FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 — Java

Algo-Thief

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

Integrantes	Padrón	Email
CAMURRI, Federico	106359	fcamurri@fi.uba.ar
REIMUNDO, Martín	106716	mreimundo@fi.uba.ar
MARTINO, Nicolás	105217	nmartino@fi.uba.ar
ORQUEDA, Carlos	101806	corqueda@fi.uba.ar
MORICI, Enrique	107115	emorici@fi.uba.ar

Índice

1. Introducción **2**

2. Supuestos **2**

3. Diagramas de clase **3**

4. Diagramas de secuencia **6**

5. Diagramas de paquete **14**

6. Diagramas de estado **17**

7. Detalles de implementación **18**
7.1. Clase Juego 18
7.2. Clase Policia 18
7.3. Interfaz OrdenDeArresto 18

8. Excepciones **19**

1. Introducción

El presente informe reune la documentación de la solución del segundo trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar una aplicación de funcionamiento similar al juego basado en el programa de los 90's 'Where in the world is Camren Sandiego' en el lenguaje Java utilizando los conceptos del paradigma de la orientación a objetos vistos hasta ahora en el curso.

2. Supuestos

Para la realización del presente Trabajo Práctico, se tuvieron en cuenta los siguientes supuestos:

- Emitir la orden de arresto, solo descontará tiempo la primera vez que es emitida. Aunque luego siga intentando emitirla no descontará tiempo.
- El jugador deberá dormir una vez que el horario dentro del juego llegue a las 20 Hs. Si el jugador necesita viajar y la duración del viaje es muy larga, de manera tal que llegará a su destino de madrugada o a la mañana siguiente, no dormirá.
- Dentro del juego, la el mapa es de 2 dimensiones, esto quiere decir, que se supone que se encuentran en un plano, y por ende las distancias entre ciudades describen una recta entre ambas, NO describen una curva como si el mapa tuviese forma esférica. Asimismo, la conversión utilizada para los cáclulos fue que cada grado (^{o}) de longitud ó latitud equivale a 100Kms.
- Los archivos que contienen las posiciones de las ciudades, así como también aquellos que contienen sus destinos y pistas, fueron confeccionados por nosotros (Se encuentran a disposición de la cátedra en el repositorio remoto de la aplicación, en formato .csv).
- El jugador deberá elegir entre tres destinos aleatorios provistos por el juego al momento de viajar. Entre los destinso mostrados, se encontrará la ciudad correcta para seguirle el paso al ladrón.
- Existen 5 lugares de interés diferente dentro del juego y, dependiendo de la ciudad, se mostrarán 3 de estos para que el jugador pueda elegir donde obtener la pista.

3. Diagramas de clase

A continuación, se mostrarán los diagramas de clases que describen la relación entre las distintas entidades del modelo.

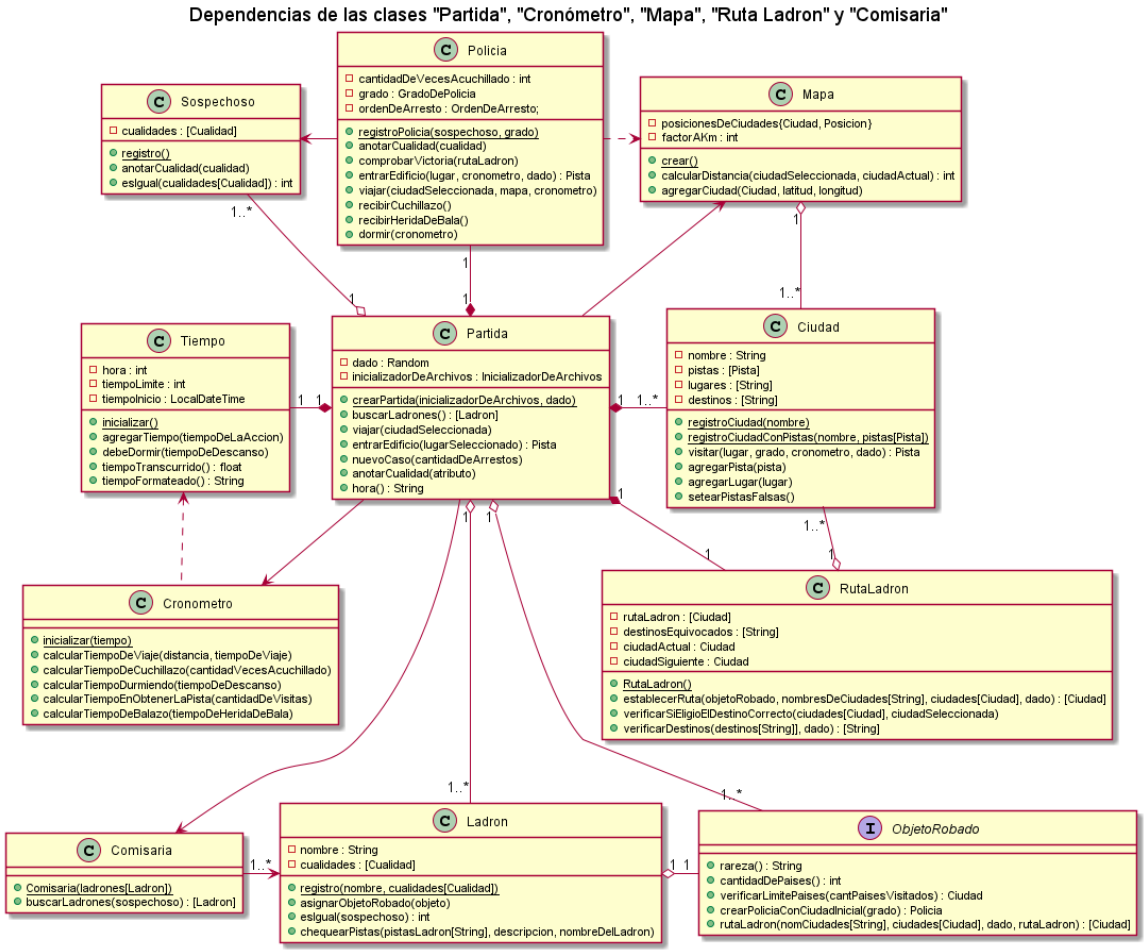

Figura 1: Interacciones de las clases 'Partida', 'Cronometro', 'Mapa', 'RutaLadron' y 'Comisaría'.

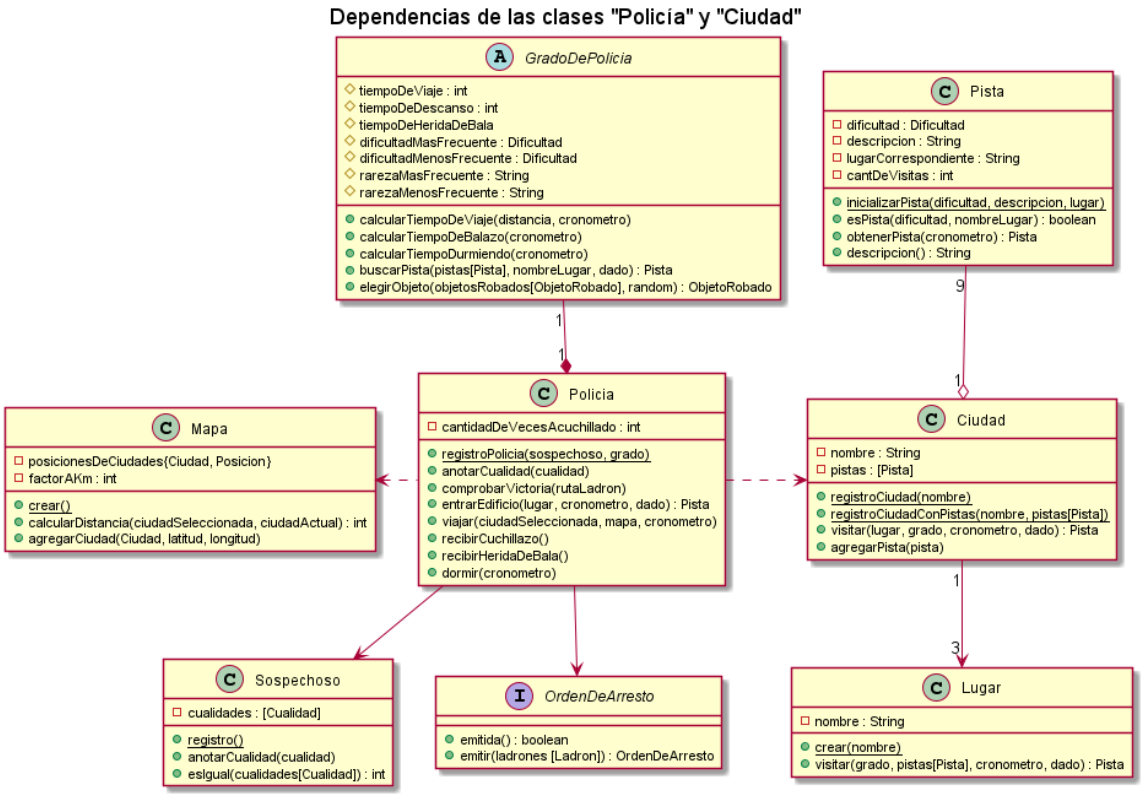

Figura 2: Interacciones de las clases 'Policía' y 'Ciudad'.

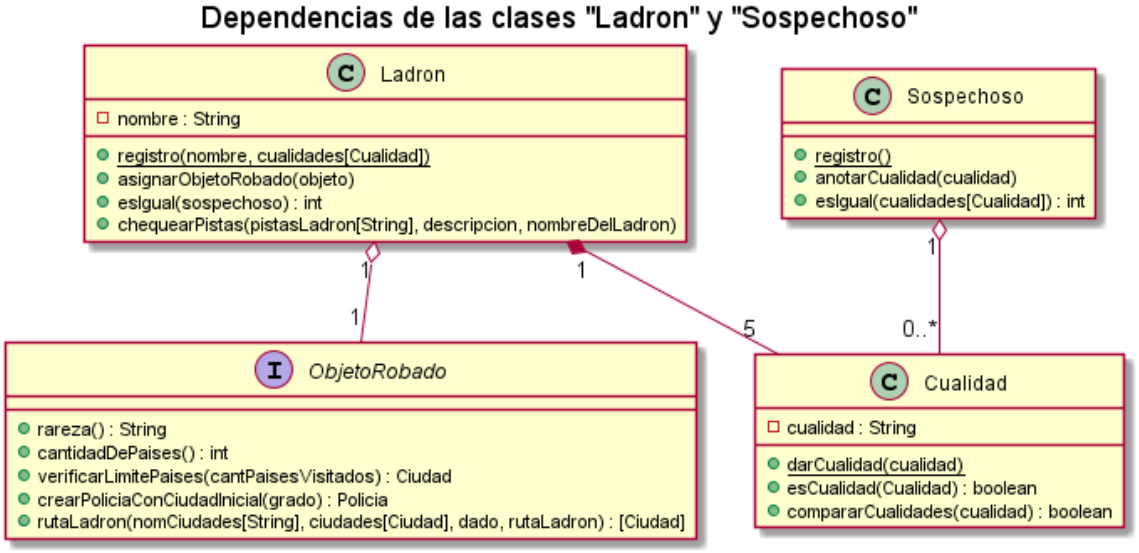

Figura 3: Interacciones de las clases 'Ladrón' y 'Sospechoso'.

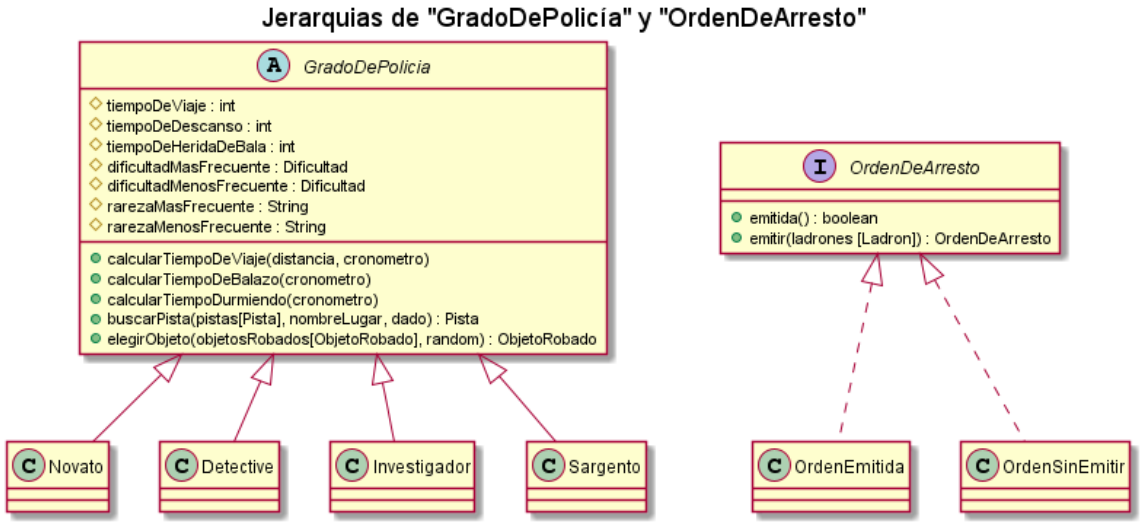

Figura 4: Jerarquías de la clase abstracta 'GradoDePolicia' y la intefaz 'OrdenDeArresto'.

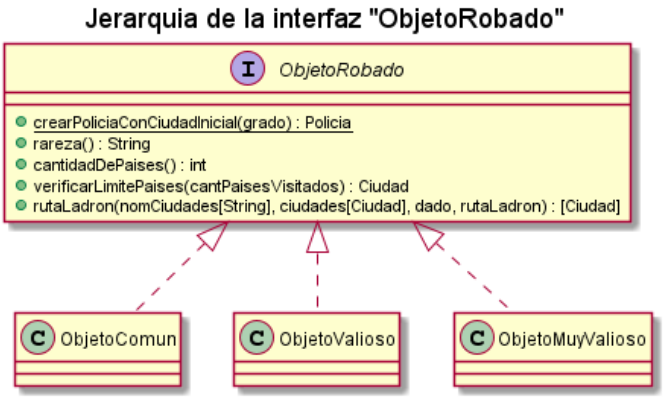

Figura 5: Jerarquías de la intefaz 'ObjetoRobado'.

4. Diagramas de secuencia

Los siguientes diagramas de secuencia corresponden a algunos casos de uso destacables dentro del funcionamiento de la aplicación.

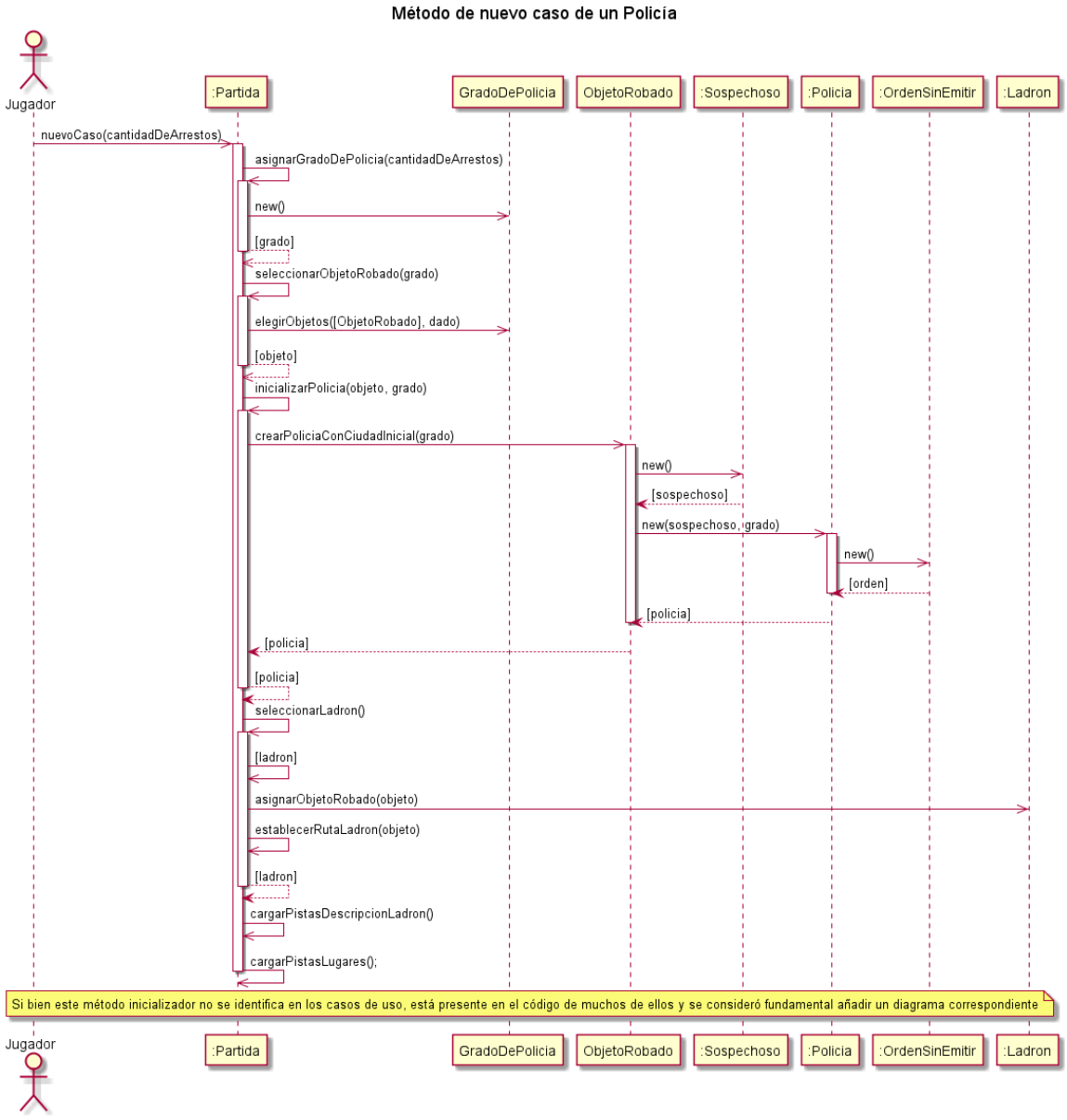

Figura 6: Creación de un nuevo caso que investigar.

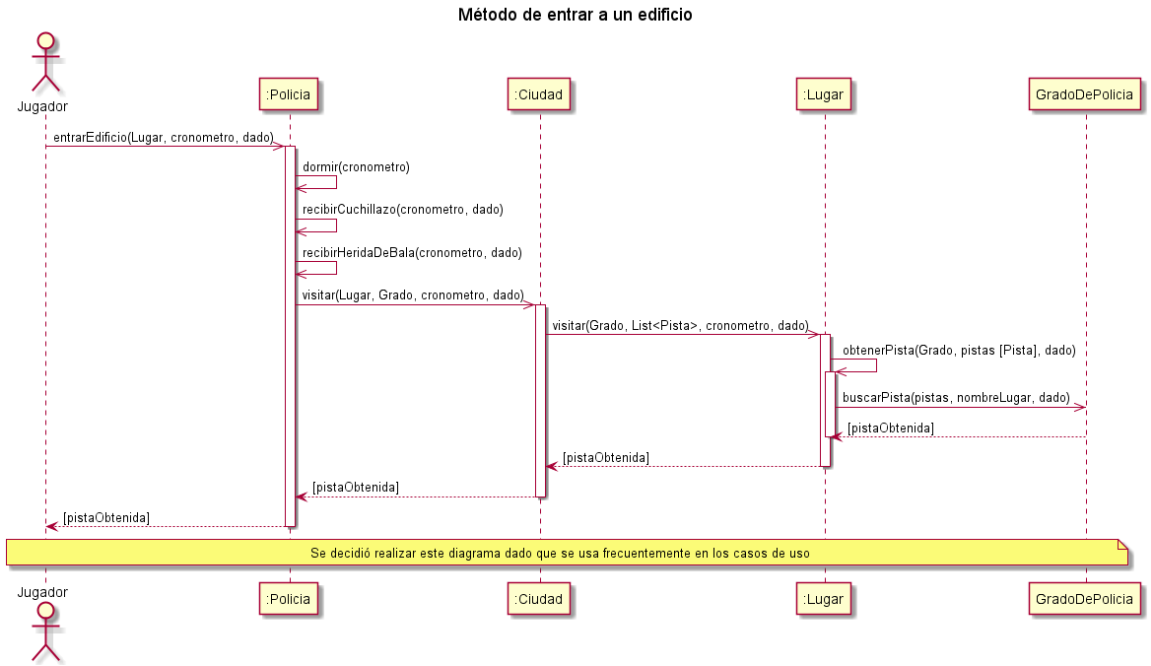

Figura 7: Jugador entra a un edificio.

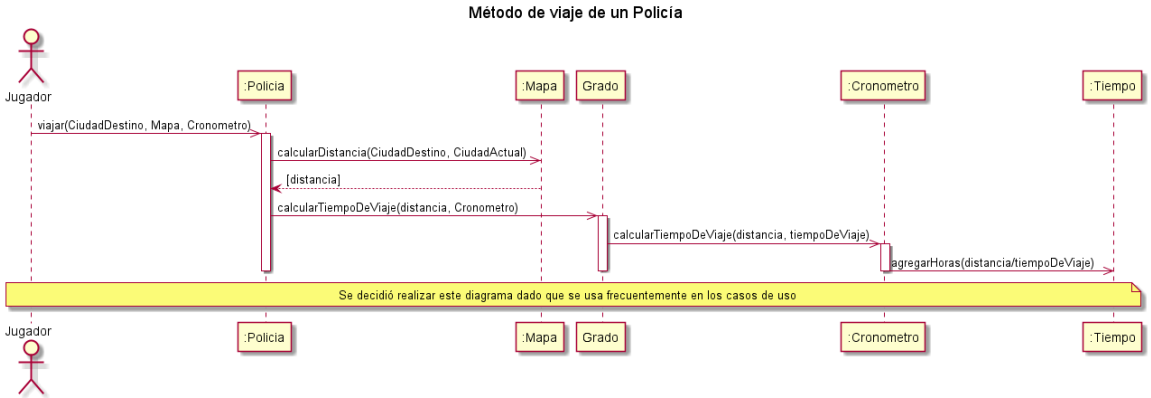

Figura 8: Jugador viaja a otra ciudad.

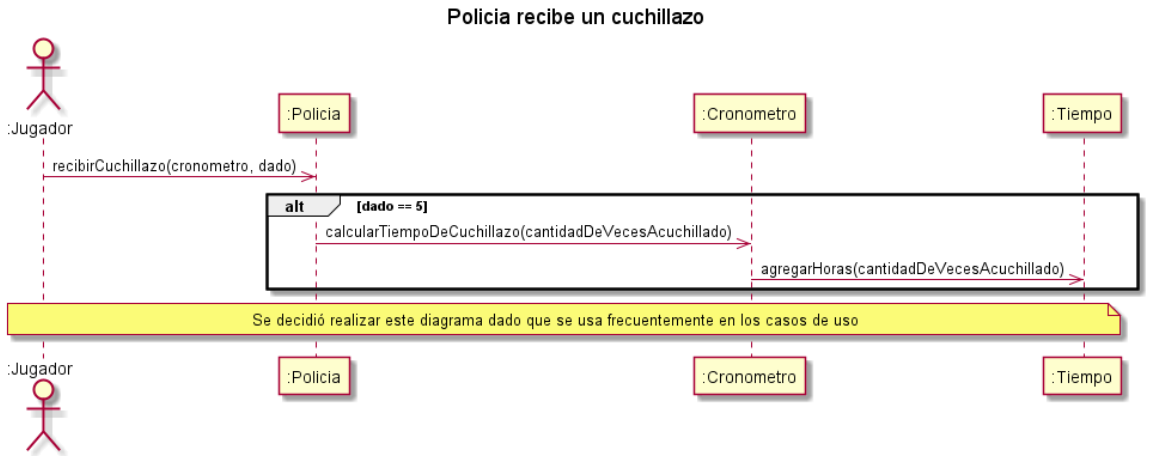

Figura 9: Jugador con rango detective recibe una herida de arma blanca.

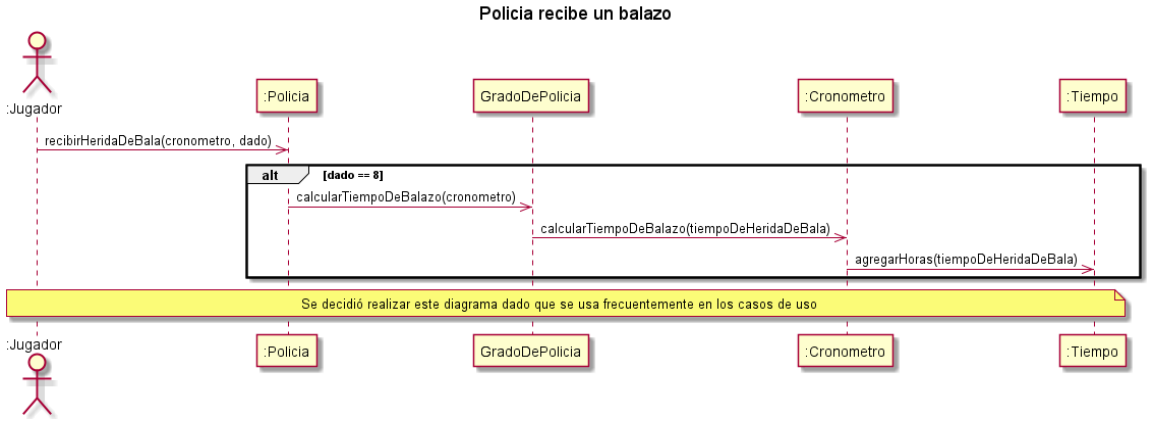

Figura 10: Jugador con rango detective recibe una herida de arma de fuego.

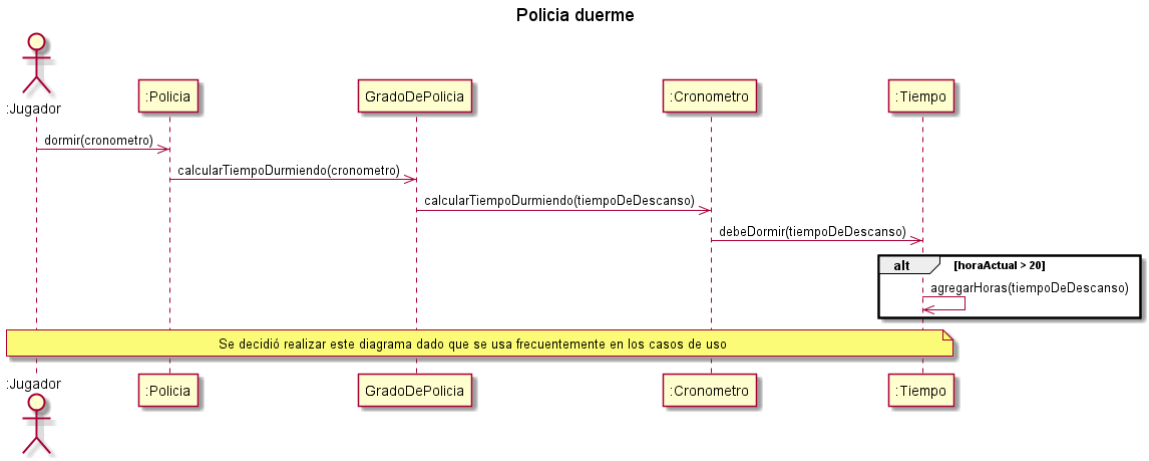

Figura 11: Jugador duerme.

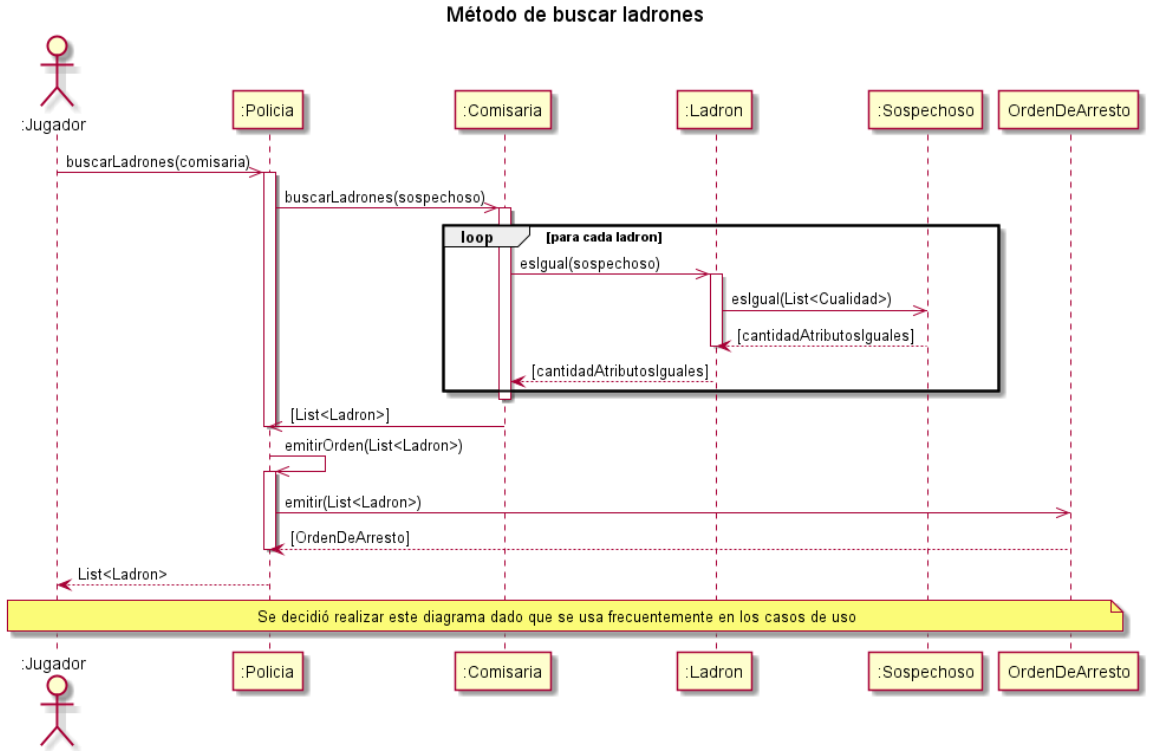

Figura 12: Búsqueda del ladrón dentro de la base de datos de sospechosos.

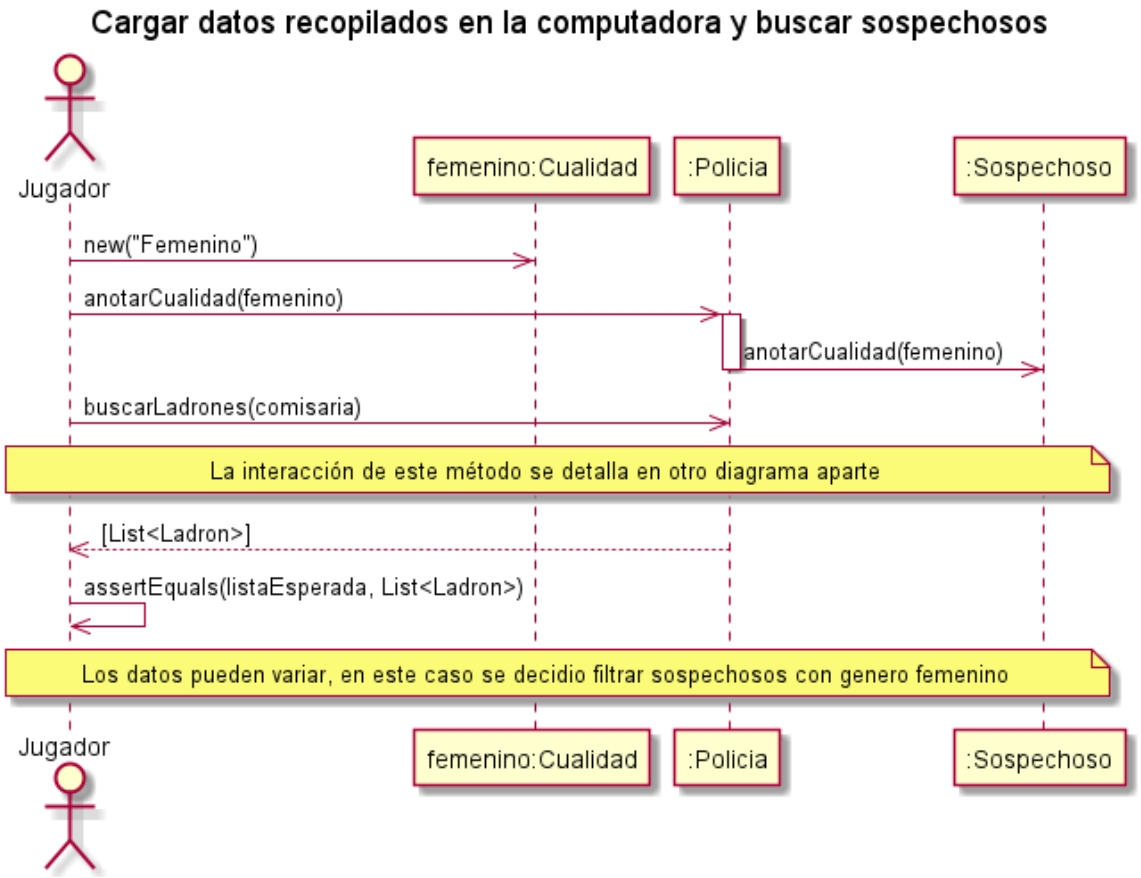

Figura 13: Secuencia donde el jugador carga los datos en la computadora y busca a los sospechosos.

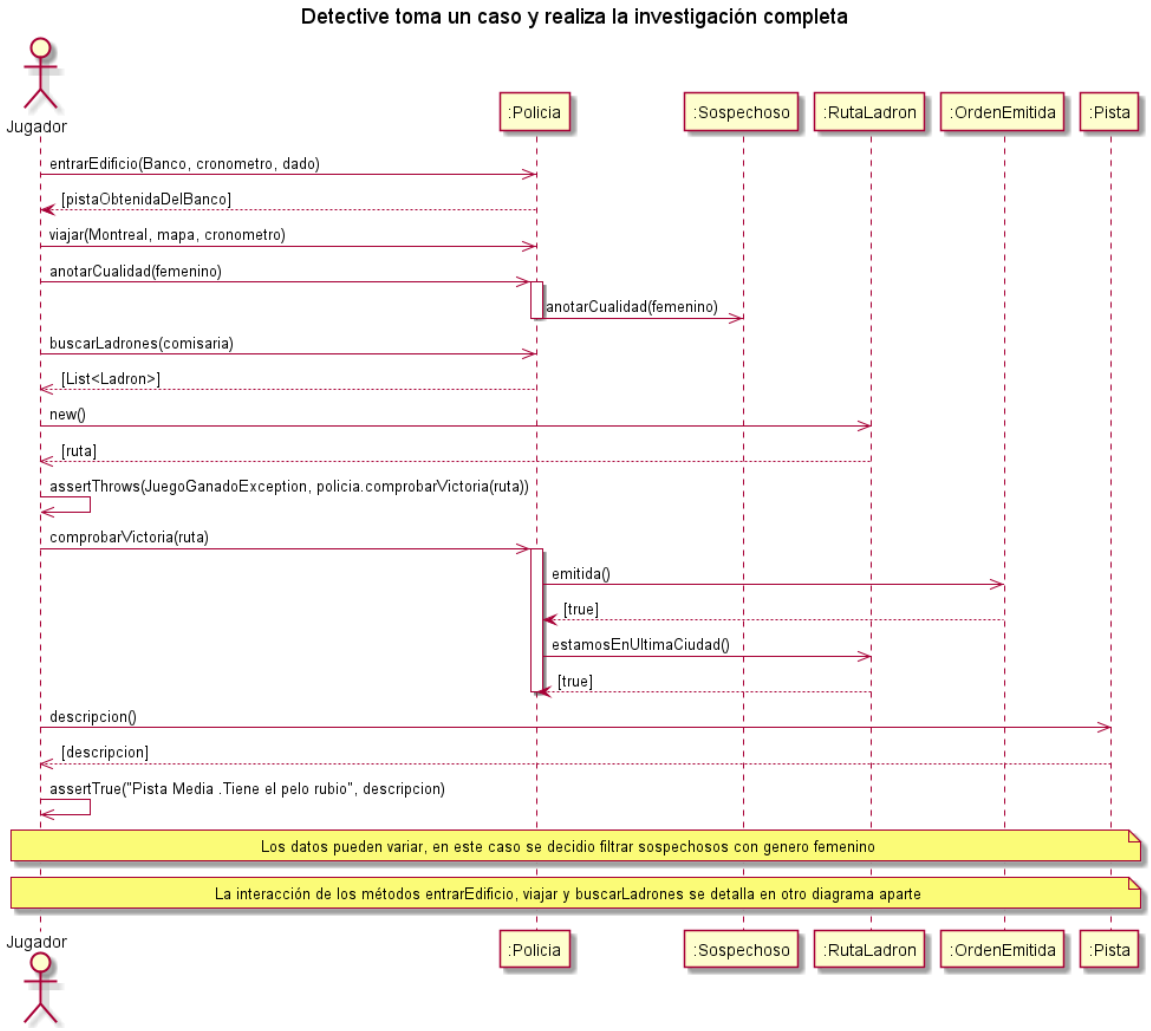

Figura 14: Jugador con rango detective toma el caso y realiza la investigación.

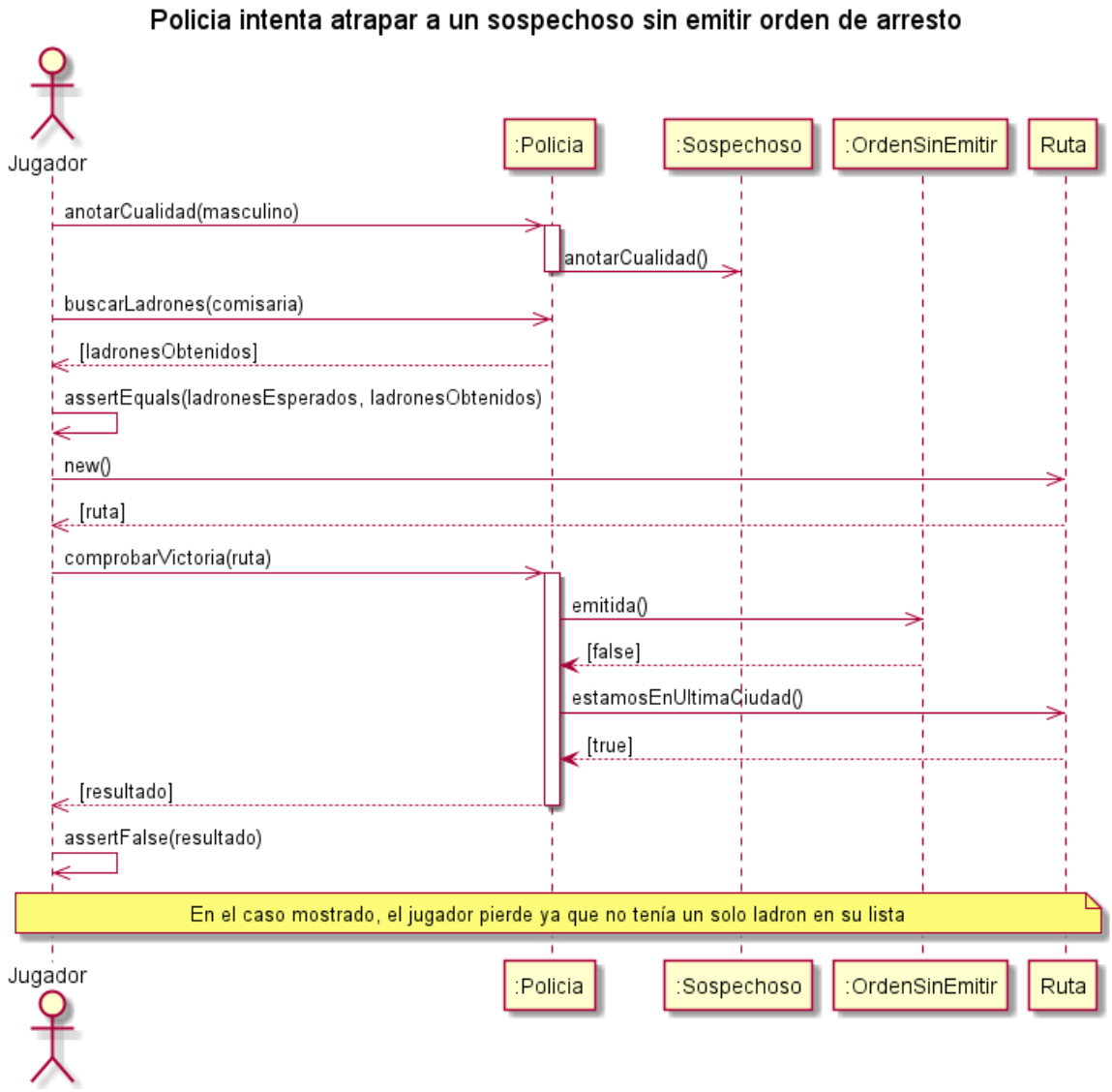

Figura 15: Jugador intenta atrapar a un sospechoso sin emitir la previa orden de arresto.

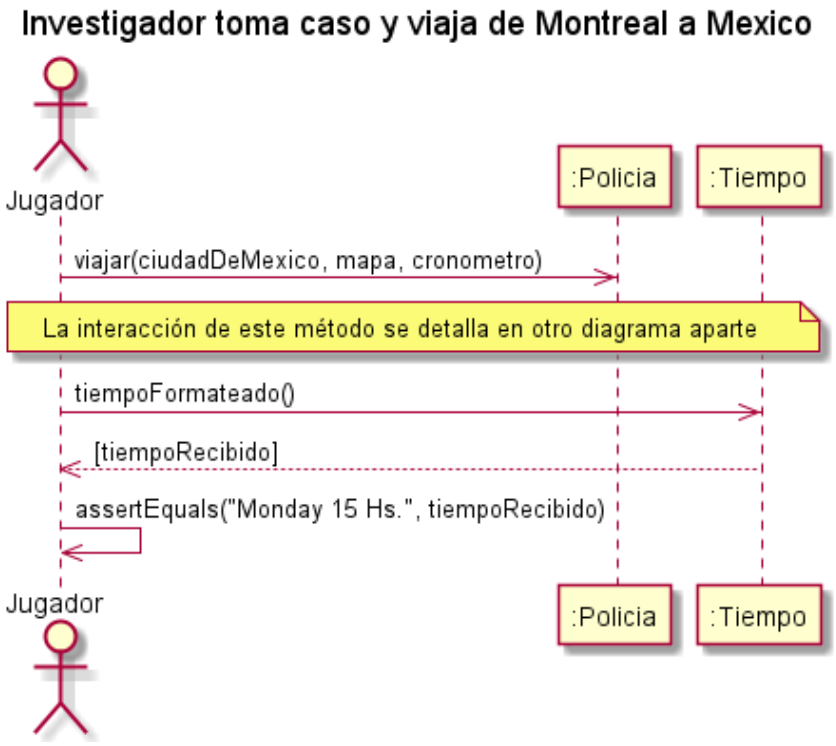

Figura 16: Jugador con rango investigador toma un caso y viaja a Montreal.

5. Diagramas de paquete

Estos diagramas muestran como se relacionan los distintos paquetes contenidos en la aplicación.

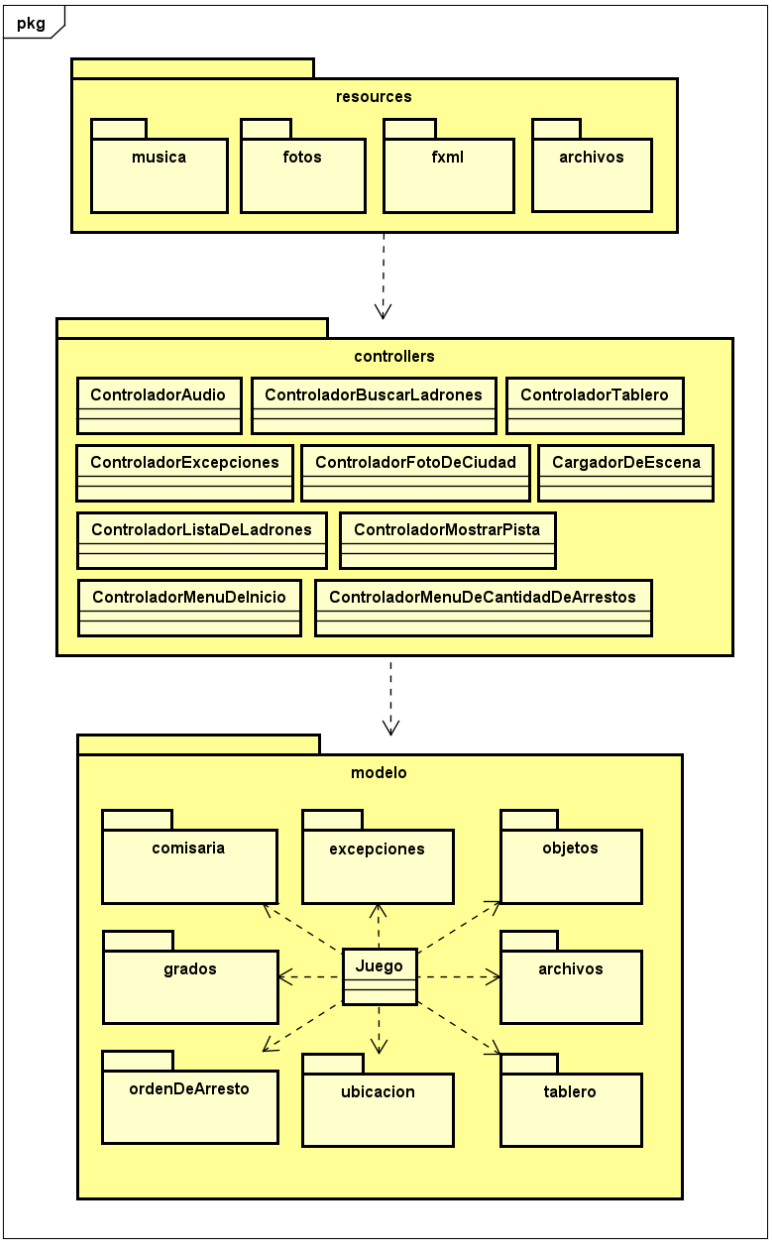

Figura 17: Diagrama de paquetes general de la aplicación.

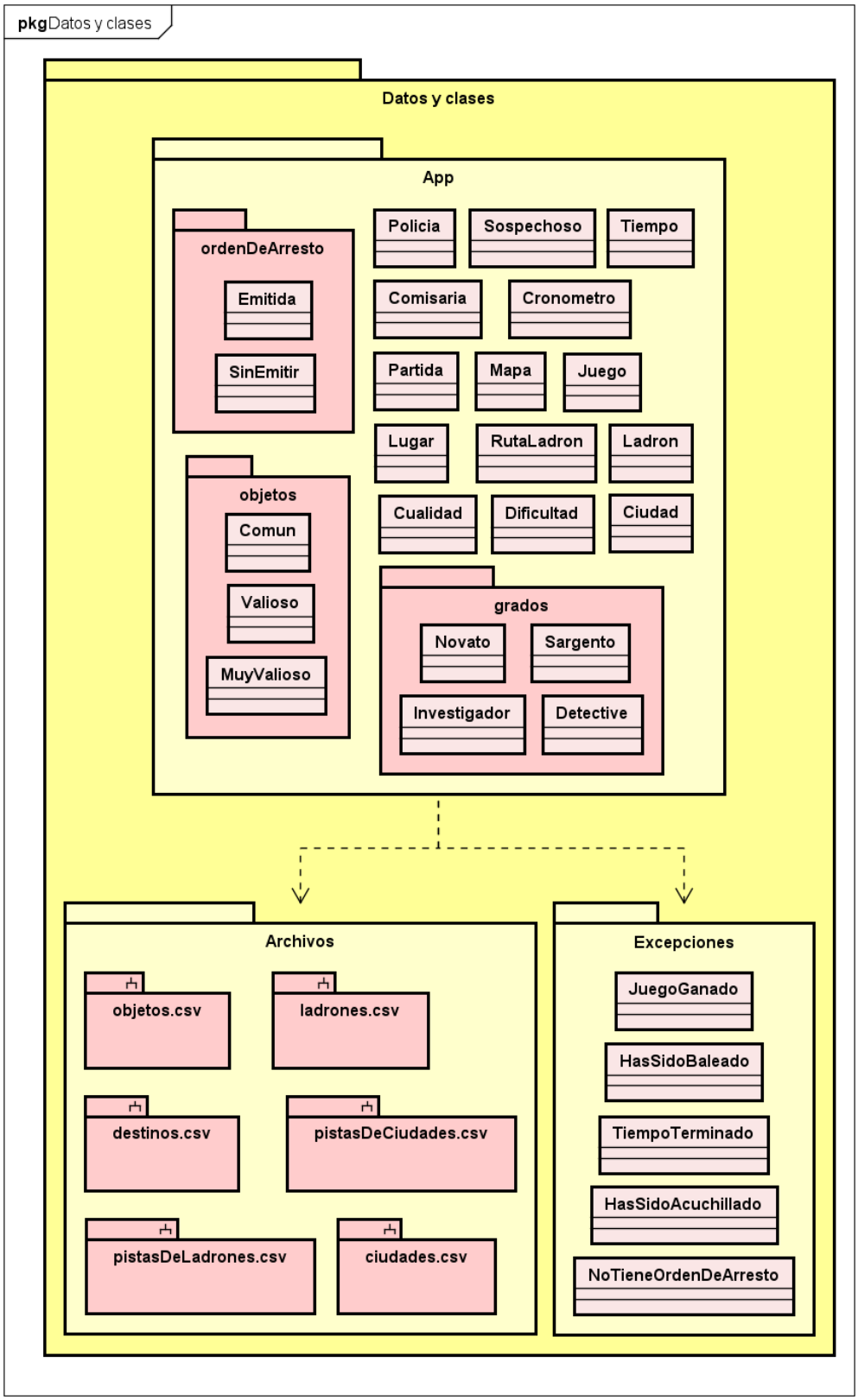

Figura 18: Diagrama de paquetes del modelo.

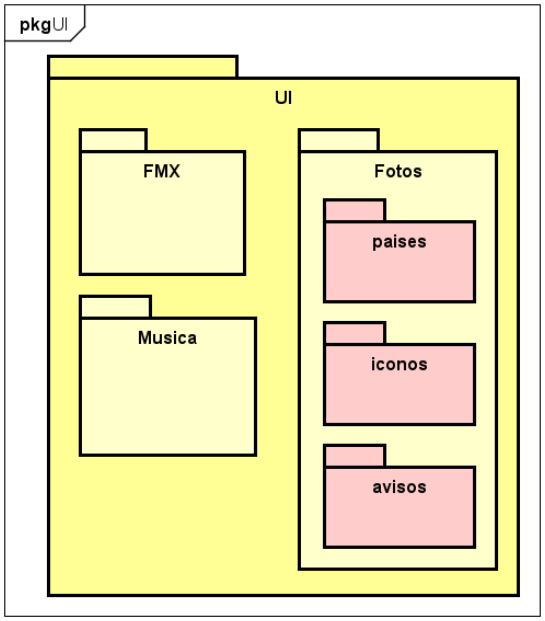

Figura 19: Diagrama de paquetes de la vista.

6. Diagramas de estado

Estos diagramas muestran el cambio de estado de una interfaz, en este caso de la interfaz 'OrdenDeArresto'.

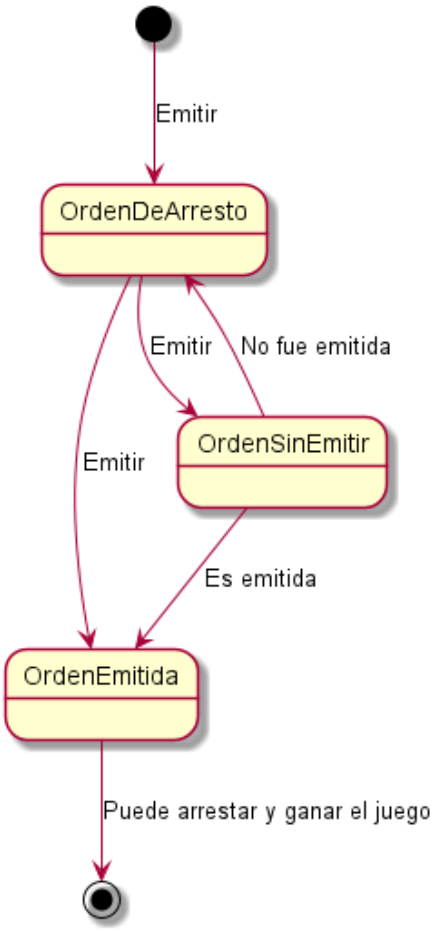

Figura 20: Diagrama de estado de la interfaz 'OrdenDeArresto'.

7. Detalles de implementación

7.1. Clase Juego

Es la clase principal del programa y la que dará vida al juego. Está compuesta por una instancia de sí misma y una instancia de Partida.Sus métodos son obtenerInstancia, viajar, entrarEdificio, buscarLadrones, anotarCualidad, nuevoCaso..

El método obtenerInstancia usa un patrón de diseño llamado SINGLETON, para lograr que se use siempre la misma instancia de la clase Juego.

```
public static Juego obtenerInstancia(){
    if(juego==null){
        juego = new Juego();
    }
    return juego;
}
```

El resto de los métodos de la clase son delegados a la clase Partida para no "sobrecargar"de responsabilidades a la clase Juego

7.2. Clase Policia

Esta clase está compuesta de un sospechoso, un grado de policia, una orden de arresto y la cantidad de veces que es acuchillado. Algunos de sus métodos son: entrarEdificio,viajar, comprobarVictoria, anotarCualidad, buscarLadrones..

El método comprobarVictoria se utiliza cada vez que el jugador entra a un edificio y puede cambiar el estado del juego, ya que puede ganar, perder, o seguir jugando

```
public void comprobarVictoria(RutaLadron rutaLadron){
    if(this.ordenDeArresto.emitida() && rutaLadron.estamosEnUltimaCiudad()){
        throw new JuegoGanadoException();
    }else if ((!this.ordenDeArresto.emitida()) && rutaLadron.estamosEnUltimaCiudad()){
        throw new NoTieneOrdenDeArrestoException();
    }
}
```

7.3. Interfaz OrdenDeArresto

Ésta interfaz posee un solo método, el cual se implementa de forma distina en sus clases hijas OrdenEmitida y OrdenSinEmitir.

OrdenEmitida:

```
public OrdenDeArresto emitir(List<Ladron> ladrones, Cronometro cronometro){
    return (new OrdenEmitida(this.ladron));
}
```

OrdenSinEmitir:

```
public OrdenDeArresto emitir(List<Ladron> ladronesPosibles, Cronometro cronometro) {
    if(ladronesPosibles.size() == 1){
        cronometro.calcularTiempoEnGenerarOrdenDeArresto(tiempoDeDemora);
        return (new OrdenEmitida(ladronesPosibles.get(0)));
    }
    return (this);
}
```

8. Excepciones

HasSidoAcuchilladoException Excepción que se arroja cuando el jugador ha recibido una herida de arma blanca al entrar a un edificio.

HasSidoBaleadoException Esta Excepción es lanzada cuando el jugador recibe una herida de arma de fuego al entrar a un edificio.

JuegoGanadoException Excepción lanzada cuando el jugador logró ganar el juego.

NoTieneOrdenDeArrestoException Esta excepción es arrojada cuando el jugador intenta atrapar a un sospechoso sin emitir una orden de arresto previamente.

TiempoTerminadoException La Excepción es lanzada cuando el tiempo se agota sin que el jugador haya logrado completar el juego.