Trabajo Práctico 2 — Java

[7507/9502] Algoritmos y Programación III
Curso 2
Segundo cuatrimestre de 2021

Integrantes:	**Mail:**	**Padrón:**
de Vedia, Agustin Mariano	adevedia@fi.uba.ar	102297
Hosain, Kamal	khosain@fi.uba.ar	104928
Alvarez, Juan Manuel	jualvarez@fi.uba.ar	101589
de Matias Pose, Ignacio	idematias@fi.uba.ar	102201
Milhas, Facundo	fmilhas@fi.uba.ar	102727

Índice

1. Introducción

En este trabajo práctico se desarrolló en Java un programa basado en el juego llamado "¿Donde está Carmen San Diego?", con el fin de hallar una solución al modelo MVC completo que permita ejecutar exitosamente las partidas de juego a través de una interfaz gráfica, utilizando la herramienta JavaFX. Para alcanzar nuestro objetivo se intentó solucionar un conjunto de Casos de Uso especificadas por la cátedra, haciendo uso de la metodología TDD y los conceptos de programación orientada a objetos vistos en el curso.

2. Supuestos

Para desarrollar el modelo se tuvieron que realizar una serie de supuestos, ya que en situaciones se hallaron casos no contemplados por la especificación brindada o decidimos modificar alguna característica.

Cuando arranca el juego, el policía puede comenzar en cualquier ciudad, no necesariamente en la que es robado el objeto. Esto se hizo así para poder utilizar más ciudades en el juego.

En caso de que el usuario se equivoque y viaje a una ciudad errónea, podrá visitar los edificios de dicha ciudad y allí, estos le entregarán las pistas necesarias para viajar a la ciudad correcta. Esta decisión se tomó para hacer más dinámico el juego y no obligar al usuario a volver a la ciudad anterior.

A la hora de elegir ciudades para viajar, el usuario puede encontrarse más de una vez con la misma ciudad, y en caso de estar en niveles avanzados (donde hay que visitar varias ciudades antes de arrestar al ladrón), las ciudades pueden repetirse. De esta manera, logramos cumplir con la dificultad del juego sin limitarnos por la cantidad de ciudades disponibles.

Hay ciudades que están muy cerca entre sí, y especialmente los policías de mayor rango, pueden viajar entre ellas de manera muy veloz. Decidimos fijar como tiempo mínimo de viaje a una hora. La idea es que aunque el lugar de destino sea cercano, tenga un costo temporal considerable.

Al momento de entrar a edificios, el usuario obtendrá pistas sobre el ladrón. Esto es así en todos los edificios pero las pistas pueden repetirse, ya que sino, el policía obtendría muy rápidamente la totalidad de las pistas acerca del ladrón.

El archivo .json de las ciudades fue modificado de forma tal que ahora los distintos atributos llevan consigo parte del cuerpo de la pista. De esta forma se hace más sencillo el procedimiento de mostrar pistas en los edificios.

Una vez que el usuario carga los datos en la computadora se borran de la lista de sospechosos los que no coinciden con estos datos. Es por esto que los datos que cargamos no pueden ser modificados, es decir, que si un sospechoso es borrado de la lista no puede ser reincorporado. La idea es que el usuario vaya cargando los datos a medida que los obtiene y no puede equivocarse al hacerlo.

El policía podrá ser acuchillado durante el transcurso del juego. Decidimos que esto sea una cuestión del azar y puede ocurrir en cualquier ciudad que visita con una probabilidad del 10 %. Creemos que esto le suma un plus al juego que puede hacerlo más entretenido. Le agrega tensión e incertidumbre a la partida.

Por último, el tiempo es contabilizado en todo momento pero se deja al jugador terminar la partida aunque se le haya acabado. Por supuesto, en ese caso, no se lo declara ganador.

3. Diagramas de clase

El diagrama de clases principal se puede observar en la figura 1. En el se muestran los principales objetos que entran en juego en el programa: Caso, Policia, Ladron, Tesoro, Computadora y Mapa. Todos éstos se relacionan con Caso en relación 1:1. Se pueden observar en el diagrama cómo interactúa Caso con las demás clases según sus métodos. Las demás clases que se irán explicando del programa no tienen un vínculo directo con Caso.

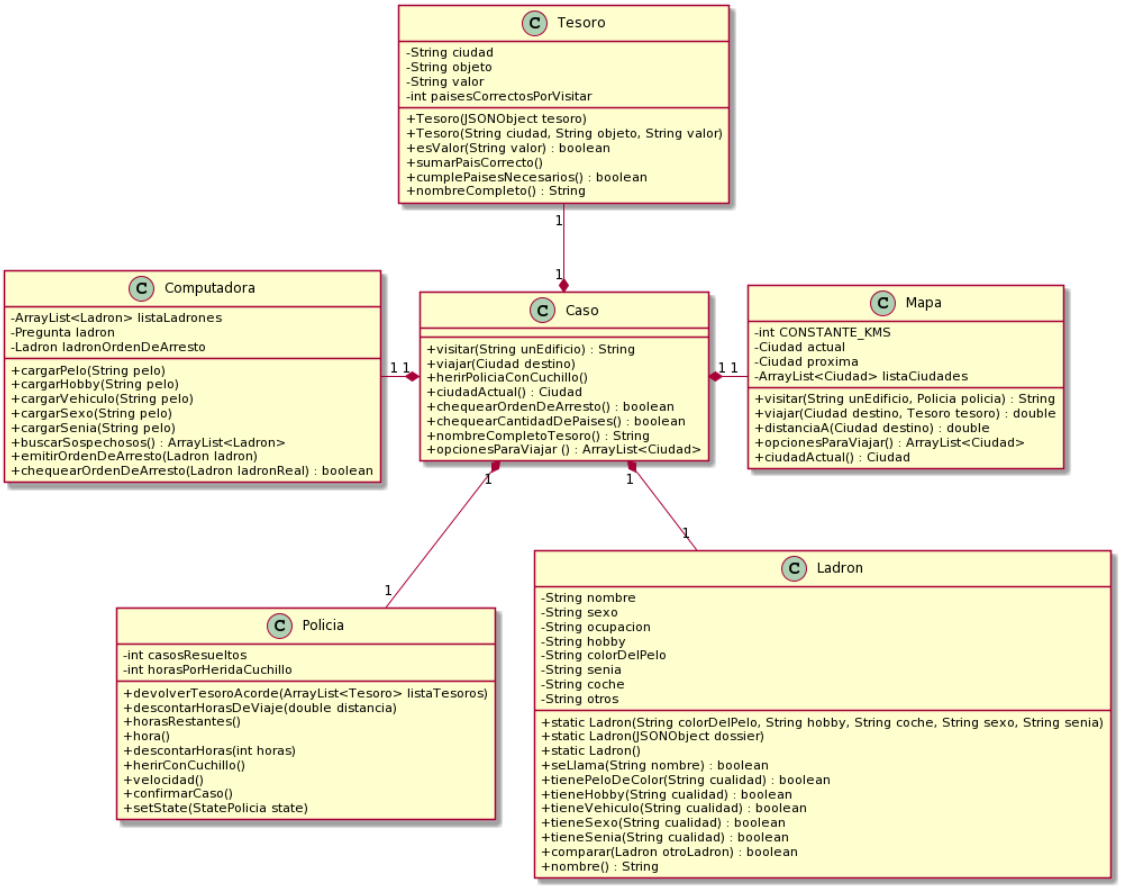

Figura 1: Diagrama de clases principal.

La clase Policia tiene implementado un patrón State, por el cual se maneja su estado de Novato, Detective, Investigador y Sargento. Éstos encapsulan distintos comportamientos y el policía varía su estado entre ellos a medida que vaya confirmando casos. StatePolicia es una clase abstracta, se utiliza herencia en los estados ya que comparten los atributos y la mayoría de sus métodos. Se puede ver el diagrama de clases correspondiente en la figura 2. La clase Policia tiene como atributo su Reloj, donde se maneja la resta de tiempo.

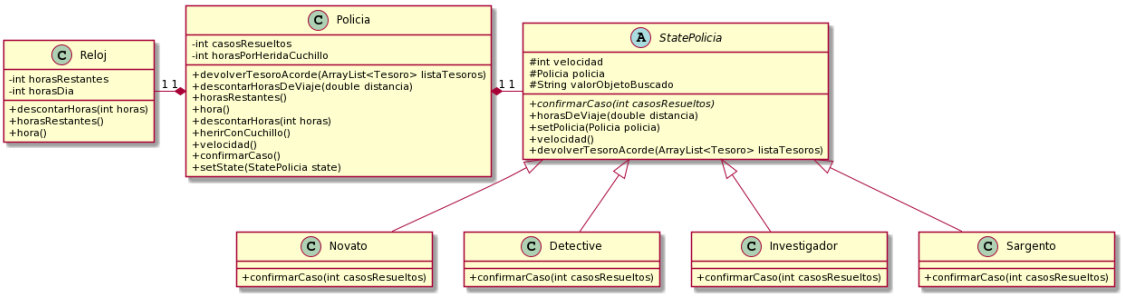

Figura 2: Diagrama de clase: Policía.

Para la clase Ladron se utilizó el patrón Decorator. Tanto la clase Ladron como la clase abs-

tracta Decorator implementan Pregunta y luego de Decorator se extienden las clases ColorDePelo, Hobby, Nombre, Senia, Sexo y Vehiculo, correspondientes a las características de los ladrones. De esta manera se manejan eficientemente las preguntas sobre los ladrones y sus comparaciones. Estas relaciones se muestran en la figura 3.

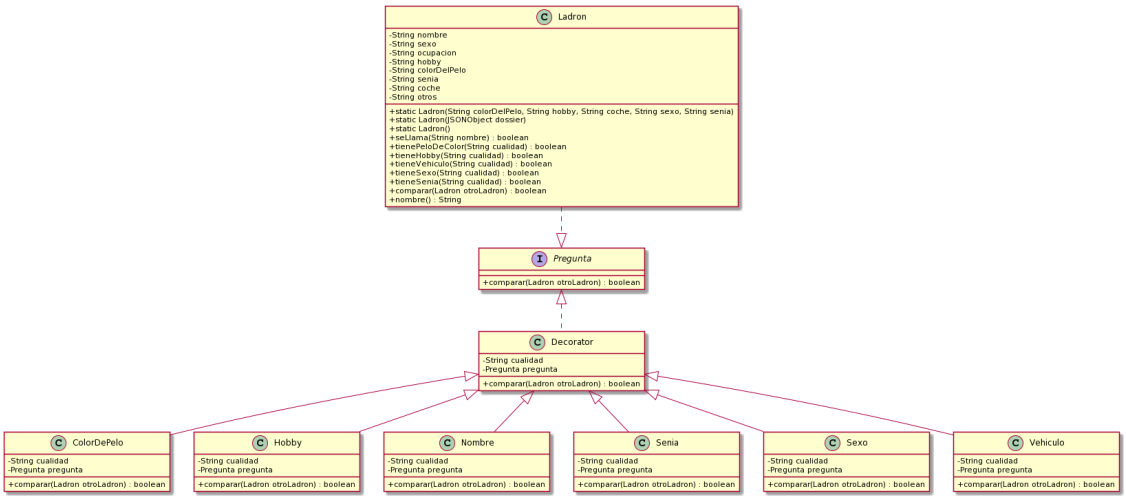

Figura 3: Diagrama de clases: Ladrón y Decorator.

La clase Mapa se relaciona con la clase Ciudad, tiene como atributos dos ciudades: actual y próxima, y la lista completa de ciudades del juego. El diagrama de clases correspondiente se puede observar en la figura 4. De Mapa se envía el mensaje de visita para Ciudad, la cual luego identifica el Edificio a visitar y de ahí continúa.

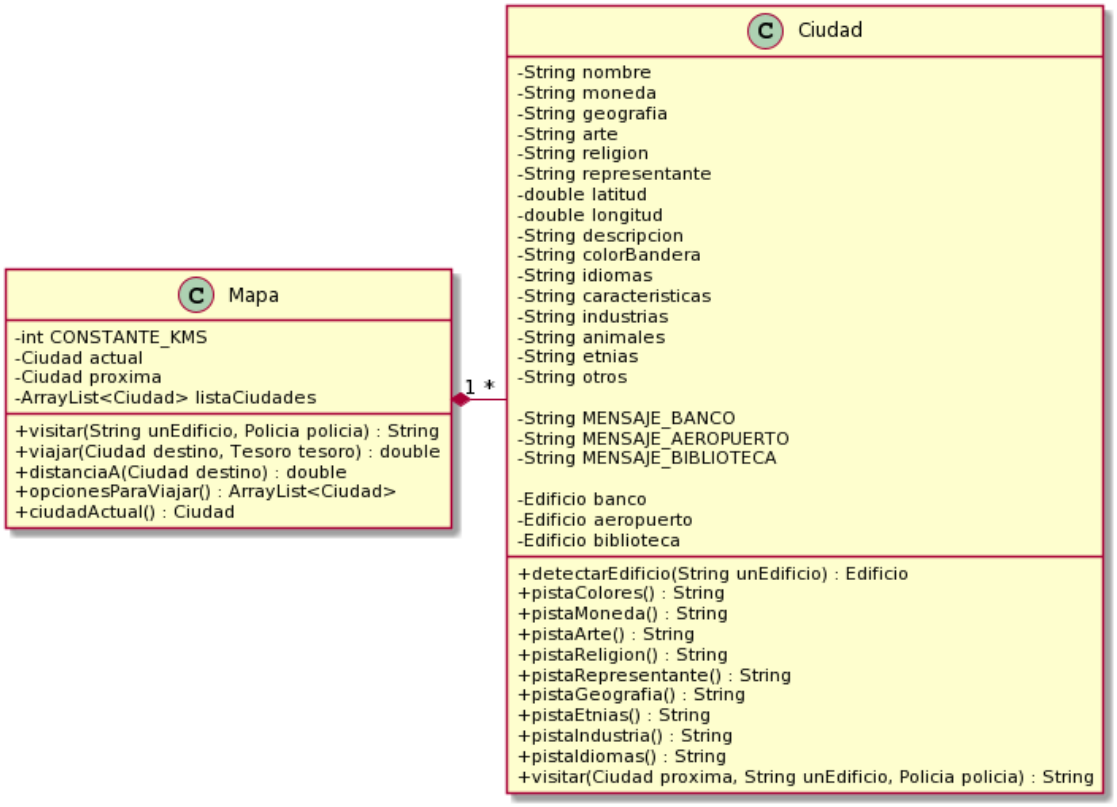

Figura 4: Diagrama de clase: Mapa y Ciudad.

Los distintos edificios Aeropuerto, Banco y Biblioteca, implementan la interfaz Edificio. El diagrama de clase se puede ver en la figura 5. Cada uno difiere en la implementación de darPista, ya que piden pistas distintas. Aeropuerto maneja pistas de viaje, Banco económicas y Biblioteca pistas de cultura.

Figura 5: Diagrama de clases: Edificio.

Finalmente, se muestra en la figura 6 el diagrama de clase referente a las pistas. Pista es una interfaz que se implementa en PistaFacil, PistaMedia y PistaDificil. La manera n que se diferencian las pistas recae en que cada clase, dentro de sus métodos, le piden a Ciudad distintos datos. Por ejemplo, en el caso de pistas económicas, las fáciles comentan sobre la moneda del país, las medias

sobre su industria y finalmente las difíciles sobre su representante.

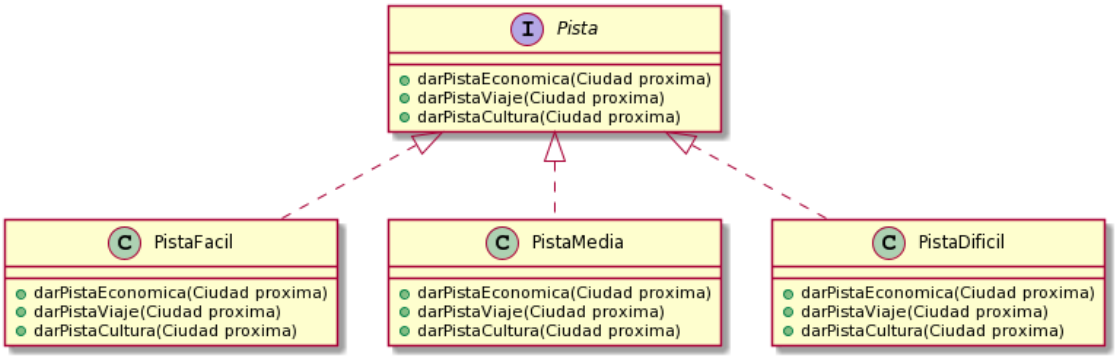

Figura 6: Diagrama de clases de Pista.

4. Detalles de implementación

4.1. Decorator

Se decidió usar el patrón de diseño Decorator para poder realizar la búsqueda de ladrones. De esta manera, sumamos la funcionalidad al cargar las cualidades del ladrón por capas, ya que se permite hacer búsquedas de ladrones de los cuales no tenemos todas las características.

4.2. State

Utilizamos el patron de diseño State porque en todo el juego hay una única instancia de Policía, la cual tiene que ir cambiando de comportamiento en tiempo de ejecución a medida que se avanza en el juego. Por eso creímos ideal el uso de este patrón.

4.3. Herencia

Utilizamos Herencia para poder hacer uso de los patrones anteriormente mencionados.

4.4. Interface

Utilizamos Interfaces en dos situaciones, en Pista y Edificio. En ambos casos, la interface era la mejor opción ya que las clases comparten funcionalidades pero no era necesario que compartan atributos.

4.5. Constructores de clase

Se hicieron varias sobrecargas de los constructores para poder realizar tests unitarios. Esto se debió a que originalmente varias clases se construyen leyendo de archivos .json.

5. Diagramas de secuencia

En el primer diagrama de secuencia, se muestra el caso en que se visita un banco y obtiene una pista económica de la ciudad próxima. Se observa que al llegar a la ciudad actual, ésta detecta el edificio que se quiere acceder y le manda a él que devuelva una pista. Se le pasa por argumento la ciudad próxima, de la cual precisa la información, y el valor de la pista que debe dar. El valor de la pista depende del rango del policía, esto se maneja dentro de su State. Luego la pista le pide la pista económica a la ciudad, la cual le responde con el String completo de la pista, y el banco le termina descontando el tiempo correspondiente al policía.

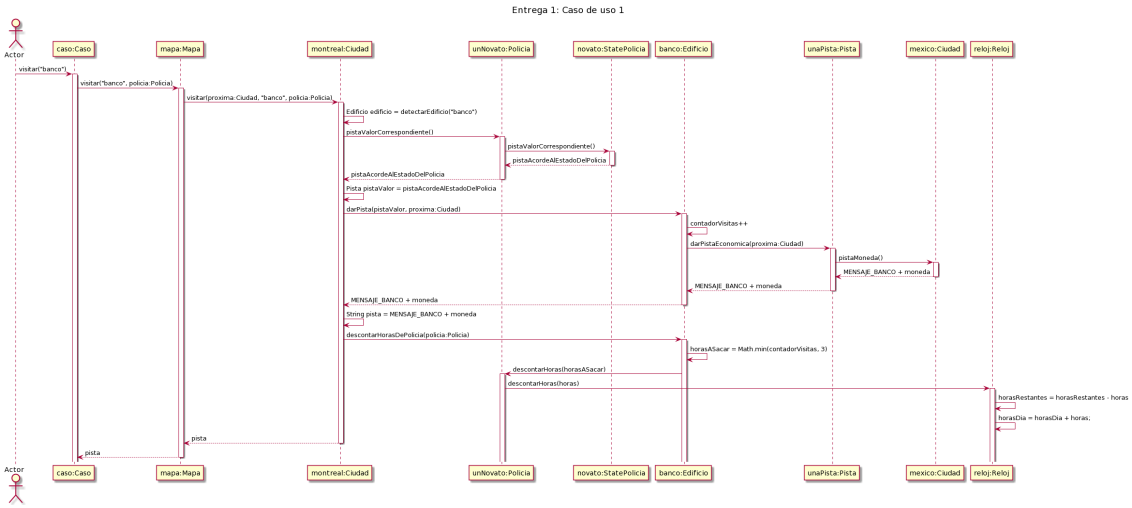

Figura 7: Diagrama de secuencia: visitar un banco.

La diferencia entre pistas fáciles, medias y difíciles radica en que se muestran distintos datos de la ciudad. En el caso de pistas económicas, las fáciles comentan sobre la moneda del país, las medias sobre su industria y finalmente las difíciles sobre su representante. El policía novato maneja pistas fáciles, los detectives e investigadores las pistas medias y finalmente el sargento obtiene pistas difíciles.

Al momento de descontar las horas correspondientes, la clase Reloj hace el chequeo de si es la hora de dormir según la hora del día que sea, y de ser necesario se suman las 8h respectivas y se ajustan las horas del día.

```
descontarHoras(horas) {
    horasDia = horasDia + horas;
    if(horasDia >= 23) {
        horasRestantes = horasRestantes - 8; // El policia duerme 8h
        horasDia += 8 - 24; // Se despierta 8h despues de irse a dormir
    }
}
```

En la figura 8 se puede ver el diagrama de secuencia correspondiente al caso donde se viaja a un país. Se viaja al país `México` y se descuentan las horas correspondientes al policía. En la primera parte, Mapa maneja el cálculo de la distancia recorrida en el viaje y le indica a Tesoro que reste el contador de `viajesPorHacer`. Mapa ajusta también su ciudad actual, a México en este caso. Si la ciudad a la cual se viaja era efectivamente la correcta ciudad próxima, se actualiza la ciudad próxima a una random en Mapa. Sin embargo, si se viaja a una ciudad errónea, se mantiene la ciudad próxima que estaba. Las pistas que luego se den serán correspondientes a esa ciudad, según lo explayado en los supuestos (sec. 2).

Al final, se pasan a policía los Kms recorridos y éste delega a su State el pasaje a tiempo, ya que depende de su velocidad, y a Reloj el descuento final del tiempo.

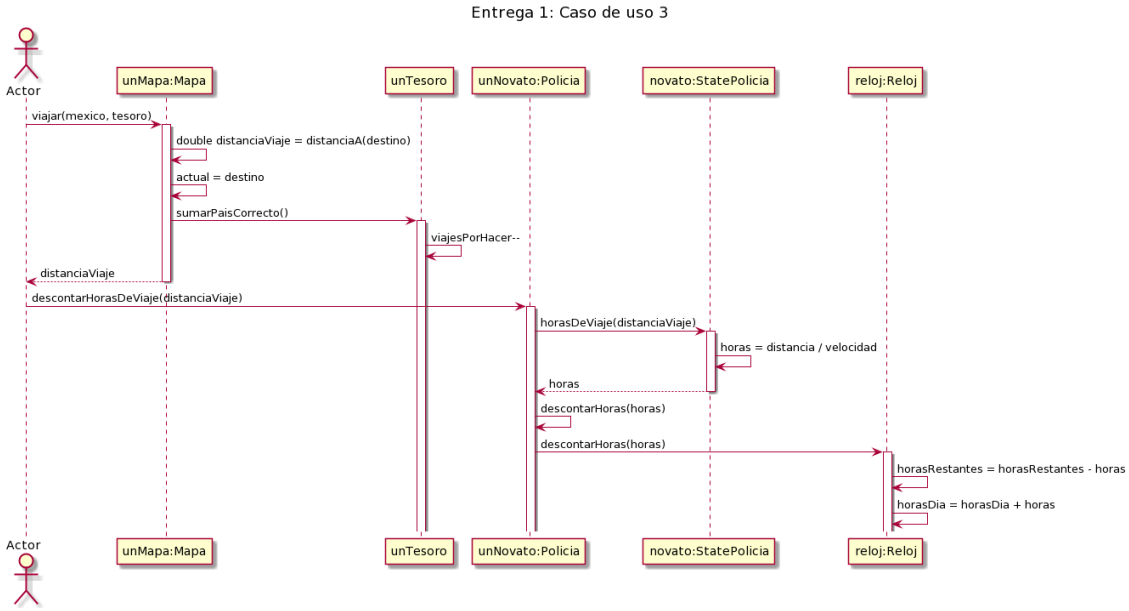

Figura 8: Diagrama de secuencia: viajar a México.

Finalmente, se muestra el diagrama de secuencia de la búsqueda de sospechosos en la computadora en la figura 9.

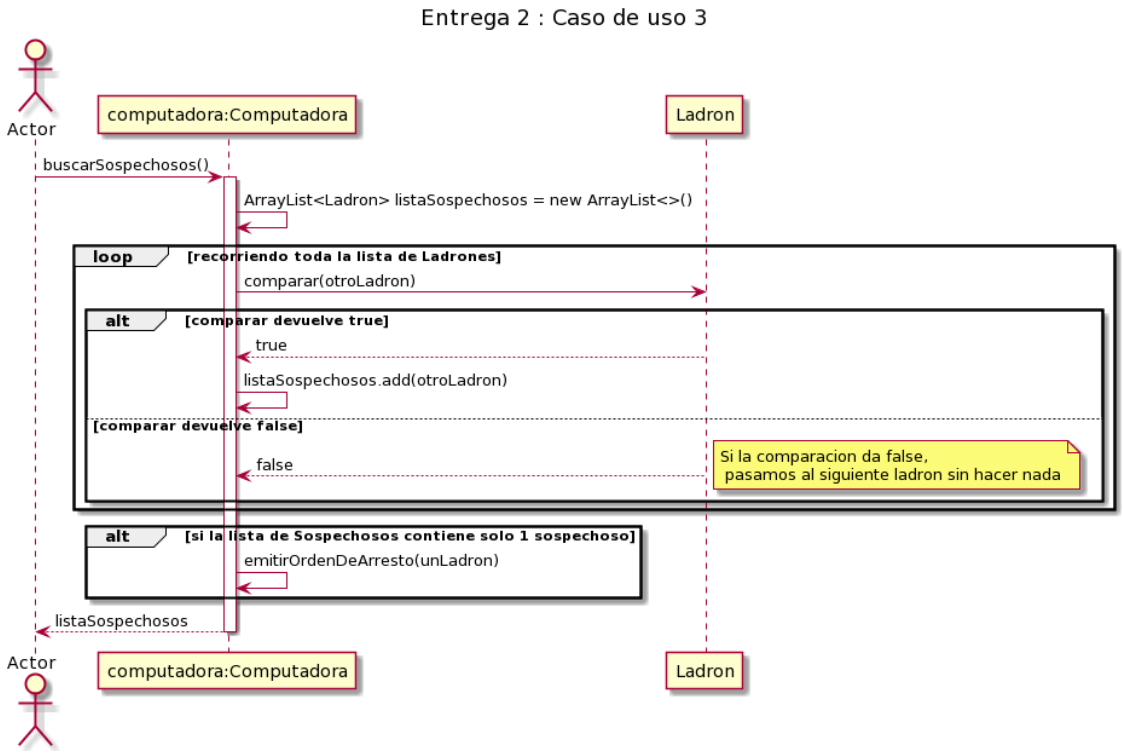

Figura 9: Diagrama de secuencia: buscar sospechosos.

La computadora ya tiene cargadas las características ingresadas por el policía y tiene un ladrón (`otroLadron`) con dichas características. Lo que realiza entonces es un filtro entre la lista de ladrones que tiene cargada. Se recorren los ladrones y se les manda un mensaje indicándoles que se comparen con `otroLadron`. Se obtiene entonces una lista de sospechosos con los posibles ladrones, según las características previamente guardadas.

Si la lista obtenida tiene un único sospechoso, se genera automáticamente la orden de arresto para dicho individuo.