FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico — Java
Algo-thief

[7507/9502] Algoritmos y Programación III
Curso 1
Segundo cuatrimestre del 2021

Alumno:	CODINA, CAMILA
Número de padrón:	106339
Email:	ccodina@fi.uba.ar
Alumno:	GRÜNER, TOMÁS
Número de padrón:	105798
Email:	tgruner@fi.uba.ar
Alumno:	LOURENGO CARIDADE, LUCÍA
Número de padrón:	104880
Email:	llourengo@fi.uba.ar
Alumno:	TOULOUSE, ALAN
Número de padrón:	105343
Email:	atoulouse@fi.uba.ar
Alumno:	VACCARELLI, SANTIAGO
Número de padrón:	106051
Email:	svaccarelli@fi.uba.ar

Índice

1. Introducción

El presente informe reune la documentación de la solución del segundo trabajo práctico de la materia Algoritmos y Programación III que consiste desarrollar una aplicación de manera grupal aplicando todos los conceptos vistos en el curso, utilizando un lenguaje de tipado estático (Java) con un diseño del modelo orientado a objetos y trabajando con las técnicas de TDD e Integración Continua.

2. Supuestos

3. Modelo de dominio

4. Diagramas de clase

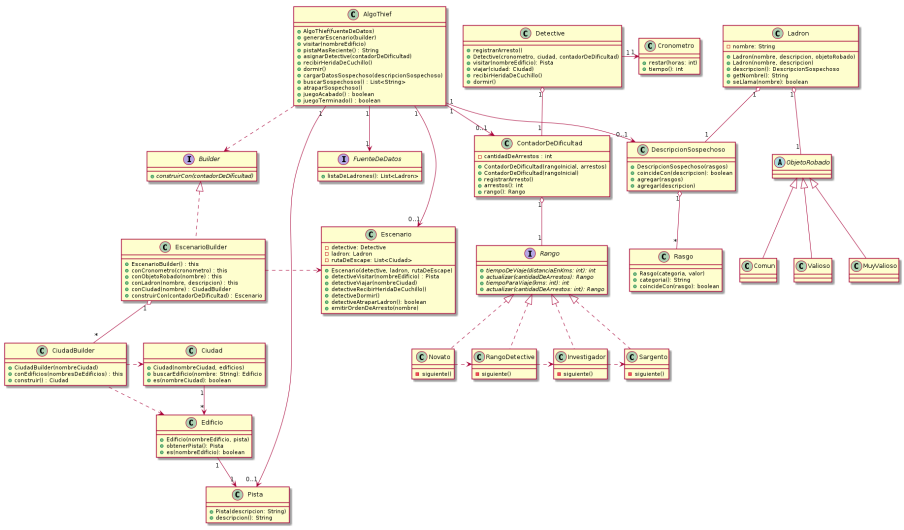

Figura 1: Diagrama de clase general.

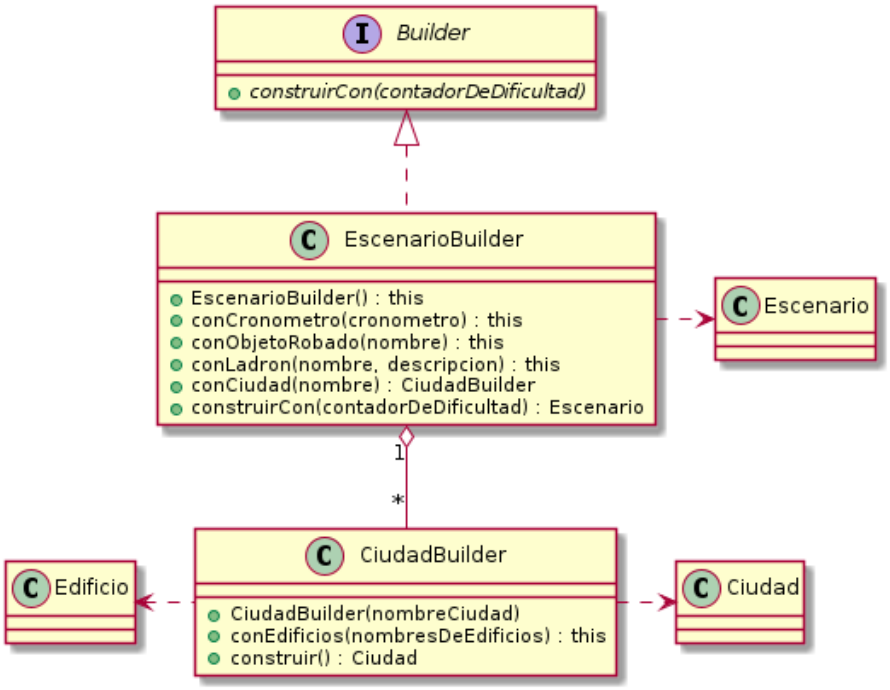

Figura 2: Diagrama de clase del builder.

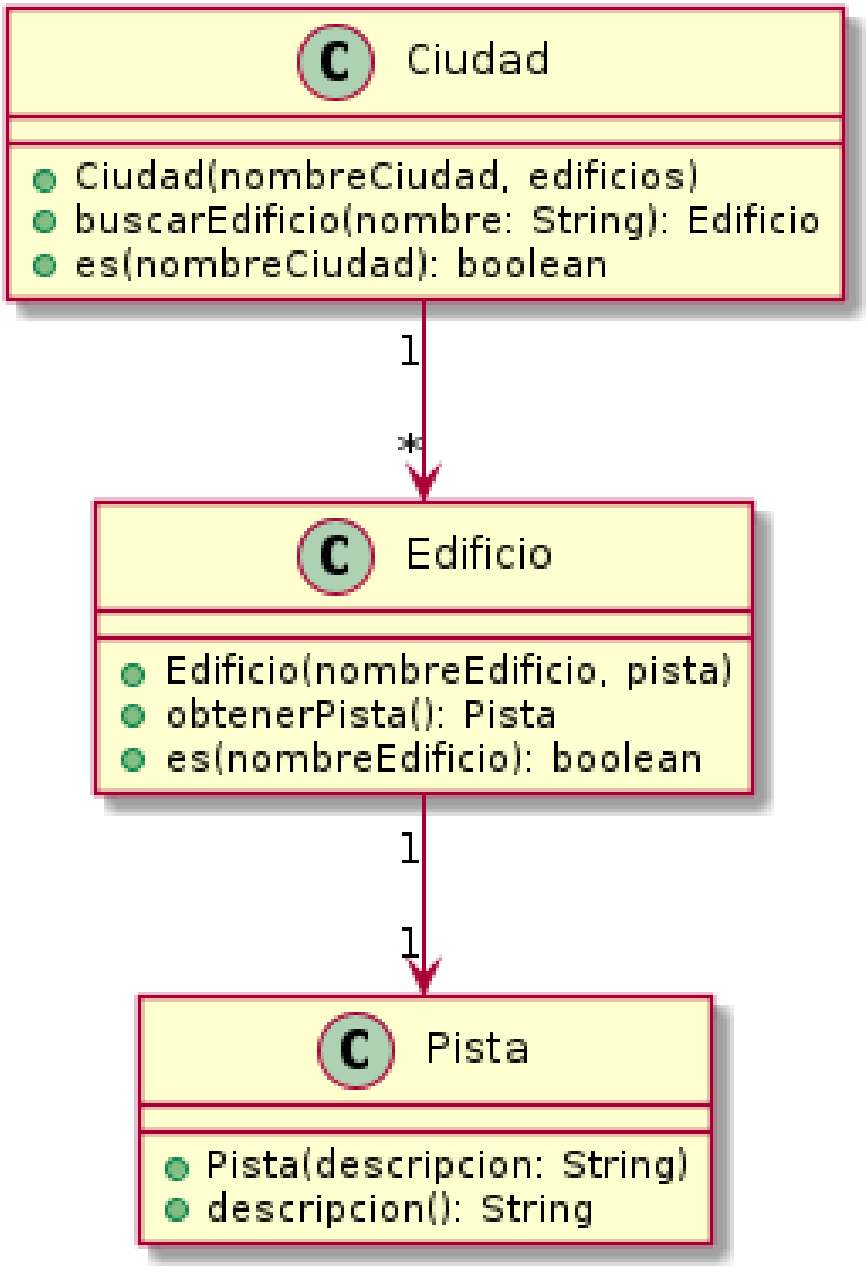

Figura 3: Diagrama de clase ciudad.

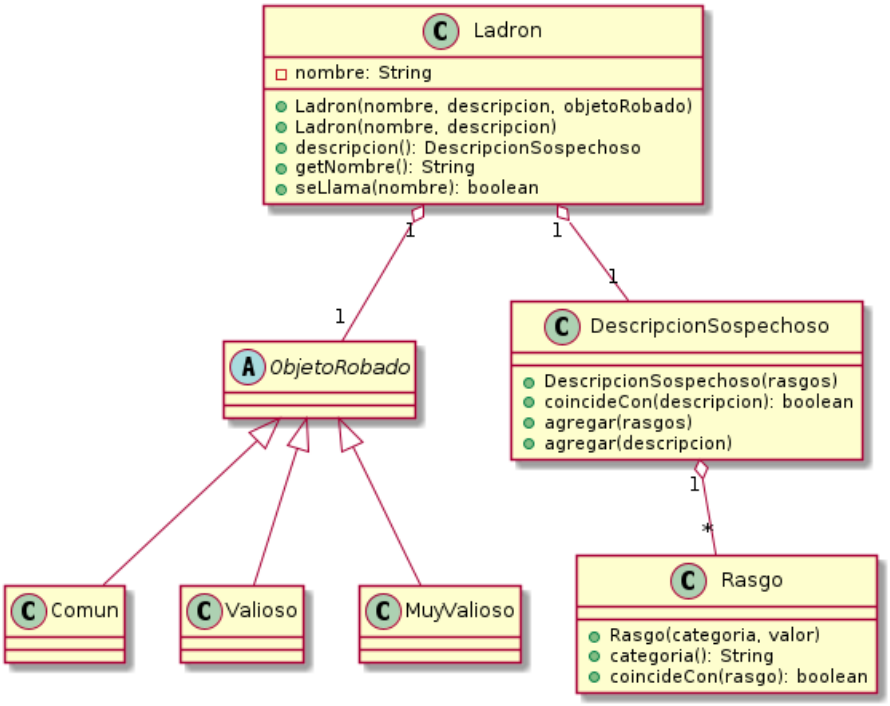

Figura 4: Diagrama de clase de Ladrón.

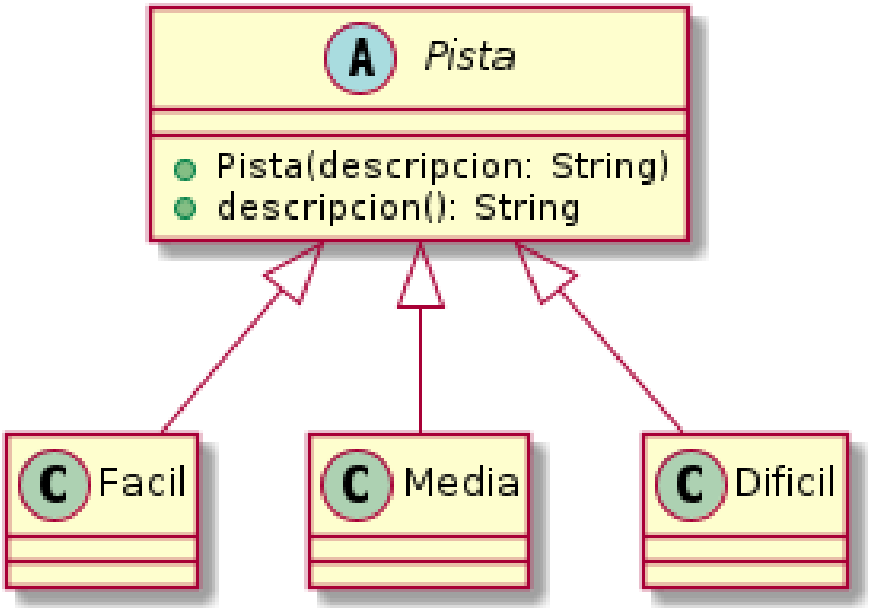

Figura 5: Diagrama de clase de Pista.

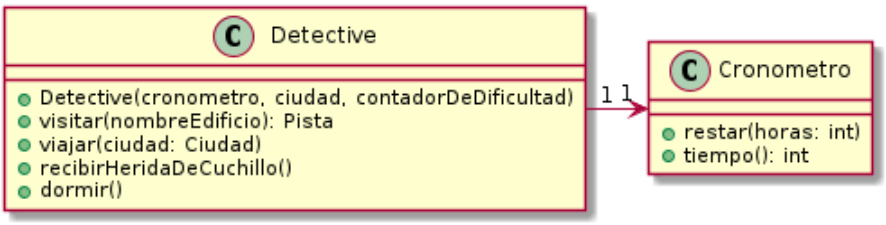

Figura 6: Diagrama de clase de Policia.

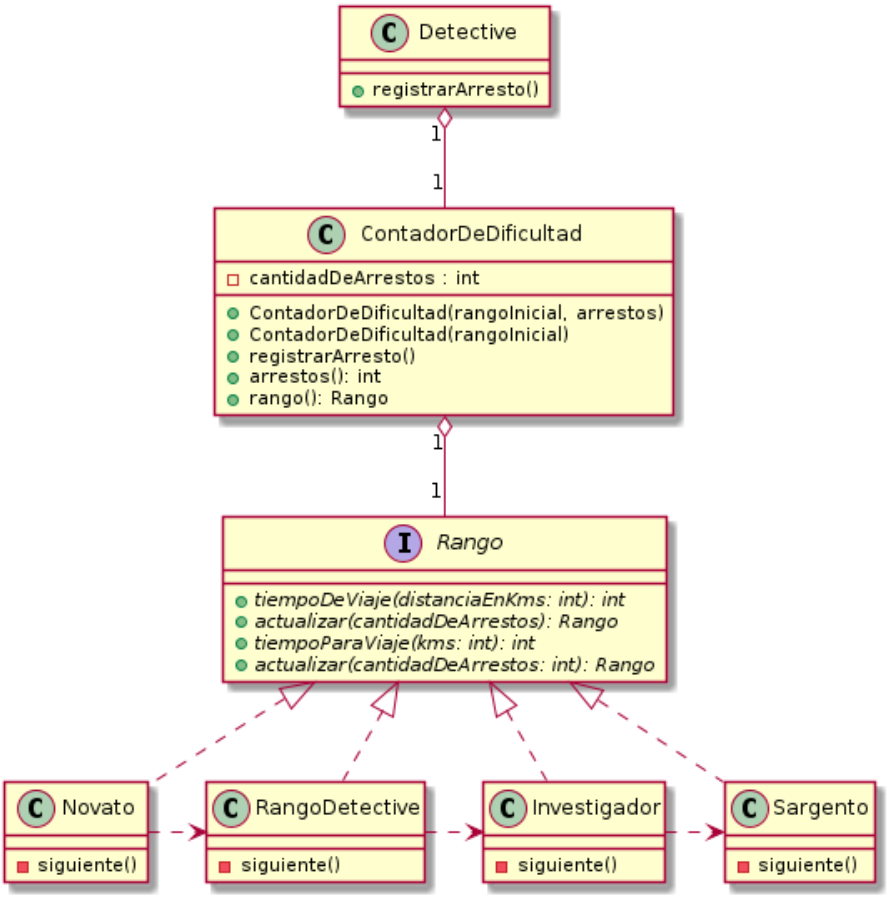

Figura 7: Diagrama de clase Rango.

5. Detalles de implementación

Con el fin de beneficiar el uso de las mejores prácticas durante la creación de nuestro proyecto, tuvimos en cuenta la posibilidad de implementar patrones de diseño para determinados casos que logramos identificar.

Utilizamos el patrón 'Builder', un patrón de diseño creacional para construir el escenario inicial de nuestro juego. Éste nos permite producir diferentes instancias de un objeto a partir del mismo constructor.

Previo a la implementación del mismo, notamos que debíamos escribir bastantes mensajes de inicialización pero logramos extraer el código de inicialización del objeto a objetos separados, los

builders.

También recurrimos al patrón 'Chain of Responsibility', un patrón de comportamiento que nos permite hacer una cadena de solicitudes: al recibir una solicitud, un manejador decide si debe procesarla o redirigirla al siguiente manejador de la cadena. Nuestro objetivo con este patrón es llevar un seguimiento sobre el nivel de dificultad de nuestro juego.

6. Excepciones

7. Diagramas de secuencia

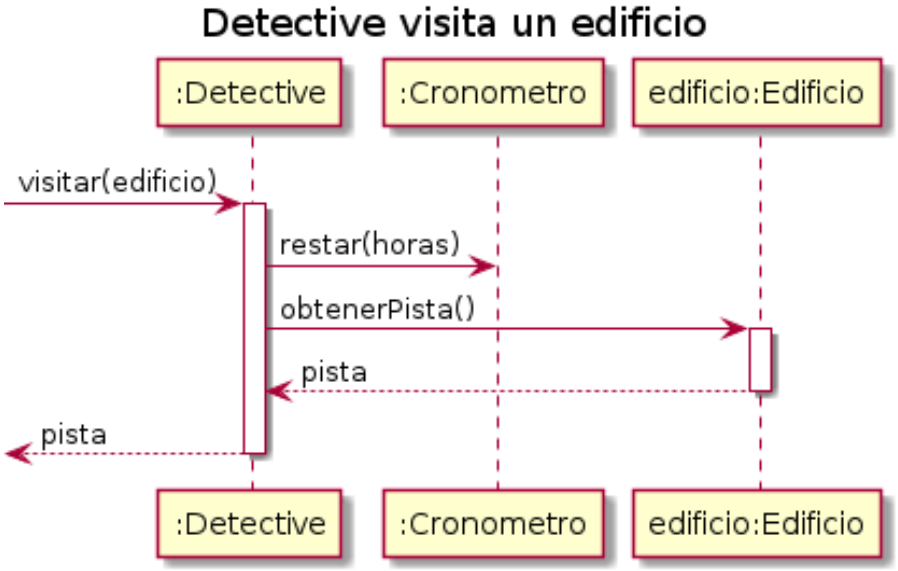

Figura 8: Diagrama de secuencia Entrega Cero.

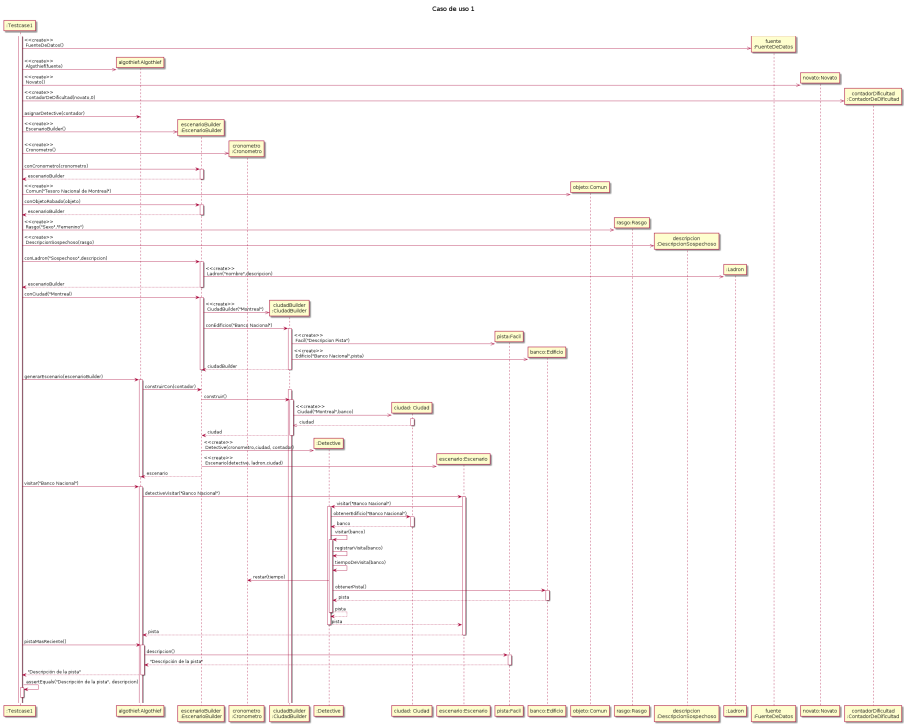

Figura 9: Diagrama de secuencia Entrega 1 TestCase01.

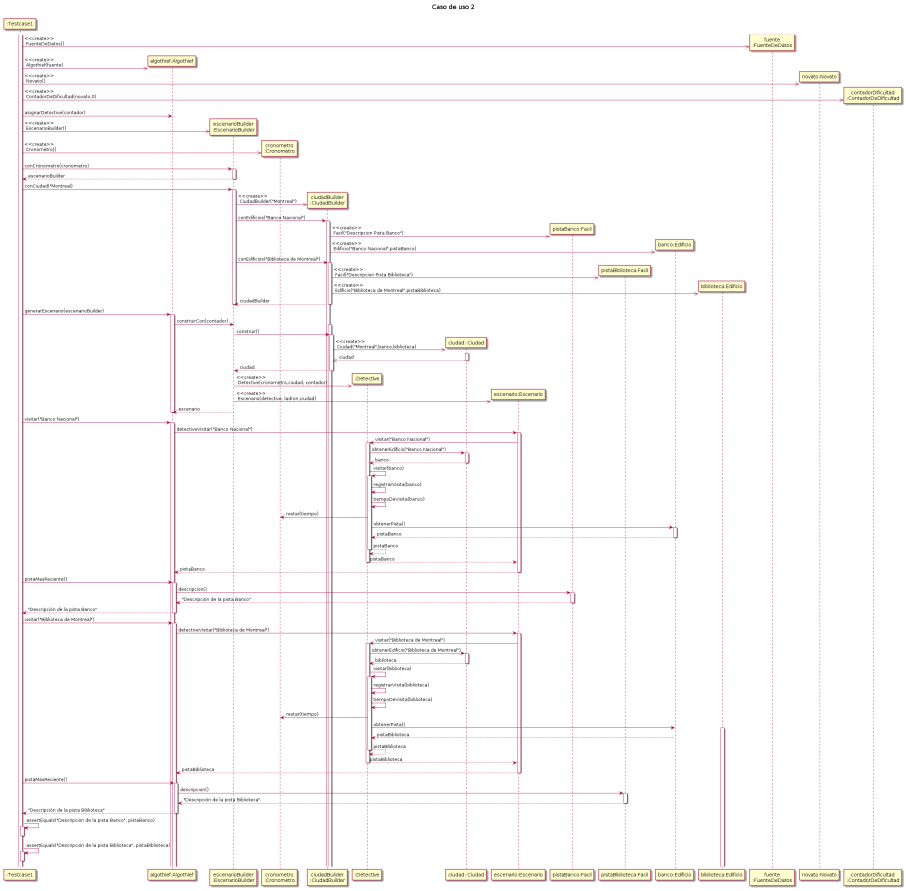

Figura 10: Diagrama de secuencia Entrega 1 TestCase02.

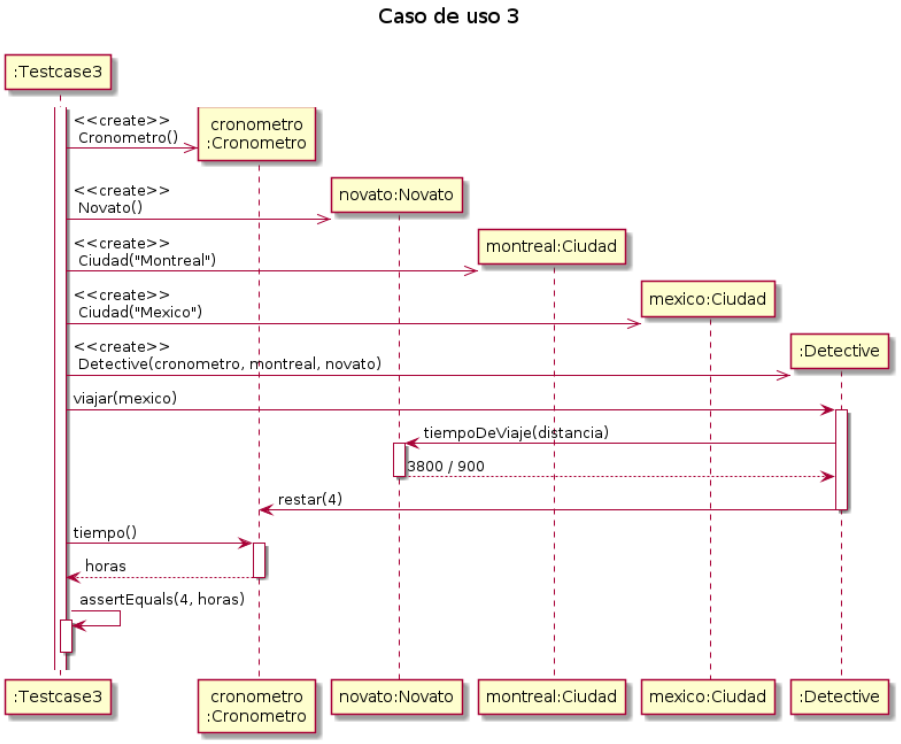

Figura 11: Diagrama de secuencia Entrega 1 TestCase03.

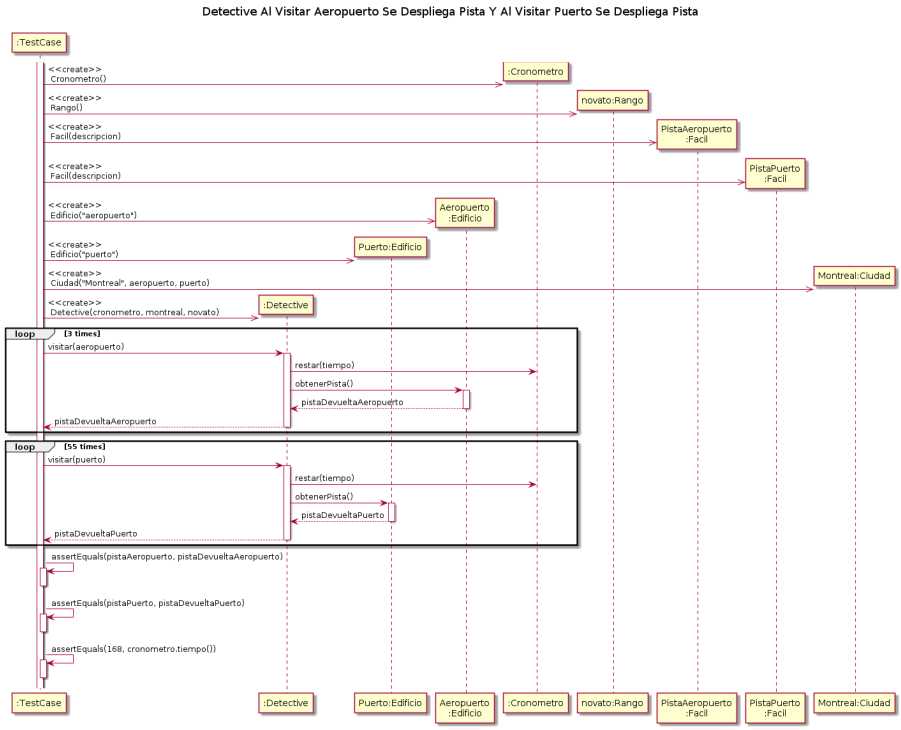

Figura 12: Diagrama de secuencia Entrega 1 TestCase04.

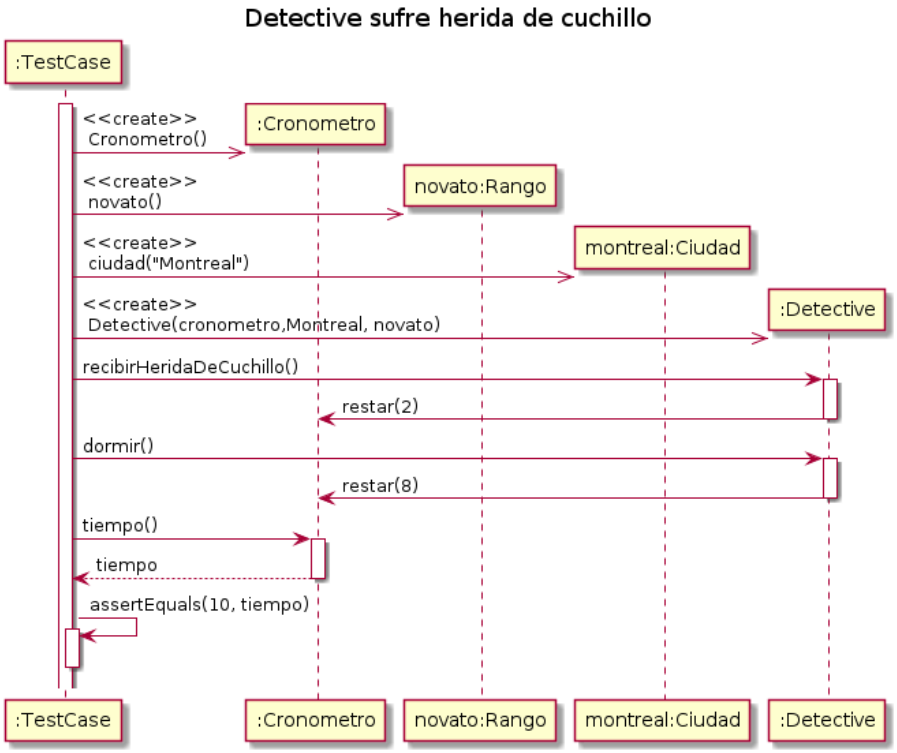

Figura 13: Diagrama de secuencia Entrega 1 TestCase05.

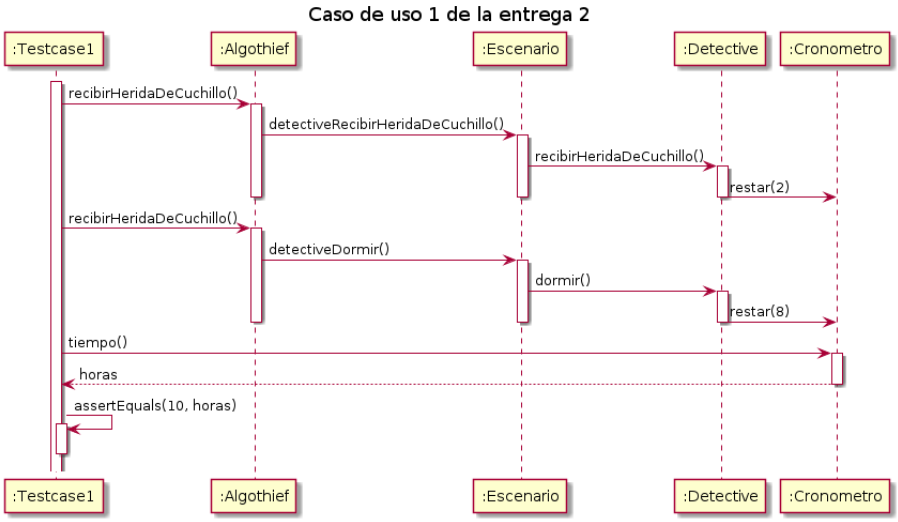

Figura 14: Diagrama de secuencia Entrega 2 TestCase01.

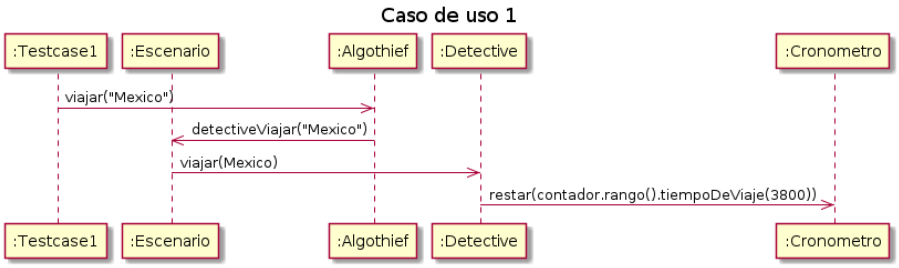

Figura 15: Diagrama de secuencia Entrega 2 TestCase02.

.png

Figura 16: Diagrama de secuencia Entrega 2 TestCase03.

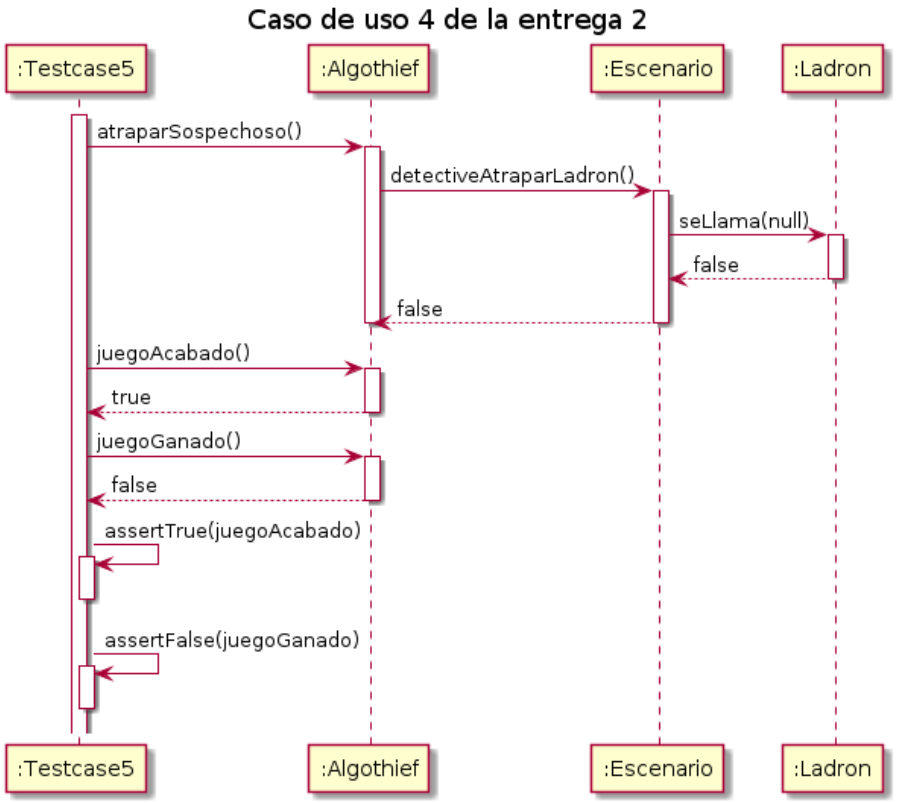

Figura 17: Diagrama de secuencia Entrega 2 TestCase04.

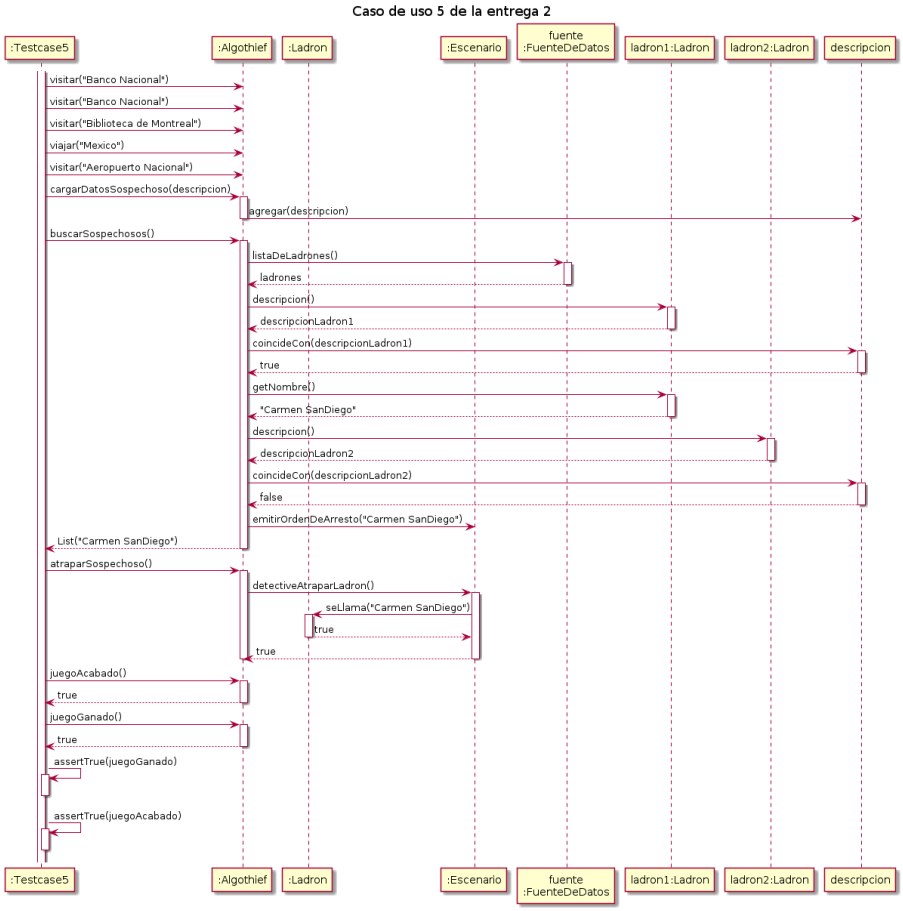

Figura 18: Diagrama de secuencia Entrega 2 TestCase05.