Memoria técnica

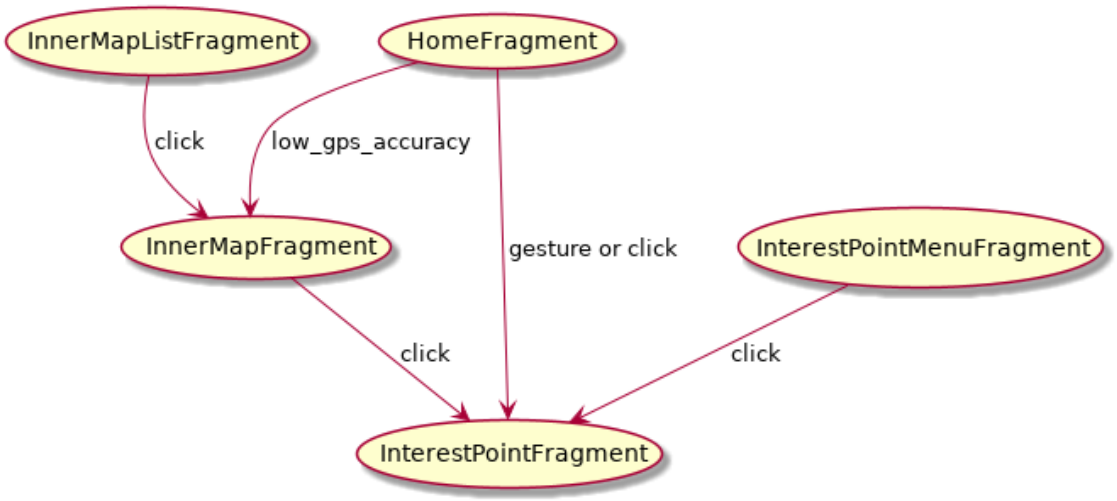
Luis Antonio Ortega Andrés
Guillermo Galindo Ortuño
Johanna Capote Robayna

1 Estructura del proyecto

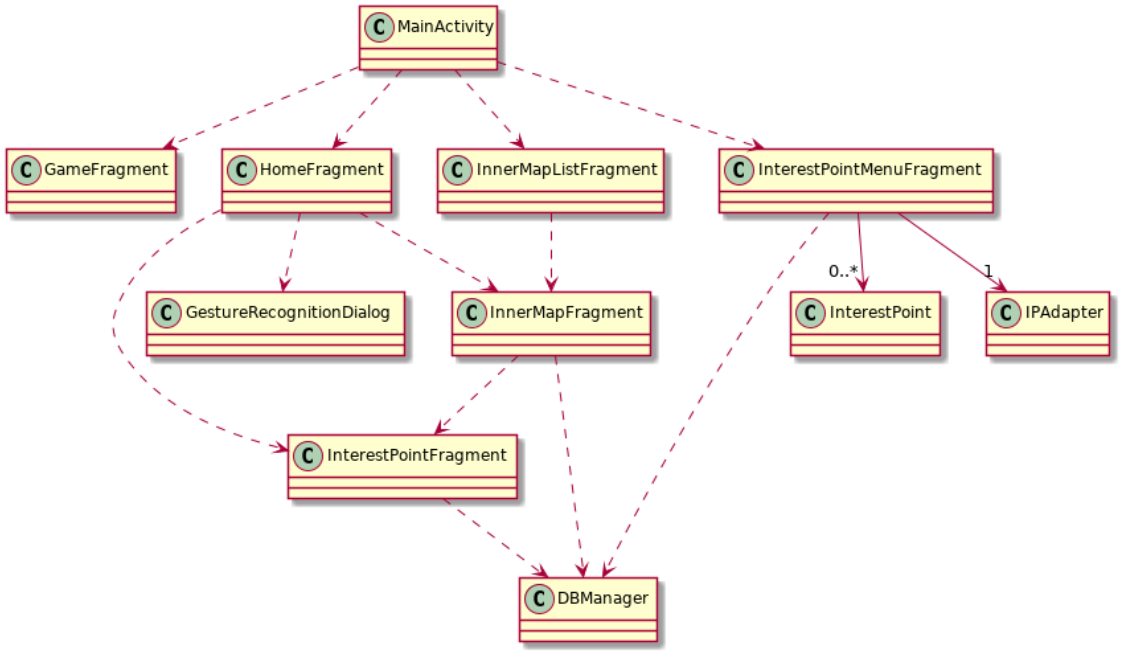
La aplicación `Alamabra1925` está implementada como un proyecto `Android` en `Kotlin`. El proyecto se encuentra estructurado en una única actividad `MainActivity` y una serie de fragmentos, cada uno de ellos encargado de una faceta de la aplicación. Estos son:

- `HomeFragment`.
- `InnerMapListFragment`.
- `InnerMapFragment`.
- `InterestPointFragment`.
- `InterestPointMenuFragment`.
- `GameFragment`.

Todos los fragmentos y las interacciones entre ellos se llevan a cabo utilizando un "controlador de navegación". Este nos permite navegar entre los fragmentos desde la barra de navegación lateral. Desde esta barra podemos acceder a `HomeFragment` (Map), `InnerMapListFragment` (Inside Maps) y `GameFragment` (Game).
A continuación mostramos el diagrama correspondiente a las comunicaciones entre cada uno de los fragmentos.



Simbolizaremos con flechas continuas la cantidad de veces que una clase contiene a la otra. Debido a la estructura con fragmentos del proyecto esto solo ocurre con la clase `InterestPointMenuFragment`. Por esto, hemos decidido simbolizar en el diagrama las comuninaciones entre las clases. Para ello usaremos lineas discontinuas.



2 Fragmentos

2.1 HomeFragment

2.2 InnerMapFragment

2.3 InnerMapListFragment

2.4 InterestPointFragment

Este fragmento nos permite visualizar toda la información acerca de un punto de interés concreto de nuestra visita. En el método `onCreateView` del fragmento, esperamos un `Bundle` con un identificador que nos permita conocer de que fragmento se debe mostrar la información.
Con esto utilizamos la base de datos para rellenar los `textView` apropiados.

2.5 InterestPointMenuFragment

A este fragmento se puede acceder desde la barra de navegación lateral bajo el nombre de `Interest Points Menu`. Aquí obtendremos una lista de todos los puntos de interés de os que se tiene información en la aplicación. Para ello utilizamos un adaptador `IPAdapter` y la base de datos.
Cada uno de los puntos de la lista tiene sobrecargado el listener de pulsación, haciendo que sea posible ir a la vista detallada de cada uno de ellos.
En el método `onCreateView` hacemos lo siguiente.

```
val mainList = root.findViewById<ListView>(R.id.list_view)
listAdapter = IPAdapter(this.context!!, list)
```

```
mainList.adapter = listAdapter

mainList.onItemClickListener = AdapterView.OnItemClickListener { _, _, position, _ ->
    val bundle = bundleOf("id" to list[position].id)
    findNavController().navigate(R.id.action_nav_ip_menu_to_nav_ip, bundle)
}

loadQueryAll()
```

- Inicializamos la variable correspondiente a la vista de la lista.
- Inicializamos nuestro adaptador. Y se lo asignamos a la lista.
- Sobrecargamos el listener de pulsación de forma que utilice controlador de navegación para cargar el fragmento del punto de interés correspondiente.
- Utilizamos `loadQueryAll` para rellenar la lista con los puntos de interés de la base de datos.

2.6 GameFragment

En este fragmento el usuario se enfrenta a una serie de preguntas que tendrá que responder lo más rápido posible.

3 Otras clases

3.1 InterestPoint

Esta clase corresponde a una simple abstracción de la información almacenada en un punto de interés.

```
data class InterestPoint(
        var id : Int,
        var title: String,
        var content : String
)
```

3.2 IPAdapter

Esta clase extiende a la clase `BaseAdapter`. La utilizamos en el fragmento correspondiente a la lista de puntos de interés `InterestPointMenuFragment` donde mostramos uan lista de todos los puntos de interes que se encuentran en la base de datos. El adaptador nos permite mostrar estos elementos en la lista de una forma más cómoda y personalizable. Podriamos haber utilizado un `ArrayAdapter` como se ha hecho en la lista de mapas interiores `InnerMapListFragment`, sin embargo, esto bloqueaba la escalabilidad de la aplicación a la hora de querer mostrar mas información en la lista (por ejemplo, imágenes o un resumen del contenido).

3.3 DBManager

Esta clase nos permite utilizar la API de `SQLite` para almacenar los puntos de interés de nuestra aplicación. Dado que estos no cambian a lo largo de la ejecución podríamos haber utilizado un

vector que los almacenara, sin embargo, vimos mas conveniente este modelo, debido a que todos los fragmentos se pueden valer de la misma base de datos (por tener solo una actividad) y no tenemos que preocuparnos de tener un vector al que todos pudieran acceder.
La información almacenada en la base de datos es:

- Un identificador numérico. La clave primaria.
- El nombre del punto de interés.
- Toda la información al respecto del mismo.
- Su possición.
  - Se utiliza un valor numérico para indicar si el punto de interes se encuentra en el interior o dentro de algún edificio.
  - En el caso de ser un punto exterior, se utilizan latitud y longitud para determinarla.
  - En caso de ser un punto interior, se utilizan un sistema de márgenes para posicionarlo en el mapa.

La clase dispone de 3 métodos importantes, todos ellos nos devuelven un cursor iterable sobre las filas resultantes en la base de datos.

- `queryById`. Nos permite obtener un punto de interés a partir de su ID.
- `queryByLocationType`. Nos permite obtener todos los puntos de interés que se encuentren en el exterior o dentro de un mismo edificio.
- `queryAll`. Nos permite obtener todos los puntos de interés.

3.4 GestureRecognitionDialog.