Trabajo Práctico 2 — Java

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

Alumno:	Retorri, Julian Rafael - 106581
Alumno:	Saez, Edgardo Francisco - 104896
Alumno:	Pensado, Ivan Manuel - 106804
Alumno:	Sabaj, Gaston - 106147

Índice

1. Introducción

El presente informe reune la documentación de la solución del 2do trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar el juego Çarmen Sandiego"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

Los supuestos que adopte a lo largo del Trabajo Práctico fueron:

1 - La cantidad de destinos al que se puede viajar desde una ciudad son 3.

2 - El policia está habilitado para dormir entre las 22:00hs y las 06:00hs del día siguiente, y la demora de tiempo al dormir son de 8 horas.

3 - Al delincuente se lo podrá intentar arrestar a la tercera entrada de un edificio cualquiera de la última ciudad del delincuente. Si al momento de intentar arrestarlo, no se tiene una orden de arresto ya computada, se va a perder la partida.

4 - Si al momento de atrapar el delincuente no se posee una orden de arresto que coincida con el nombre del mismo , entonces se perderá la partida.

5 -Los ataques del delincuente se realizan en los edificios de la última ciudad recorrida por el delincuente.

6 -El delincuente está ubicado en la última ciudad de su recorrido, a la espera de que el policia llegue.

3. Modelo de dominio

El trabajo práctico entregado trata de una implementación de una versión del juego *Carmen Sandiego*, el cuál trata de viajar a ciudades del mundo buscando a un delincuente que cometió un robo de un objeto importante de un país, y arrestarlo. Mientras se desarrolla el juego, el usuario puede: -Ver los viajes posibles desde la ciudad actual, hacia otras ciudades.

-Puede viajar hacia esos destinos posibles

-Puede entrar a cualquiera de los 3 edificios de su ciudad actual

-Puede entrar a la computadora de Interpol para poder cargar las pistas recopiladas acerca de las características del delincuente, como su color de pelo, su hobbie principal, su sexo, etc.

En el caso de que se tenga una orden de arresto para la hora de arrestar al delincuente, se gana la partida, de lo contrario, si se tiene una orden de arresto de un sospechoso incorrecto, o no se tiene ninguna orden de arresto, se pierde la partida.

4. Diagramas de clase

A continuación se muestran los diagramas de clase del modelo, el cuál muestran la relacion entre la clases que nos ayudaron a resolver el problema.

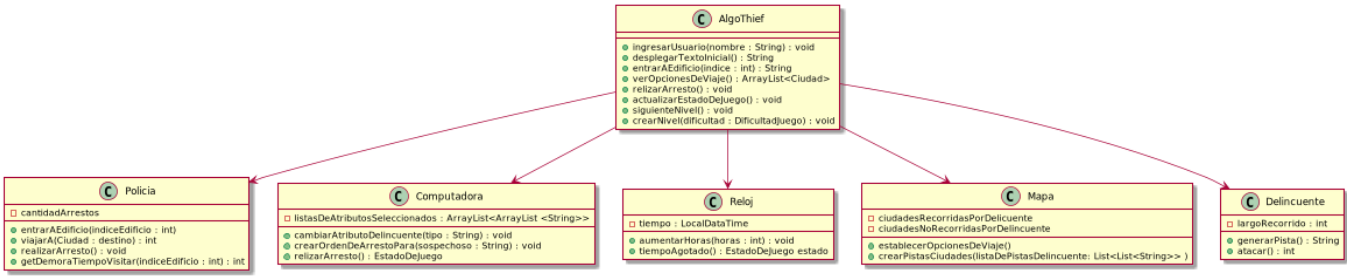


Figura 1: Diagrama de clase 1 de AlgoThief

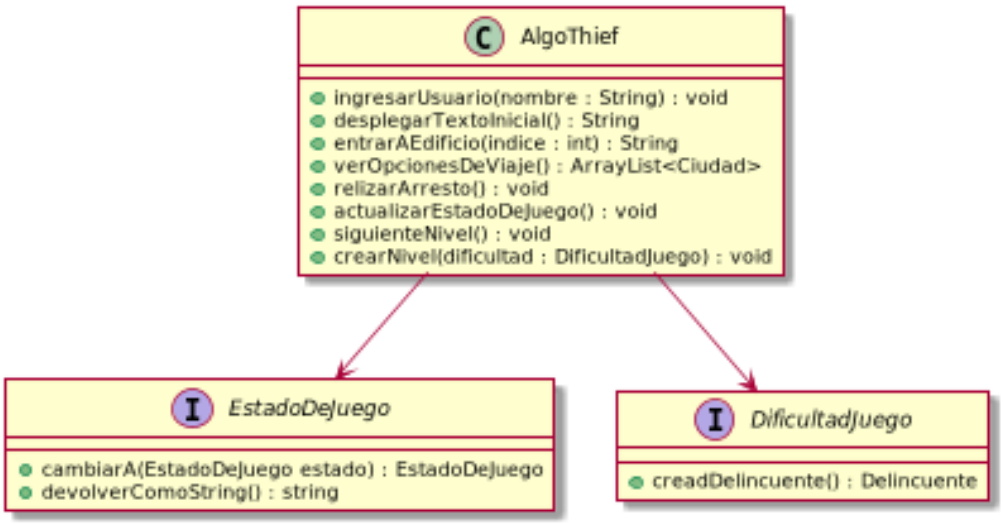


Figura 2: Diagrama de clase 2 de AlgoThief

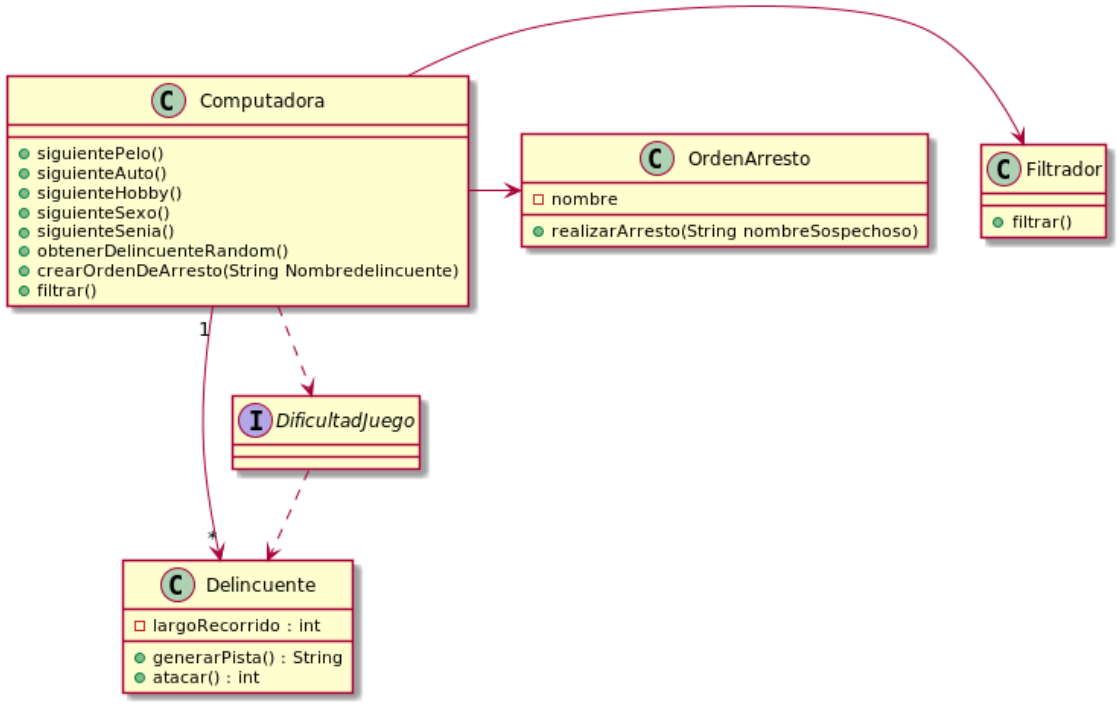


Figura 3: Diagrama de clase de Computadora

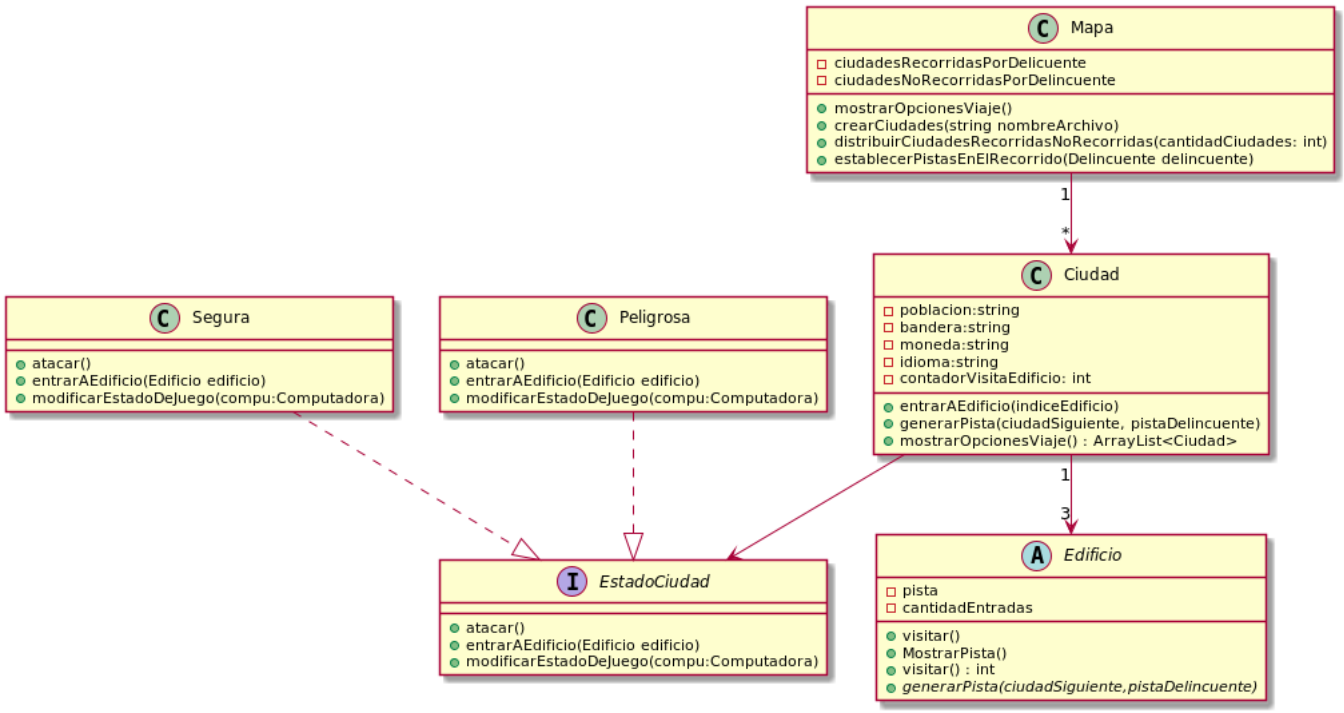
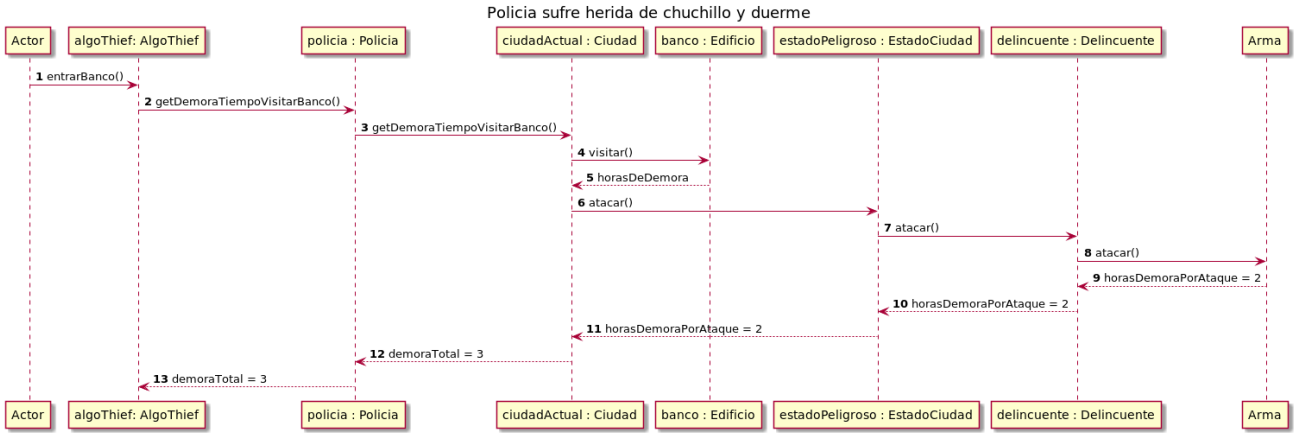
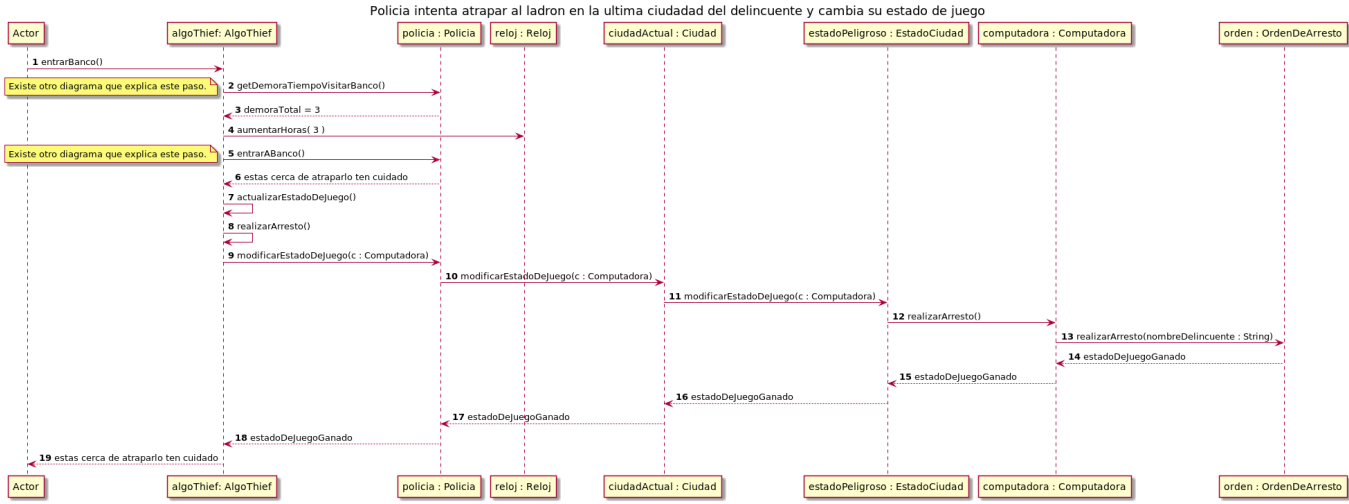


Figura 4: Diagrama de clase de Mapa

5. Diagramas de secuencia





6. Diagramas de paquetes

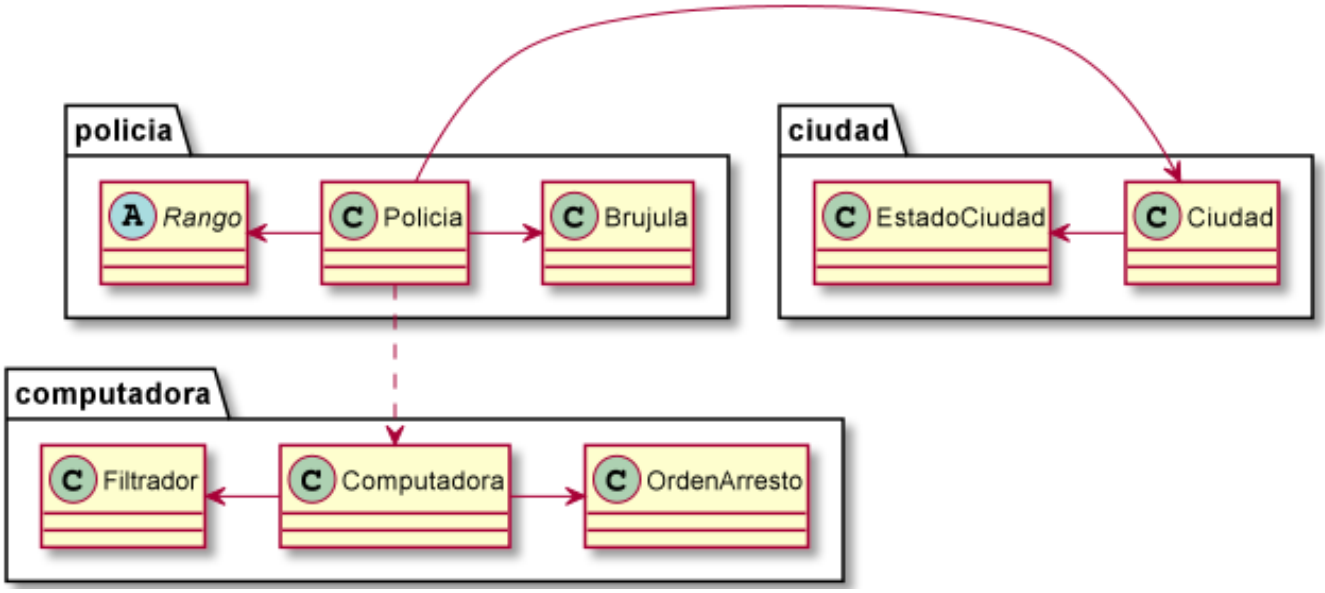


Figura 5: Diagrama de Paquetes 1

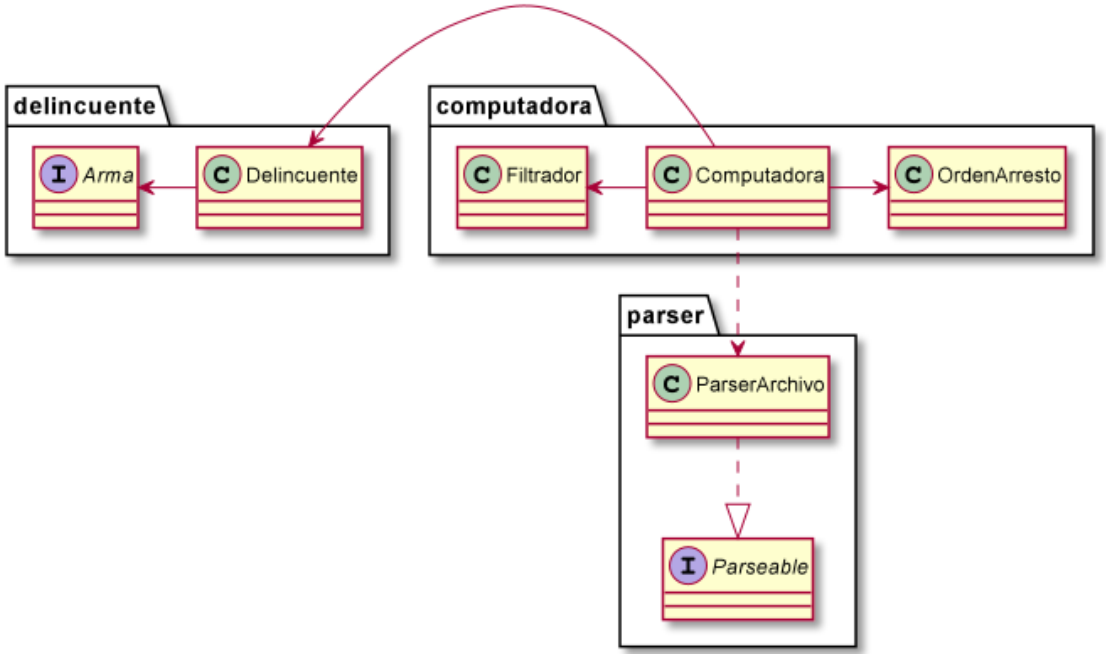


Figura 6: Diagrama de Paquetes 2

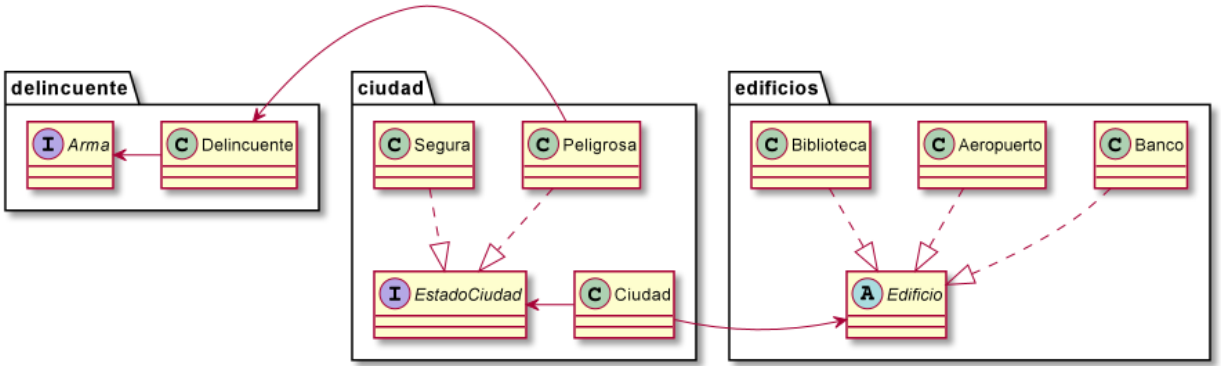


Figura 7: Diagrama de Paquetes 3

7. Detalles de implementación

Utilizamos el patron Model–view–controller para separar los datos y la lógica del juego de su representación y el módulo encargado de gestionar los eventos y las comunicaciones. Tambien utilizamos el patron state para diferenciar los estados de algoThief, tanto en su dificultad como en su estado de nivel (ganado, perdido, jugando) con lo que un mismo metodo desencadenara diferentes acciones dependiendo de su estado ahorrandonos el uso de muchos condicionales, aplicando de esta manera el concepto de polimorfismo.

La delegacion de tareas se da reiteradas veces en el modelo, por ejemplo , el delincuente le delega a su arma el daño (entendido como horas) de su ataque, o el policia le delega a su clase Brujula el tiempo del viaje.De esta manera, evitamos que las clases se sobrecarguen de responsabilidades. Así como utilizamos delegación, también utilizamos herencia con la clase abstracta edificio, ya que los tipos de edificios heredan métodos a implementar, y un método de código común para todos, el cual se encarga de incrementar el contador de visitas al respectivo edificio.

Respecto a principios de diseño, utilizamos el principio de inversión de dependencias, por ejemplo para que las clases que implementan las interfaz *DificultadJuego*, no dependan de métodos que no implementan. También aplicamos el principio de abierto/cerrado, para que el código de las clases pueda ser extensible sin modificar código ya creado, por ejemplo si el día de mañana queremos agregar nuevos edificios; nuevos estados del juego o nuevas dificultades del juego.

8. Excepciones

Creamos las excepciones CiudadNoExistente y DatoNoExistente, esto debido al riesgo que conlleva que los archivos de texto utilizados tengan un formato erroneo o introduzcan texto equivocado.