**FIUBA - 7507**

**Algoritmos y programación 3**

*Trabajo práctico 2: Algo-thief*

1er cuatrimestre, 2014

(trabajo grupal)

Alumnos:

|  |  |  |
| --- | --- | --- |
| Nombre | Padrón | Mail |
| Coria, Flavia | 95760 | [flaviaacoria@gmail.com](mailto:flaviaacoria@gmail.com) |
| Mussi, Tomas | 94485 | [tomasmussi@gmail.com](mailto:tomasmussi@gmail.com) |
| Perisset, Elisa | 95404 | [eliperisset@gmail.com](mailto:eliperisset@gmail.com) |
| Poncet, Nicolas | 94983 | [poncetnicolas@hotmail.com](mailto:poncetnicolas@hotmail.com) |

Fecha de entrega final:

Tutor: Pablo Matías Rodriguez Massuh

Nota Final:

## *Supuestos*

Cuando se ingresa a los edificios de la última ciudad a recorrer, donde encuentra al ladrón, por cada ingreso fallido, el policía será acuchillado o disparado según el grado. Si el policía tiene grado Novato o Detective será acuchillado, y si el grado es Investigador o Sargento recibirá un disparo.

Cuando el jugador se desvíe del recorrido que lo llevará hacia el ladrón, no podrá llegar a alguna ciudad siguiente del recorrido sin pasar en el orden correcto. Ej: si tengo un recorrido compuesto por New York, Buenos Aires y New Delhi, si el jugador desde New York se va por un camino incorrecto, no podrá llegar a Buenos Aires o New Delhi si no vuelve a New York para seguir con el camino correcto.

Según el grado de policía serán los objetos a robar. Para el caso de policía novato se robará un objeto común. Para un policía de grado Detective e Investigador se robará un objeto valioso y el grado Sargento habrá un robo de un objeto Muy Valioso.

Según el enunciado, a partir de un objeto el ladrón se escapa por una **x** cantidad de ciudades, entonces en total el recorrido constará de **x + 1** cantidad de ciudades. Por ejemplo, si el ladrón se roba un objeto común de Buenos Aires, el mismo podría escaparse por New York, Rio de Janeiro, Oslo y Tokio. El ladrón se escapó a través de cuatro ciudades, pero el recorrido que hace el policía es a través de cinco ciudades por situarse al principio en Buenos Aires, que es la ciudad donde se originó el robo.

Los archivos xml a leer de configuración (ciudades, pistas, objetos y ladrones) se buscarán en el mismo directorio que contenga el ejecutable de la aplicación. Por ejemplo, si tengo el jar ejecutable en el directorio **C:\** entonces el archivo **ladrones.xml** deberá estar en el directorio **C:\ladrones.xml**.

Los nombres de los archivos a leer son:

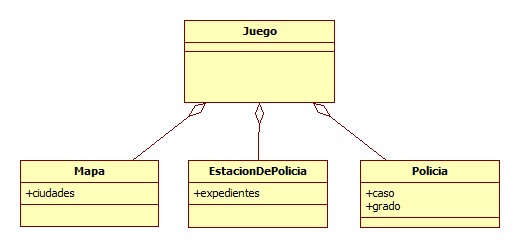
* **ladrones.xml**
* **objetos.xml**
* **pistas.xml**
* **ciudades.xml**

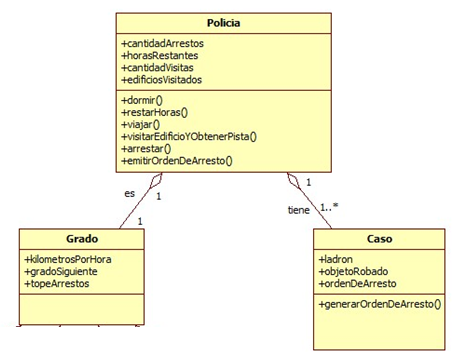
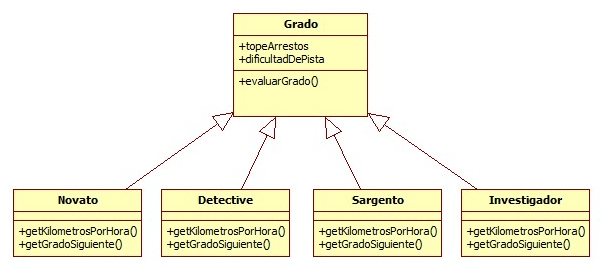
***Modelo de dominio***

Las entidades del modelo de dominio son:

* **Ladron**: representa al ladrón que roba el objeto de una ciudad y se escapa a través de las ciudades del recorrido. La clase Ladrón se compone de una CaracterísticaLadron, de un objeto Robable, de una ciudad actual y de su ruta de escape. El ladrón genera una lista de ciudades como inicio para su ruta de escapatoria que luego se encarga de informarle al mapa cuales son las ciudades para que a partir de ellas, el mapa pueda generar nuevas.
* **CaracterísticaLadron**: representa las características de un ladrón, como son el nombre, sexo, hobby, color de cabello, el vehículo que utiliza y característica que lo identifica como por ejemplo joyas.
* **Ciudad**: ciudad está compuesta por una InformacionCiudad, un nombre, sus tres posibles edificios y sus coordenadas. Son las ciudades que componen el Mapa y por las cuales el ladrón se va escapando.
* **InformacionMapa**: es un contenedor de ciudades leías desde el archivo de ciudades del XML.
* **InformacionCiudad**: contiene toda la información de una ciudad como nombre, colores de bandera, moneda, información geográfica e histórica, entre otros.
* **OrdenDeArresto**: es la orden de arresto generada por el policía. Contiene un CaracteristicaLadron con el cual se identificará al ladrón que es perseguido.
* **Mapa**: es el encargado de generar y manipular las ciudades del mundo a las que se pueden viajar. Por cada petición del jugador para ver cuáles son las ciudades disponibles que tiene para viajar a partir de una ciudad origen, da cuatro ciudades de destino posibles.
* **Caso**: Esta entidad se encarga de elegir un ladrón de la lista de ladrones, asignarle un objeto aleatoriamente. También controla el reloj del juego que es el indicador para saber si se terminó el caso o no por falta de tiempo.
* **Recorrido**: esta clase es la que se encarga de darle las opciones de ciudades a donde puede viajar el policía en busca del ladrón.
* **Juego**: inicializa el juego, asigna casos al jugador y los controla. Tiene dos estados: caso en progreso y caso no iniciado. Caso en progreso es cuando el jugador está persiguiendo al ladrón en su escapatoria y caso no iniciado es cuando se terminó un caso o el jugador recién abrió la aplicación. Contiene las opciones de juego para que el policía pueda perseguir al ladrón, obtener pistas y generar órdenes de arresto.
* **Grado**: El grado de policía se divide en **GradoNovato**, **GradoDetective**, **GradoInvesigador** y **GradoSargento**. Cada uno tiene una funcionalidad diferente, como devolver cantidad de kilómetros por hora que viaja, que tipo de objeto se roba el ladrón y que visitador tendrá para entrar a los edificios y obtener pistas.
* **Pista**: Las pistas se leen desde un XML diferenciado por dificultad y por una entidad determinada. Por ejemplo: dificultad Facil, entidad Moneda devuelve una pista determinada.

## Diagramas de clases

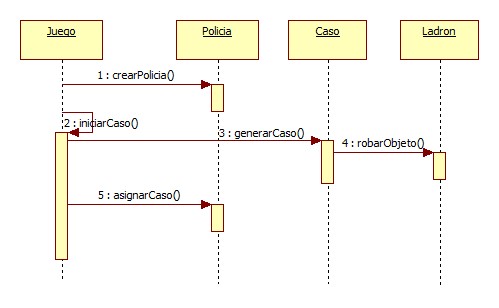


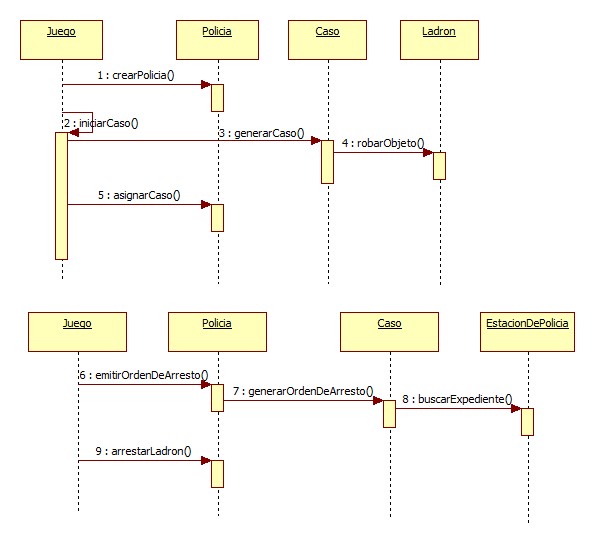
## 

## 

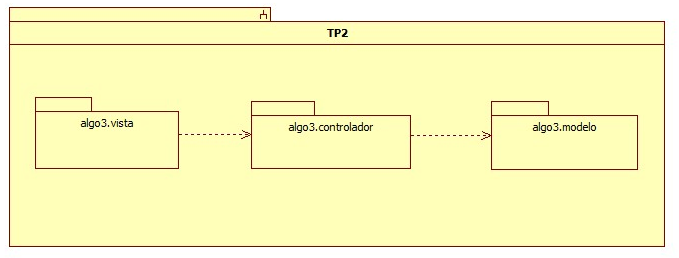
**Diagramas de Secuencia** Se crea un jugador (esto es interactivo desde GUI) y se le asigna un caso.

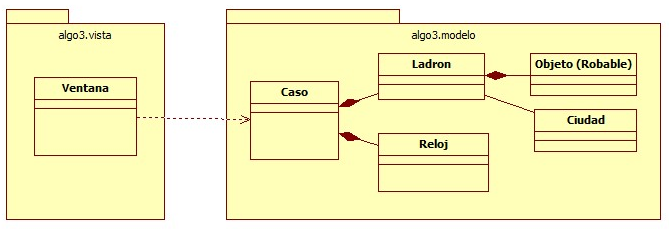


El policía arresta ladron.



**Diagramas de Paquete**





## Detalles de implementación

* **Generación de escape del ladrón:** cuando se llama al método robar de ladrón y se le pasa un objeto robable, el ladrón delega la generación de ciudades en **CiudadFactory** que es la encargada de armar el recorrido. Lo que hace **CiudadFactory** es pedirle a **InformacionMapa** la información disponible de ciudades que fueron leías de un archivo xml de ciudades. Conociendo la información sobre ciudades, esa lista se desordena para dar un poco de aleatoriedad a la generación de recorrido y lo que se hace es iterar esa lista e ir generando una ciudad a partir de la información de ella y también **información de la siguiente ciudad**. Esto es necesario para que cada edificio de una ciudad sepa cuál es la ciudad siguiente, hasta llegar a la ciudad del final de recorrido. El ladrón también deja su conjunto de características en todos los edificios para que esa información sea devuelta al jugador a medida que va visitando edificios.
* **Funcionamiento de Mapa:** el mapa se construye a partir de una lista de ciudades que eligió el ladrón para escaparse, lo cual es muy importante, porque el mapa en base a esas ciudades, generará nuevas. Uno no le puede pedir al mapa una ciudad cualquiera, sino que a partir de que el caso sitúa al jugador en una ciudad válida, el jugador deberá pedir a que ciudades puede viajar, y en caso de que no tenga ciudades disponibles, mapa elige de una lista de información de ciudades para crear nuevas ciudades y vincularlas entre sí.
* **Grados de policía:** el policía por tener distintos comportamientos según va progresando el jugador y aumentando la cantidad de casos resueltos, cambia la cantidad de kilómetros por hora que puede viajar y la dificultad de las pistas que se le otorgarán. Por lo cual se aplicó un patrón **Strategy** el cual consiste en que para cada grado de policía distinto haya comportamiento distinto. El paso de un grado de policía a otro se origina cuando el policía arresta a alguien y el grado de policía evalúa en base a la cantidad de arrestos si llegó a la cantidad tope de su grado. Si alcanzó la cantidad tope, entonces al policía se le asignará un nuevo objeto grado en base al grado que este tenía. Esto implica que cada grado conoce a su grado siguiente, exceptuando el grado sargento que en vez de devolver el grado siguiente, devuelve la referencia de él mismo, dejando al policía con el mismo grado sargento.
* **Creación de ciudades:** de la creación de ciudades y edificios se encargará **CiduadFactory**. Hay un conjunto de métodos estáticos que son llamados por mapa para la creación de ciudades. También creará edificios con sus entidades necesarias para la devolución de pistas acordes. Es una implementación del patrón de diseño **Factory**.
* **Modelo, vista y controlador:** la vista del juego será una ventana en la cual aparecerá información sobre el juego entero, como el tiempo transcurrido, la ciudad actual, los edificios a visitar y las ciudades posibles a viajar. Está implementado con el patrón **Observer-Observable**, en el cual el observado es el modelo, por ejemplo, el reloj y cada vez que transcurre tiempo, se actualiza en la vista. Los controladores tomarán los eventos producidos en la vista y transmitirán los mensajes al modelo y este se actualizará para reflejarse en la vista.

## Excepciones

CiudadNoEncontradaException: se eleva una excepción de este tipo cuando desde afuera se intena utilizar una ciudad no válida porque no figura en el mapa del juego. Las ciudades del juego hay que pedírselas al mapa. También se elevará esta excepción si, por ejemplo, en el archivo de objetos robados figura la ciudad **Roma** que luego en el archivo de ciudades no figura, por lo tanto no se podrá construir un objeto ciudad Roma.