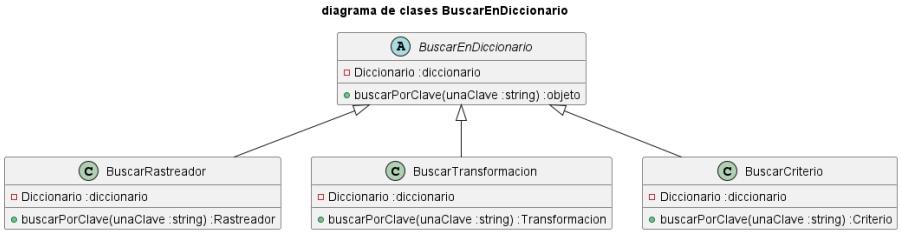

Figura 5: Diagrama del diccionario.

5. Detalles de implementación

5.1. implementacion de registrar peleador

el uso y delegacion de clases y polimorfismo me permitio el convertir una clase compleja como muchas interrelaciones como lo es registrar peleador con nombre ki base y transformacion en una clase que se ve asi.

```
registrarRastreoConModelo: unModelo DelPeleadorConNombre: unNombre KiBase: unKi yTransformacion:
|peleador  transformacion rastreadorUsado|

rastreadorUsado := seleccionarRastreador buscarPorClave: unModelo  .

transformacion := identificarTransformacion buscarPorClave: unaTransformacion  .

peleador := rastreadorUsado registrarRastroConNombre: unNombre  Ki: unKi  yTransformacion: transf

rastreos agregarLectura: peleador
```

6. Excepciones

Exception Basandonos en la logica de dragon ball un ki jamas puede ser menor a 0 ya que esta seria una suerte de energia de vida por lo cual no puede haber una menos vida por lo cual las variables encargadas de definir el ki tanto en peleador como en peleador registro no puede ser menores que 0.

Excepcion No se puede buscar una transformacion, criterio o rastreador no existentes por lo cual se debe enviar una excepciona al enviarles una clave invalida.

7. Diagramas de secuencia

De todos los metodos que componen a algo rastrador z me centre en los 3 principales: registrar peleador tanto cuando el rastreador es nuevo o viejo, nivel de pelea y obtener peleador segun criterio ya sea fuerte o debil.

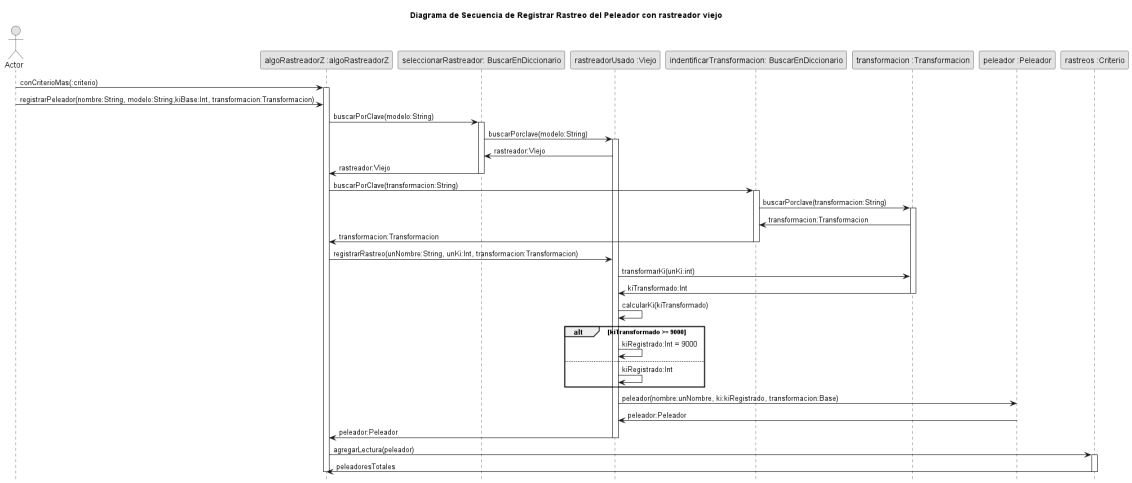

Figura 6: diagrama de secuencia de registrar peleador con rastreador viejo.

Como se puede observar en el diagrama, un Actor interactúa con el sistema .ªlgoRastreadorZ"para registrar un rastreo de un peleador.

El sistema inicia identificando la transformacion y seleccionando un rastreador.

Posteriormente, el sistema crea una instancia de rastreador asignandole un peleador con un nombre, ki y transformacion.

Como el rastreador es viejo si el ki es mayor a 9000 este se registrara como 9000.

Luego este peleador es guardado en rastreo.

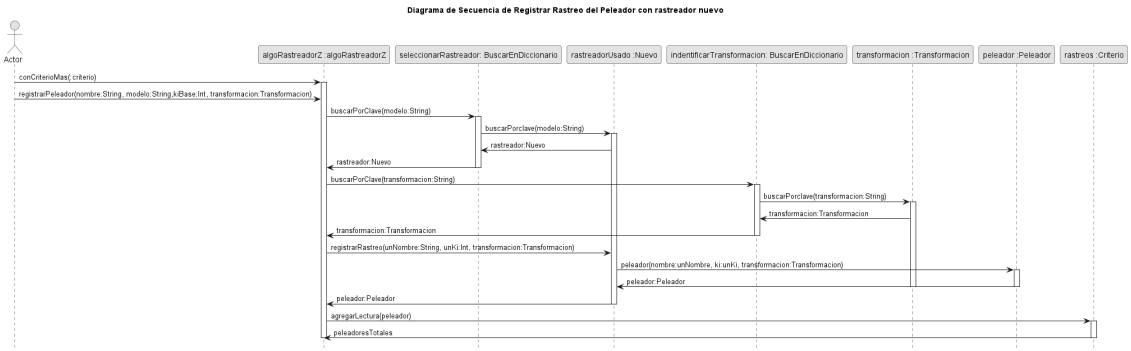

Figura 7: diagrama de secuencia de registrar peleador con rastreador nuevo.

Como se puede observar en el diagrama, un Actor interactúa con el sistema .ªlgoRastreadorZ"para registrar un rastreo de un peleador.

El sistema inicia identificando la transformacion y seleccionando un rastreador.

Posteriormente, el sistema crea una instancia de rastreador asignandole un peleador con un nombre, ki y transformacion.

Como el rastreador es nuevo el peleador se guarda tal cual es registrado.

Luego este peleador es guardado en rastreo.

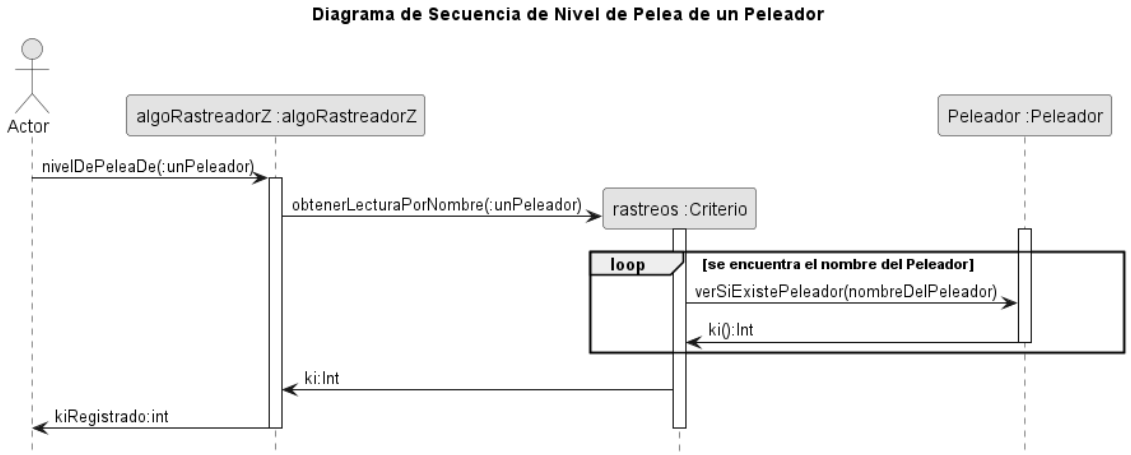

Figura 8: diagrama de secuencia de nivel de pelea .

Aca se describe cómo se determina el nivel de pelea de un peleador en el sistema .ªlgoRastreadorZ".

Un Actor solicita al sistema .ªlgoRastreadorZ.el nivel de pelea de un peleador específico.

El sistema busca dentro de rastreos a un peleador por su nombre y este le devuelve el peleador si es que existe.

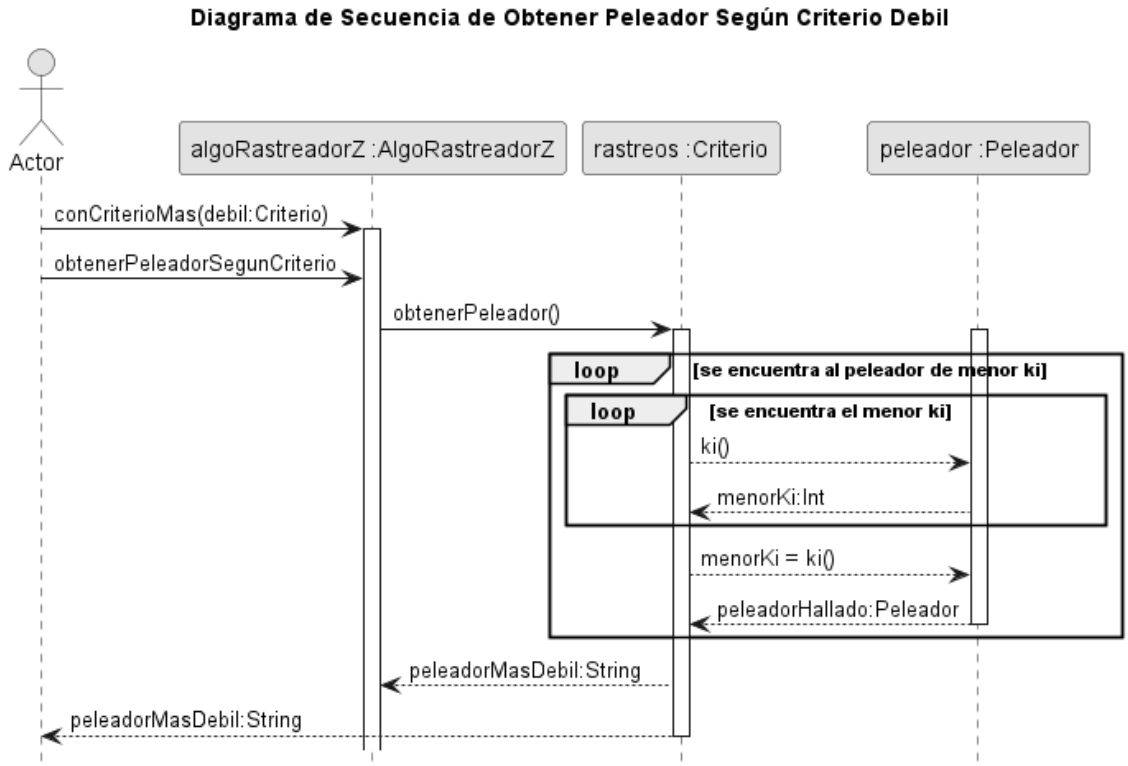

Figura 9: diagrama de obtener peleador segun criterio debil.

Por ultimo este diagrama de secuencia describe cómo el sistema .ªlgoRastreadorZ.ºbtiene un peleador según un criterio especificado por un Actor.

El Actor solicita al sistema .ªlgoRastreadorZ.ºbtener un peleador.

El sistema le envia el mensaje a rastreos de .ºbtener criterioz como este criterio es debil este retorna al peleador mas debil.

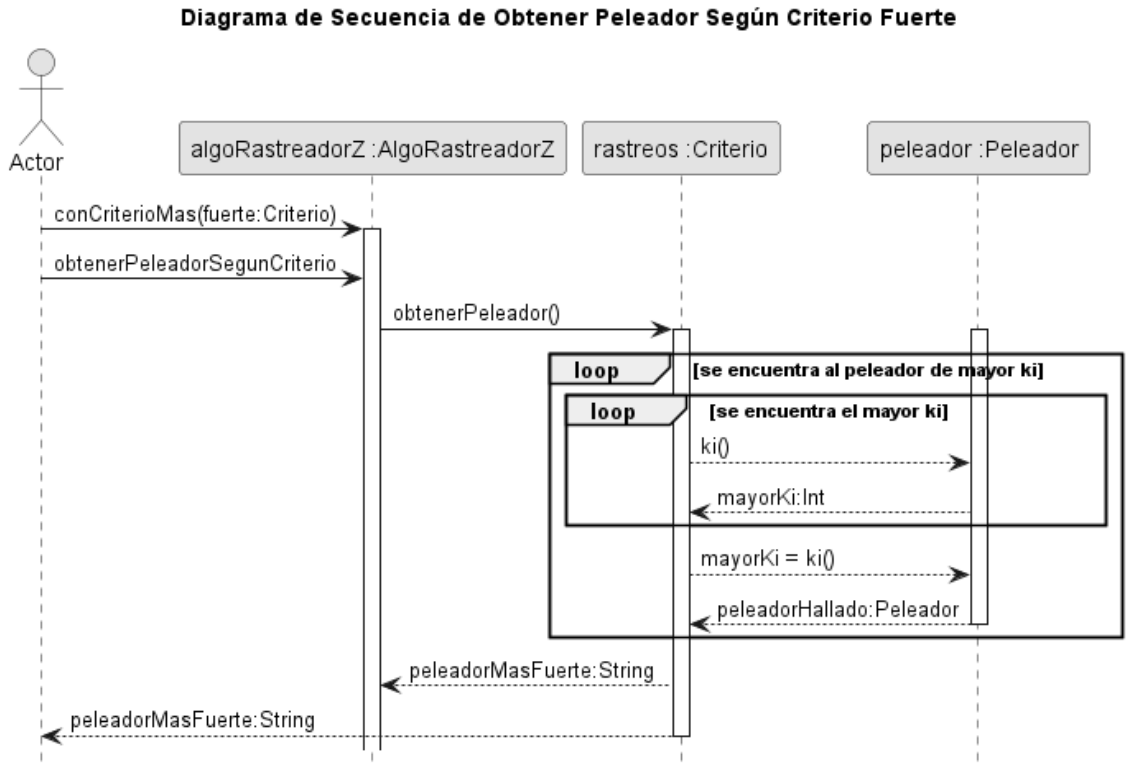

Figura 10: diagrama de obtener peleador segun criterio fuerte.

Este diagrama de secuencia describe cómo el sistema .ªlgoRastreadorZ.ºbtiene un peleador según un criterio especificado por un Actor.

El Actor solicita al sistema .ªlgoRastreadorZ.ºbtener un peleador .

El sistema le envia el mensaje a rastreos de .ºbtener criterioz como este criterio es fuerte este retorna al peleador mas fuerte.