



## Ejercicio Práctico de Aprobación “Aplicación Móvil SIU GUARANÍ”



### DOCENTES:

»Profesor Adjunto: Lic. FIRMENICH, Diego.

»JTP: Lic. SCHANZ, Sebastián.

### INTEGRANTES:

CALFUQUIR, Jorge Nicolás

PARRA, Iván Javier

SANTOS, Carla



# **INTRODUCCIÓN**

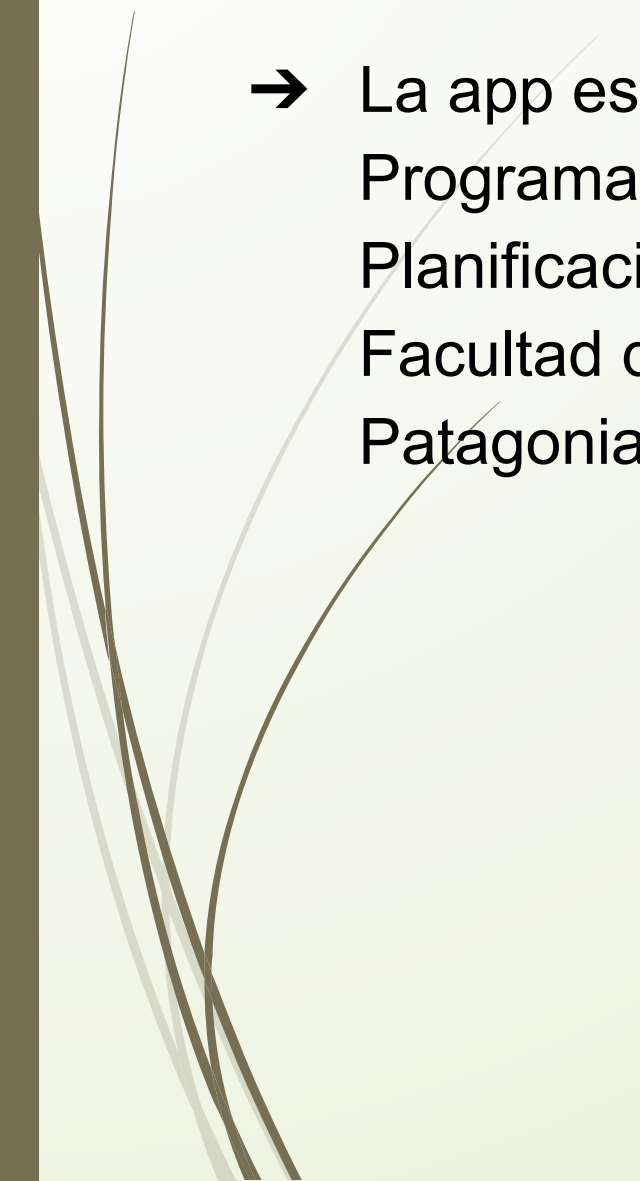
# Aplicación Móvil SIU GUARANÍ...

## ¿De qué se trata?

- Se trata de una aplicación móvil nativa en Android que realiza web scraping sobre el Sistema de Gestión de Alumnos de la Facultad de Ingeniería Sede Trelew de la UNPSJB llamado SIU GUARANÍ. Esta aplicación permite al alumno:
  - ◆ Iniciar sesión en el Sistema SIU GUARANÍ desde el dispositivo móvil.
  - ◆ Ver mesas de examen e inscripciones.
  - ◆ Inscribirse/desinscribirse a una mesa de examen.
- Periódicamente, sin la intervención del usuario, se verifica si hay o no mesas disponibles y si está inscripto o no a la misma. Se le informa al mismo a través de determinados mecanismos que ofrece Android.



## ¿A quién está destinada?


- La app está destinada a los alumnos de las carreras Analista Programador Universitario (APU) y Licenciatura en Sistemas (Or. Planificación, Gestión y Control de Proyectos Informáticos) de la Facultad de Ingeniería Sede Trelew de la Universidad Nacional de la Patagonia San Juan Bosco (UNPSJB).
- 



# **FUNDAMENTOS DE ANDROID**



# ¿Qué es Android?

- Android es una plataforma basada en el sistema operativo Linux, diseñada principalmente para dispositivos móviles con pantalla táctil, como smartphones, tablets y smartwatches. Hoy en día se usa además en televisores y automóviles.
- 

# Características de Android

## → Android combina:

- ◆ Software Libre: Apache, SQLite, WebKit, etc.
- ◆ Portabilidad: Aplicaciones de usuario escritas en Java.
- ◆ Estándares bien conocidos: UI basada en XML.

## → Android ofrece:

- ◆ Muchos servicios para las aplicaciones: Localización, BD, sensores, etc.
- ◆ Alto nivel de seguridad: La seguridad es administrada por la plataforma.
- ◆ Optimización: Kernel y máquina virtual optimizados para dispositivos móviles.
- ◆ Alta calidad para la reproducción de contenido multimedia: Basado OpenGL y distribuyendo codecs de audio y video más populares.
- ◆ Excelentes herramientas de desarrollo.
- ◆ Grandes comunidades de desarrollo.
- ◆ Mucha documentación y material.

## ¿Por qué Android?

- La elección de esta plataforma para el desarrollo de nuestra aplicación se debe a que la misma posee mayor cantidad de usuarios respecto a otras plataformas, es de código abierto, gratuita y está respaldada por una Alianza de 84 empresas mundialmente reconocidas llamada Open Handset Alliance (OHA) que busca sostener y hacer crecer Android. Dentro de estas empresa podemos mencionar: Telefónica. Telecom, Asus, Acer, Intel, Google, Ebay, Samsung, etc.
- Su gran popularidad ha hecho que cuente con la comunidad mundial más grande de desarrolladores, con múltiples eventos, concursos, competencias y reuniones así como diversas vías de comunicación como foros y chats oficiales para fomentar la participación y la colaboración para encontrar mejoras e ideas para futuras versiones.



# Arquitectura de Android

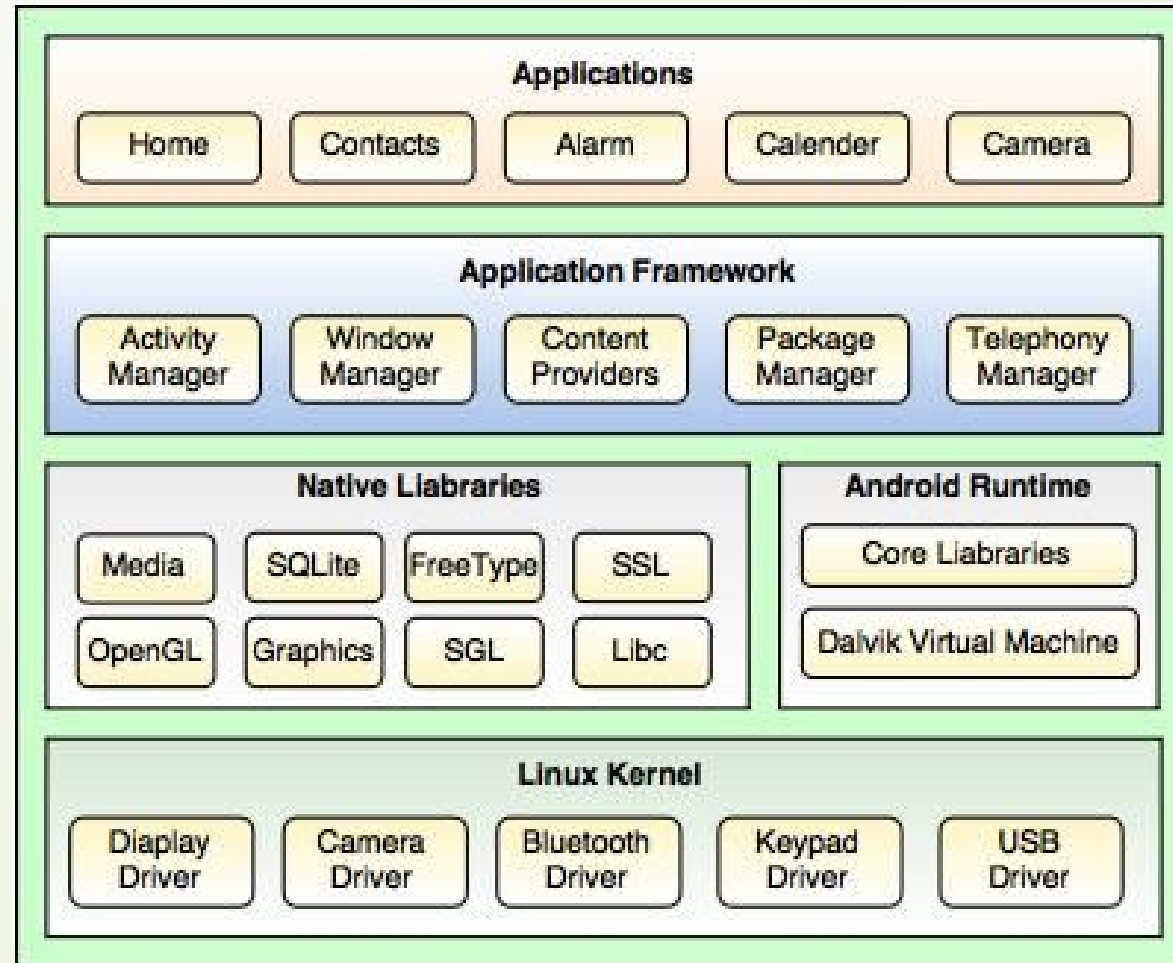
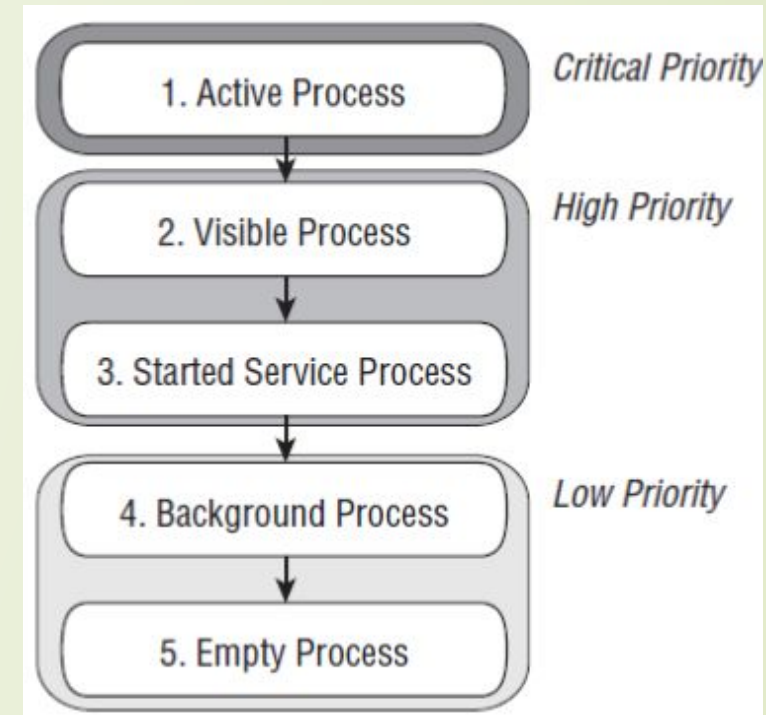


Fig. Android Architecture

# Ciclo de Vida de los procesos en la plataforma

→ La plataforma Android crea un proceso para cada una de las aplicaciones cuando estas son ejecutadas. Además, se encarga de administrar el ciclo de vida de las aplicaciones y de sus componentes manipulando los procesos en los que estas se ejecutan. Los procesos continúan en memoria tanto como sea posible, Cuando hace falta memoria, la plataforma decide que proceso finalizar.



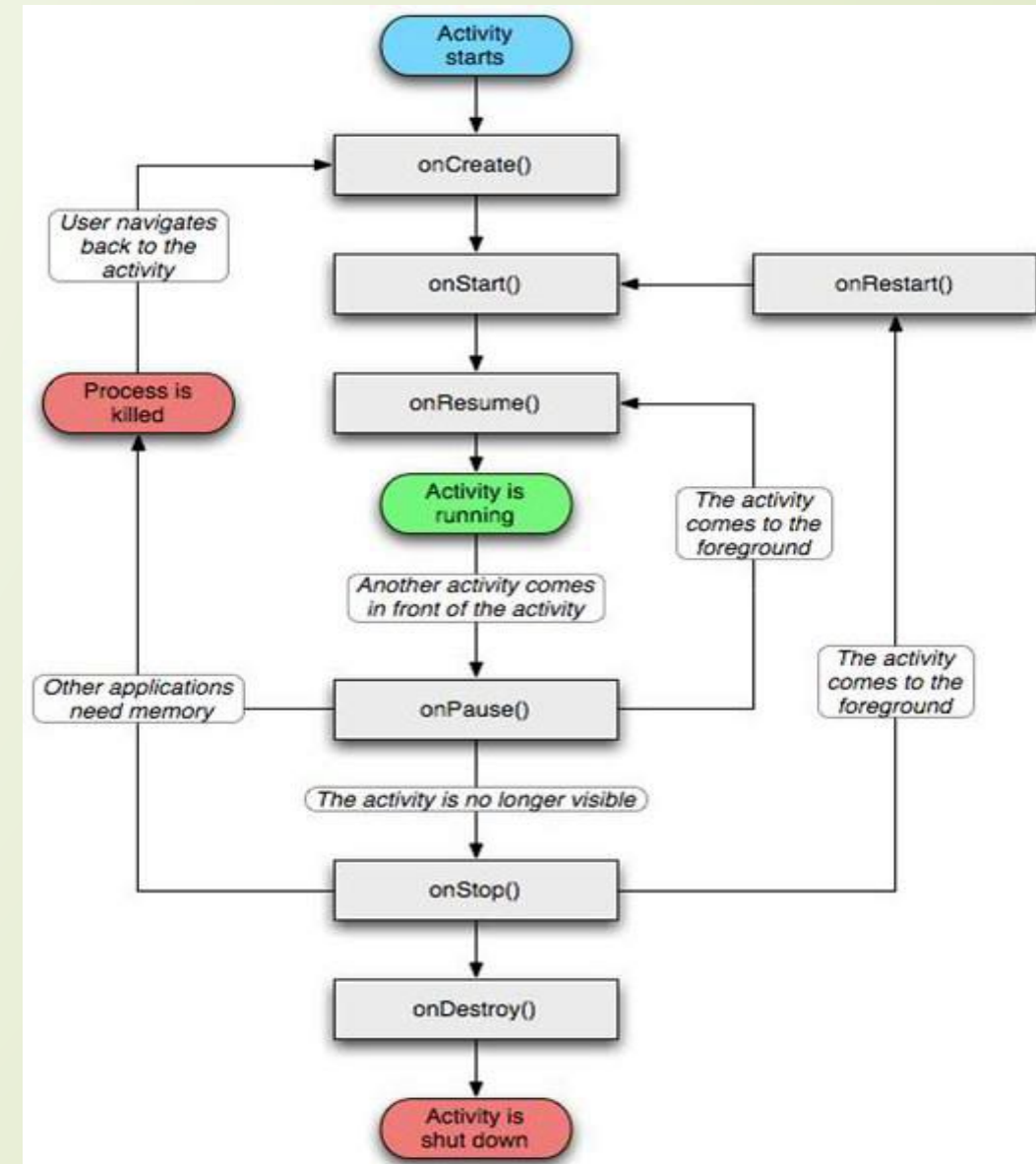
1. El “Active/foreground Process” es el proceso que contiene la actividad que el usuario está utilizando.
2. Los “Visible Process” son aquellos que contienen porciones visibles de la pantalla, pero que no están al frente.
3. Los “Service Process” son aquellos que ejecutan un servicio en segundo plano.
4. Los “Background Process” son aquellos que contiene una aplicación que no es visible al usuario.
5. Los “Empty Process” son aquellos que no contiene una aplicación, pero el proceso se mantiene para evitar la creación de uno nuevo cuando haga falta.

# Componentes de una Aplicación Android

- Una aplicación Android puede estar compuesta por cuatro tipo de componentes principales:
  - ◆ Activities.
  - ◆ Services.
  - ◆ Broadcast Receivers,
  - ◆ Content Providers.
- Todo componente de una aplicación debe estar declarado en el Manifest de la aplicación.

# Activities

- Las actividades son piezas ejecutables que pueden ser instanciadas por el usuario o por la plataforma. Pueden interactuar con el usuario, o con otras actividades y servicios. Las actividades muestran pantallas de usuario, y para cada actividad se muestra una pantalla al usuario.
- El ciclo de vida de las actividades está diseñado para ahorrar recursos. Las actividades no están activas constantemente. El Framework manipula las actividades enviándoles mensajes para activarlas, iniciarlas, mandarlas al segundo plano, pausarlas, resumirlas, desactivarlas.



# Services (1)

- Un servicio es un componente de aplicación que puede ejecutar operaciones a largo plazo en segundo plano. Estas operaciones son llevadas a cabo sin una interfaz de usuario. Los otros componentes de la Aplicación pueden lanzar un servicio y éste se continuará ejecutando en segundo plano incluso cuando el usuario pasa a utilizar otra Aplicación.
- Los servicios pueden funcionar de dos modos:
  - ◆ Iniciados (Started)
  - ◆ Ligados (Bounded)
- Un servicio puede funcionar de ambas formas (iniciado y ligado). Para ello debe implementar dos callbacks: `onStartCommand()` y `onBind()`.

## Services (2)

Start Service	Bind Service
<p>Los inician otros componentes de la aplicación (ej. Una actividad) para realizar una tarea en segundo plano. Para ello, deben ejecutar el método <code>startService()</code>. Pueden correr en segundo plano tanto tiempo como sea requerido, incluso si el usuario deja de usar el componente que originó su ejecución. Generalmente realizan una única acción y finalizan sin devolver resultados a quién lo inició.</p>	<p>Son servicios que se ejecutan en segundo plano con la intención de recibir solicitudes cliente/servidor, desde uno o varios componentes de la aplicación. Las aplicaciones se ligan a estos servicios a través del método <code>bindService()</code>. El servicio ligado podrá ofrecer una interfaz cliente servidor, permitiendo a los otros componentes interactuar con él enviando mensajes y recibiendo respuestas. El servicio finaliza cuando no hay componentes ligados a él.</p>



## Services (3)

- Los servicios, por omisión, corren sobre el hilo de ejecución principal del proceso de aplicación que les dio lugar.
- Por ello debemos tener presente que si haremos operaciones que pueden bloquear el hilo de ejecución, necesitaremos un nuevo hilo para nuestro servicio, o bien para la tarea que requiere dicho procesamiento. Aquí es donde viene la separación entre Service e Intent Service.
- Un Service comparte el proceso de ejecución con la Aplicación. (Pueden ser iniciados o ligados).
- Un Intent Service es un tipo especial de Servicio que no comparte el proceso de ejecución con la Aplicación, son utilizados para realizar tareas en segundo plano que puedan bloquear el hilo principal.

## Services (4)

Service	Intent Service
<p>Usamos un Service cuando requerimos un servicios en el segundo plano por un tiempo indeterminado e independientemente de los componentes que puedan interactuar con él. No debe bloquear el UI THREAD. Se pueden lanzar tareas asincrónicas desde el servicio para realizar tareas de mayor procesamiento en otro hilo.</p>	<p>Un Intent Service está diseñado para ejecutar una determinada tarea en un hilo independiente al hilo principal de la aplicación. No es ni más ni menos que un tipo particular de servicio Android que se preocupará por nosotros de la creación y gestión del nuevo hilo de ejecución y de detenerse a sí mismo una vez concluida su tarea asociada. Es esencialmente útil si necesitamos realizar varias veces las mismas tareas y queremos apilar las solicitudes en un worker distinto al de la UI principal. Usualmente sin comunicación con el hilo principal.</p>



# Broadcast Receivers

- Los Broadcast Receivers son componentes que nos permiten escuchar por eventos que se difunden en la plataforma. Responden a las solicitudes de otras aplicaciones.
- Los Broadcast Receivers, responden mensajes/anuncios enviados a todo el sistema. Los mensajes pueden provenir de la plataforma (Por ejemplo: batería baja) o de una aplicación. Nuestra aplicación escucha mensajes desde ambos lugares.
- Para crear un Broadcast Receiver nuestra clase debe extender de `BroadcastReceiver` y debe implementar el método `onReceive()`. Además, en el archivo `AndroidManifest.xml` indicamos qué evento debe atender.

# Content Providers

- Es un componente de aplicación que permiten compartir datos entre las aplicaciones. Un Content Provider utiliza una interfaz en forma de URI para permitirle a las otras aplicaciones acceder a sus datos. Las otras aplicaciones no necesitan saber que aplicación responde su solicitud, simplemente utilizan la uri.



# **WEB SCRAPING**

# ¿Qué es Web Scraping?

- Scraping es un término que, traducido al español, literalmente quiere decir “rascado”. Sin embargo, en este contexto, se refiere a la limpieza y filtro de los datos.
- Web scraping es una técnica utilizada para extraer información de sitios web, es decir, es el proceso de recopilar información de forma automática de la Web. Conocer la estructura de una página web (DOM) es el primer paso para extraer y usar los datos.

# Web Scraping en nuestra Aplicación

- Para el desarrollo de nuestra aplicación, tuvimos que scrapear la página de SIU GUARANÍ

<http://www.dit.ing.unp.edu.ar/v2070/www/>

- El primer paso consistió en descargar todas las páginas web de dicho sitio, porque las inscripciones a mesas de examen duran pocos días, y nosotros las necesitamos para poder scrapear datos. Para esto, utilizamos Wget.



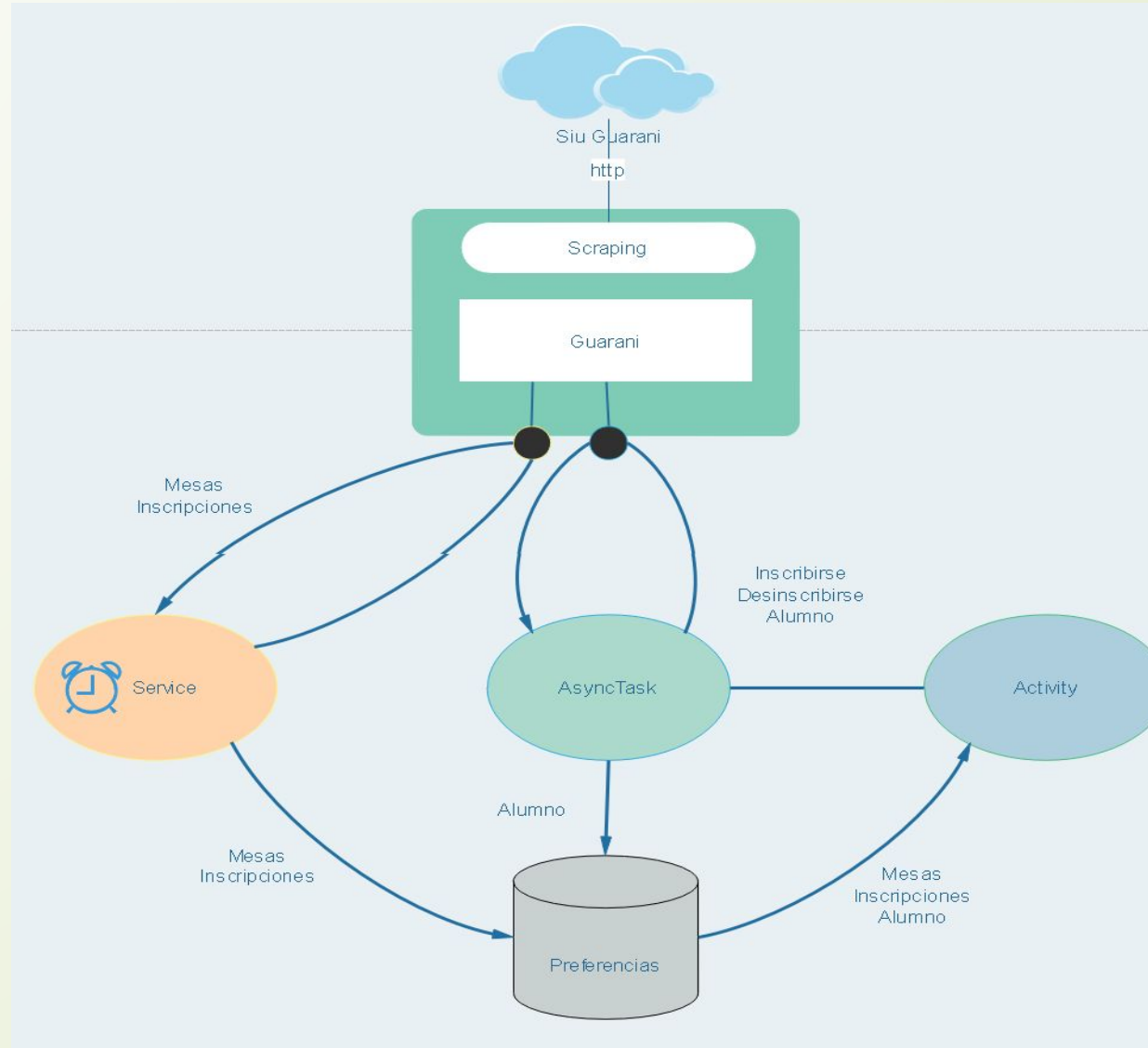
# **SIU GUARANÍ MÓVIL POR DENTRO**

# La Aplicación por dentro

→ Podemos abordar la aplicación de la siguiente manera:

- ◆ Funcionamiento general de la aplicación.
- ◆ Arquitectura de la Aplicación.
- ◆ Modelo de dominio GUARANÍ.
- ◆ Estructura de carpetas.
- ◆ Tecnologías y Herramientas utilizadas.
- ◆ Web Scraping desde la aplicación.
- ◆ El Servicio de la aplicación.
- ◆ La Alarma de la aplicación.
- ◆ Los Broadcast Receivers de la aplicación.
- ◆ Los AsyncTask de la aplicación.
- ◆ Los Dialogs de la aplicación.
- ◆ Las Preferencias de la aplicación.
- ◆ El uso de Fragments en la aplicación.

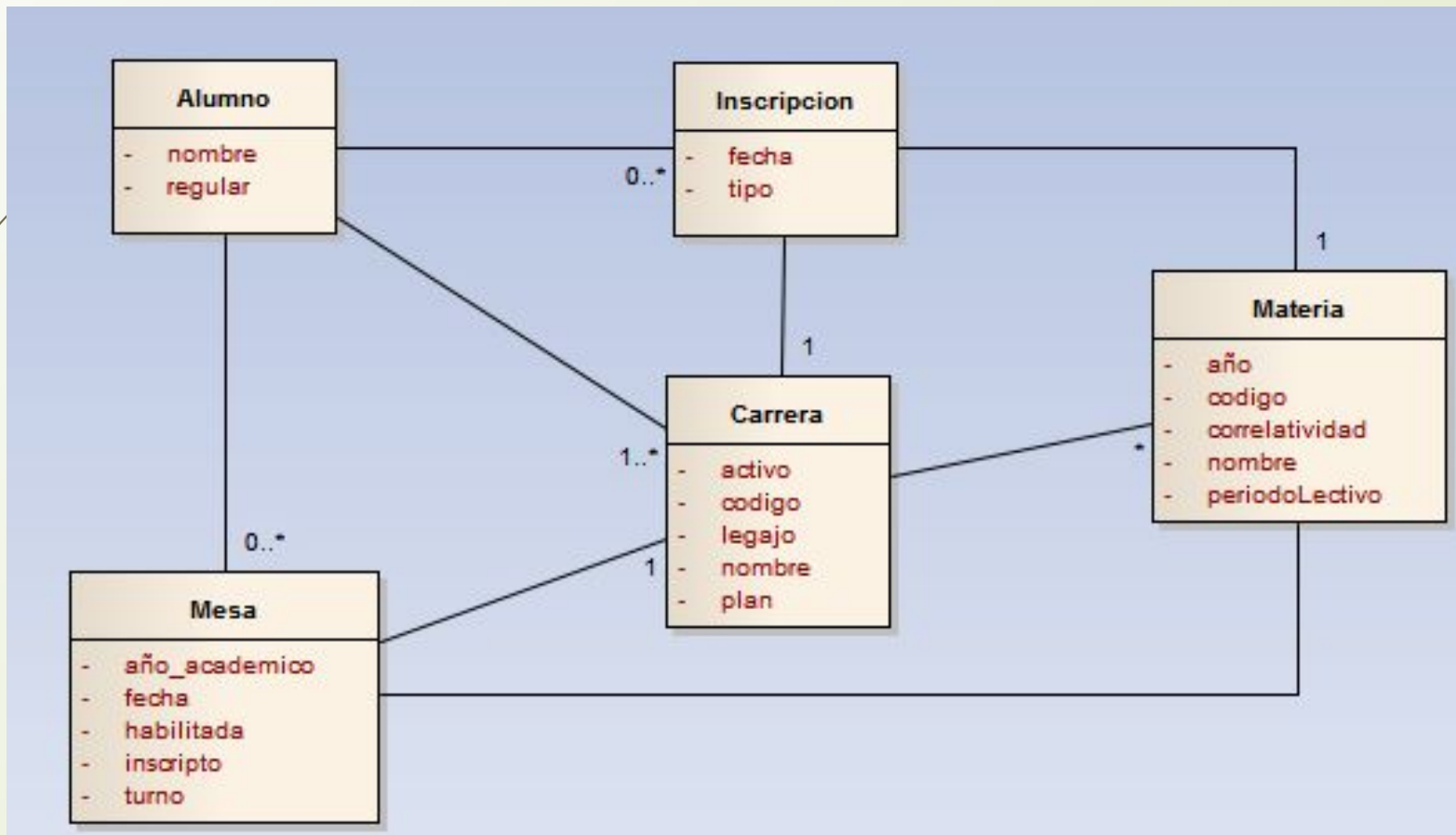
# Arquitectura de la Aplicación



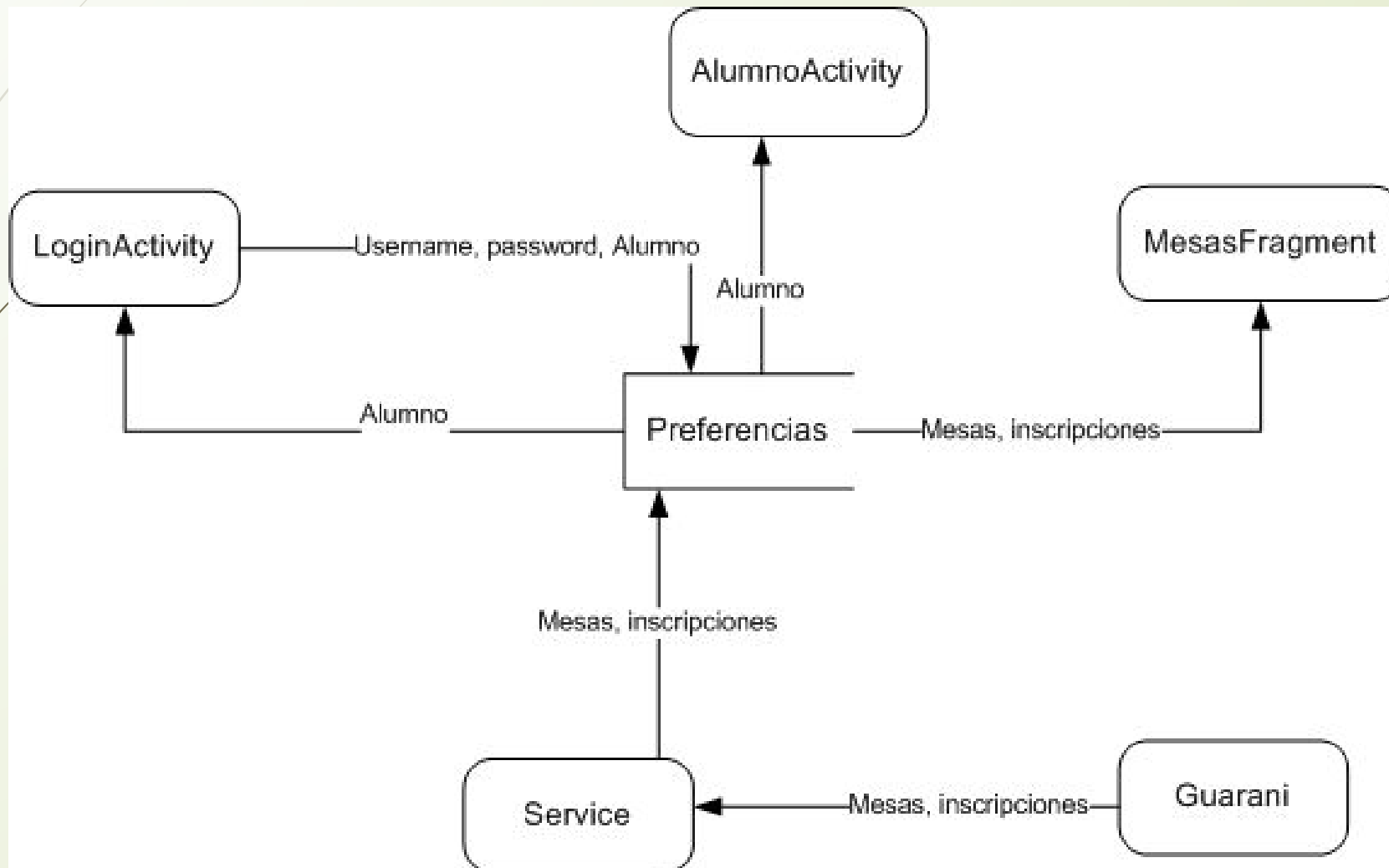


# Modelo de dominio GUARANÍ

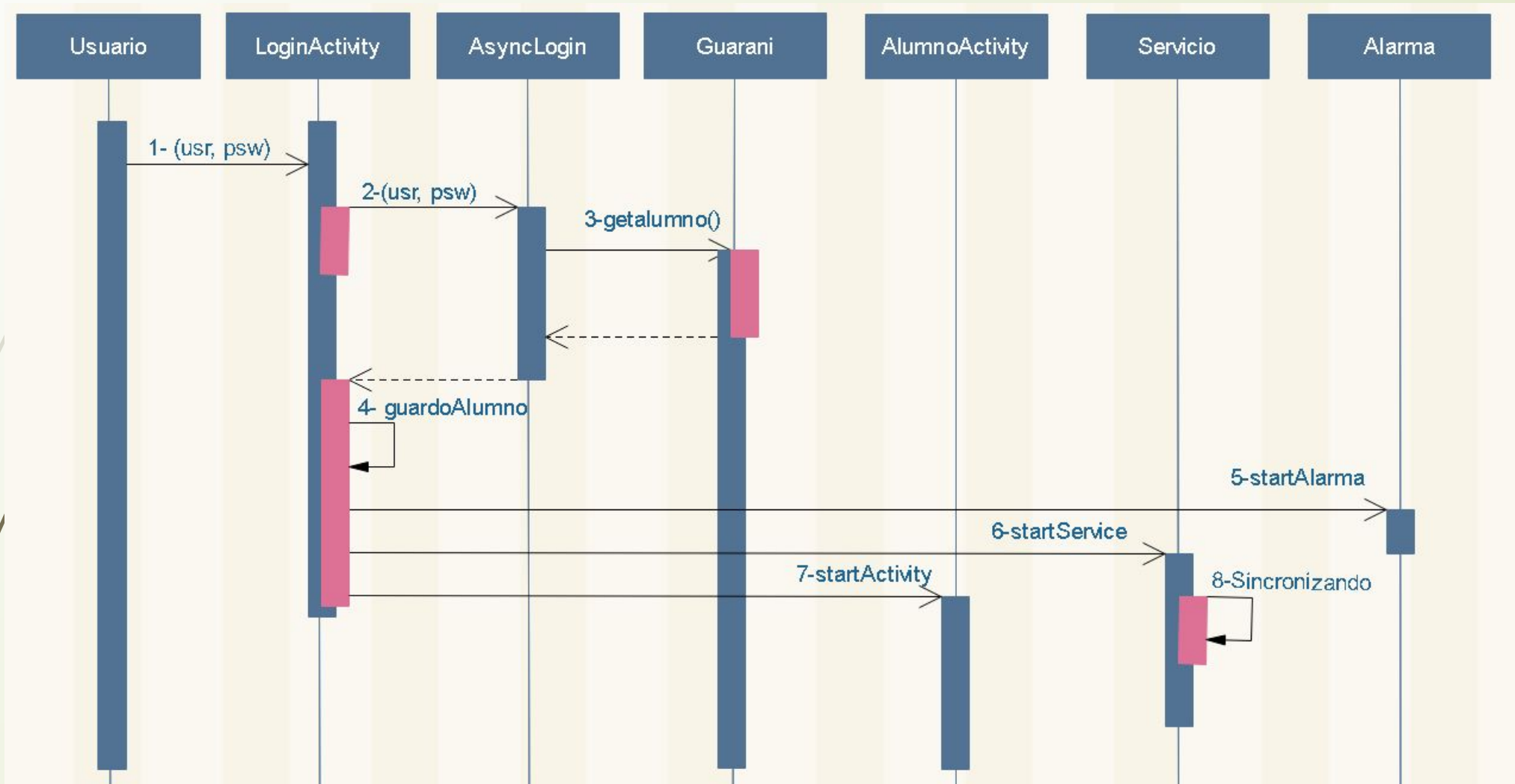
→ Para el desarrollo de la aplicación nos basamos en el siguiente modelo que hemos realizado:



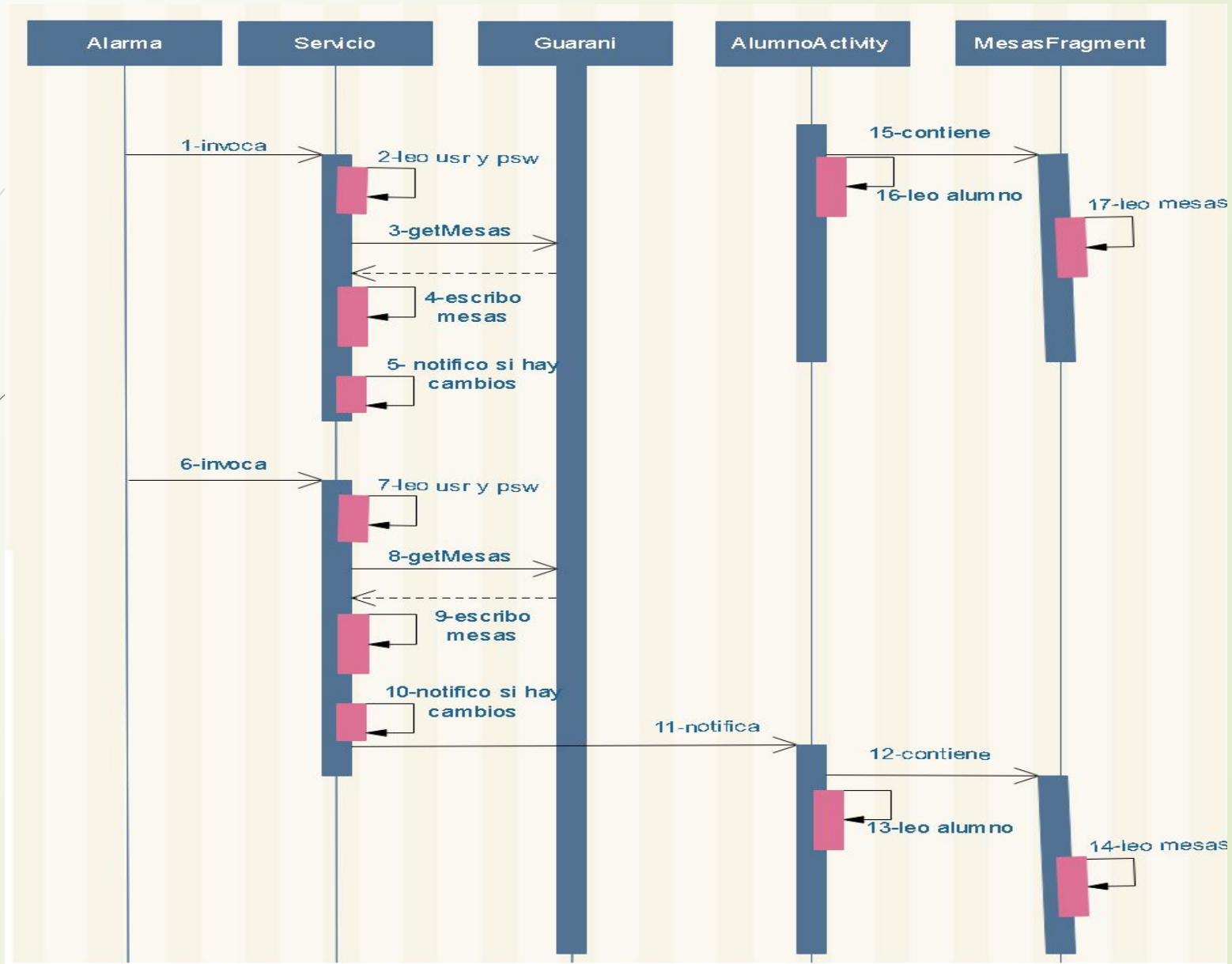
# Flujo de Datos



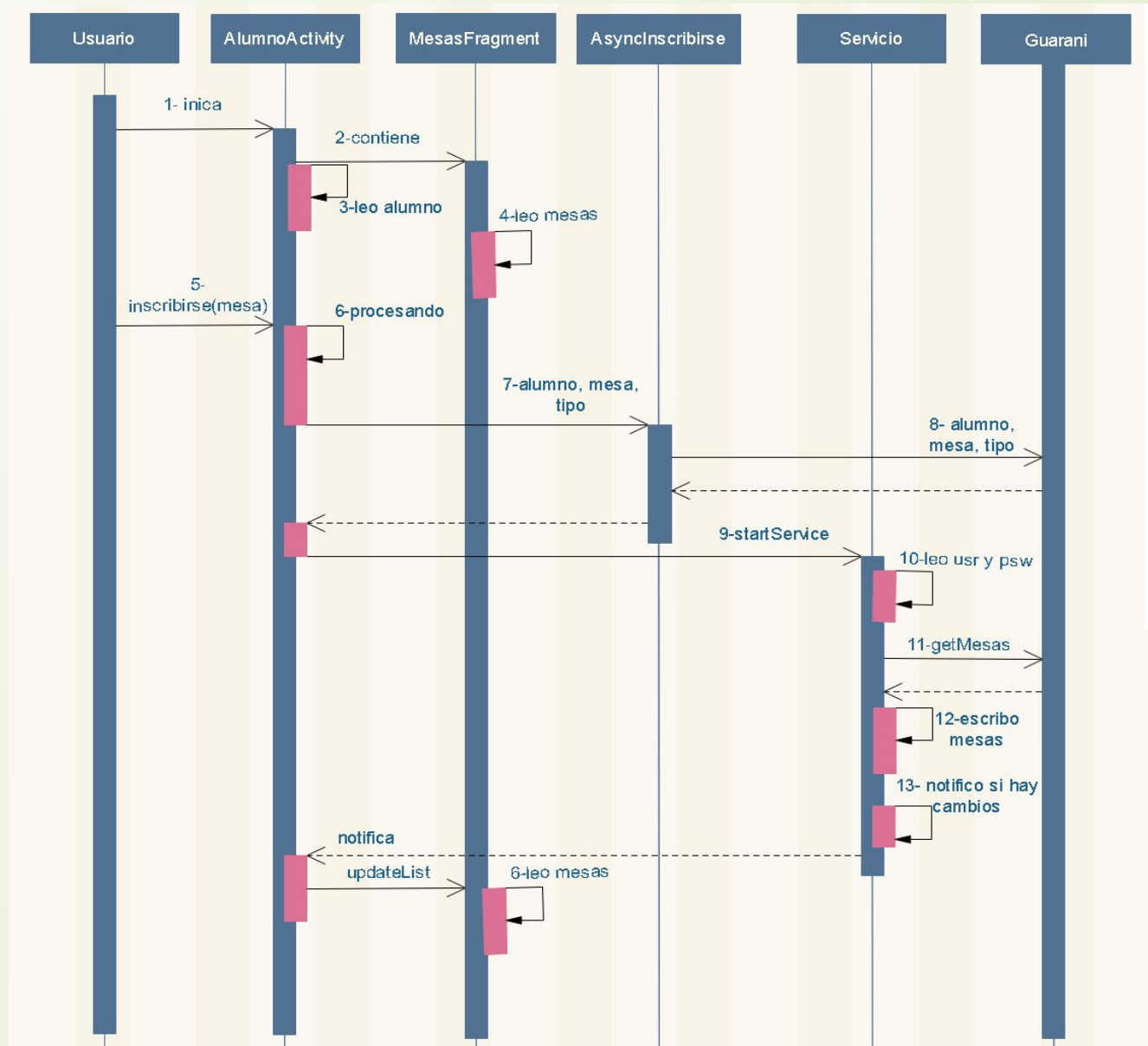
# Diagrama de Secuencia: Login



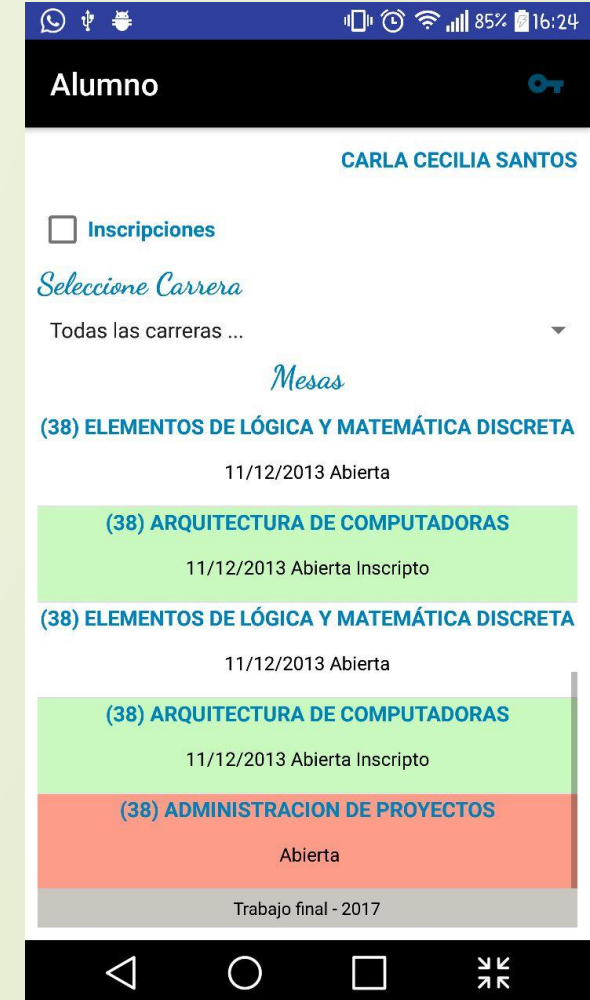
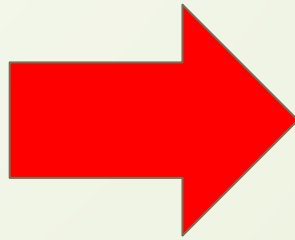
# Diagrama de Secuencia: Servicio y Notificaciones



# Diagrama de Secuencia: Inscripción



# Funcionamiento general de la aplicación (1)

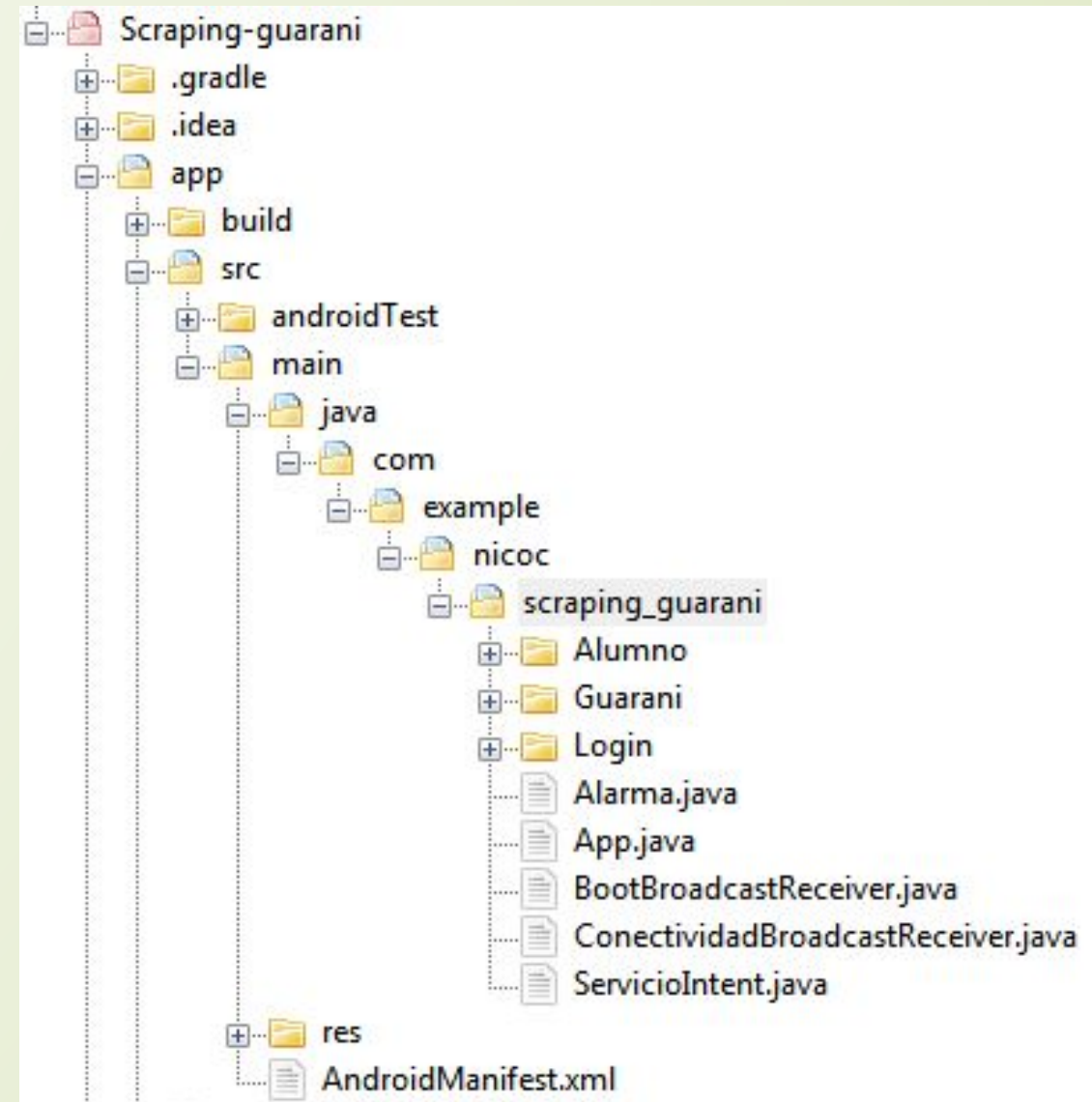


# Estructura de carpetas (1)

La estructura de carpetas de nuestra aplicación “**Scraping-guarani**” está en la imagen.

La carpeta **Java** contiene los archivos fuentes de la aplicación. Podemos ver que hay 3 carpetas principales: **Alumno**, **Guarani** y **Login**.

1) **Alumno**: contiene funcionalidad relacionada con el fragment de mesas, AsyncTask para inscribirse/desinscribirse a una mesa de examen y desloguearse de GUARANÍ, y clases para la gestión de la Interfaz de Alumno y del alumno que se ha logueado.



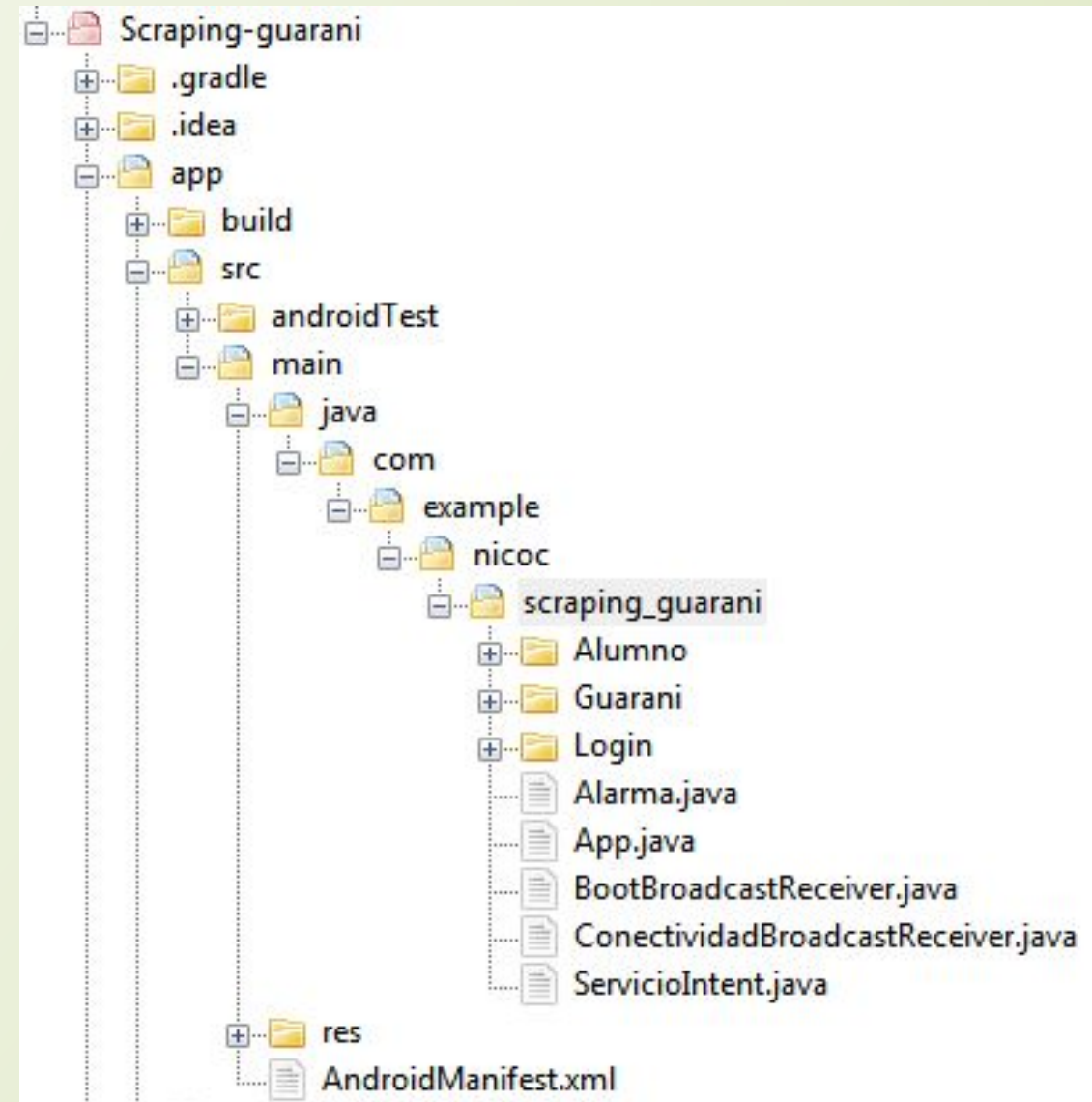


## Estructura de carpetas (2)

2) **Guaraní:** contiene las clases que representan el **modelo de dominio** GUARANÍ (Alumno, Carrera, Inscripción, Mesa, Materia, Auth) y la clase **Guarani.java** que encapsula toda la funcionalidad de scraping hacia el sistema SIU GUARANÍ.

3) **Login:** contiene la clase **LoginActivity.java** que es la que se encarga de mostrar la pantalla de login al usuario.

Contiene un **AsyncTask** para iniciar sesión llamado **AsyncLogin.java**.



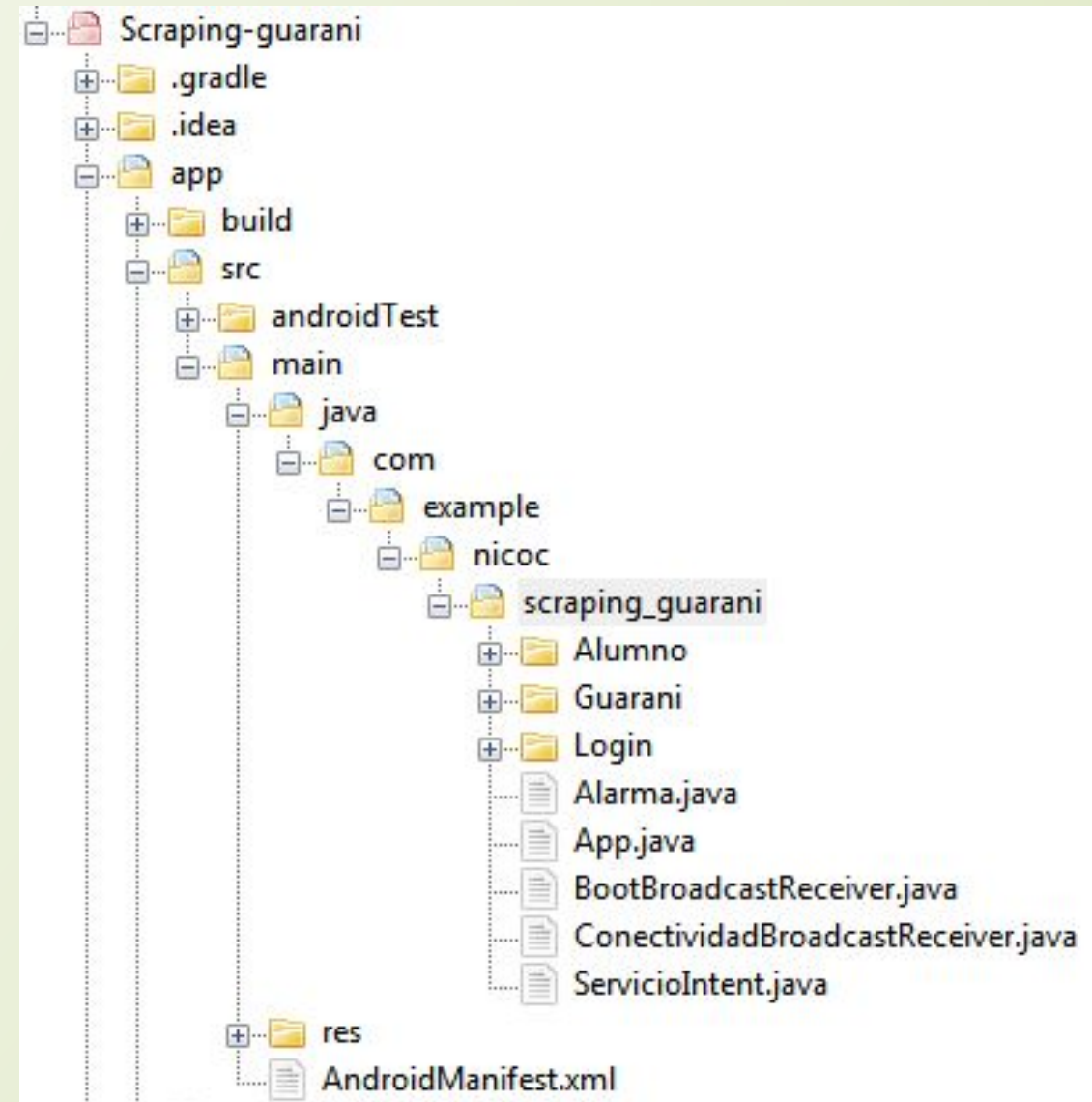


## Estructura de carpetas (3)

El resto de las clases que están sin una carpeta particular, es decir:

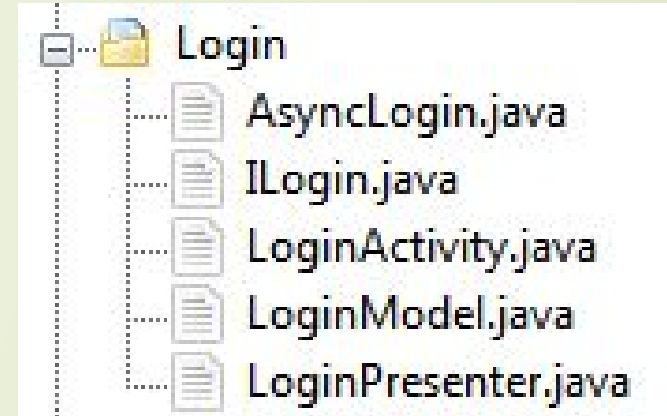
- Alarma.java
- App.java
- BootBroadcastReceiver.java
- ConectividadBroadcastReceiver.java
- ServicioIntent.java

Encapsulan toda la funcionalidad del servicio que se encarga de consultar las mesas/inscripciones a GUARANÍ.



## Estructura de carpetas (4)

Se optó por seguir el patrón arquitectónico MVP (Modelo-Vista-Presentador). Por ejemplo si desplegamos la carpeta “Login” veremos lo siguiente:



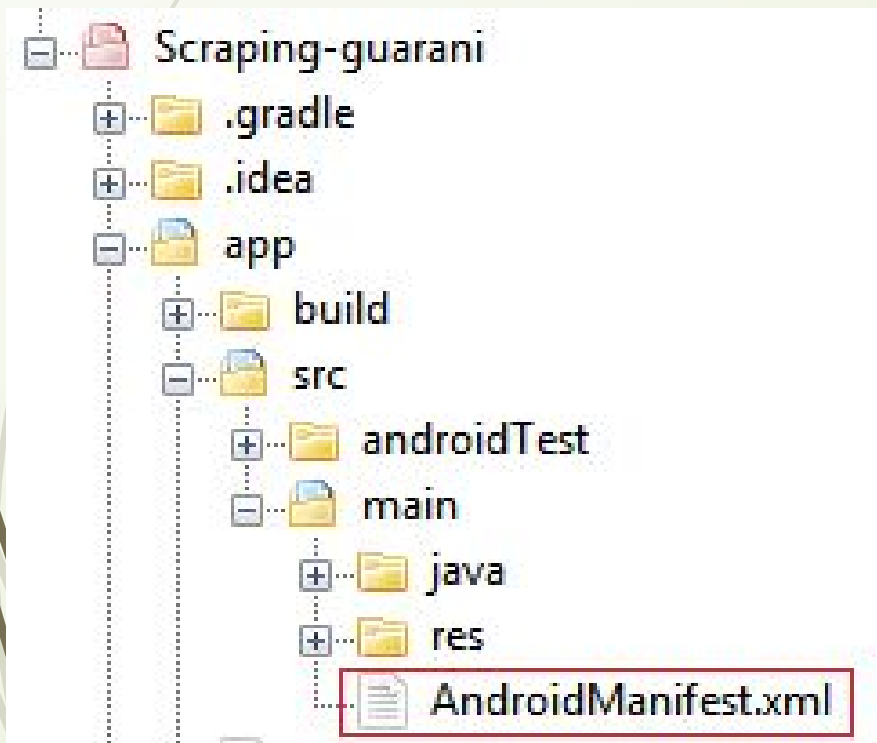
**Modelo:** está representado por la clase LoginModel.java. Maneja todas las operaciones con datos necesarios para el logueo del usuario.

**Vista:** está representada por la clase LoginActivity.java. La vista es una interfaz pasiva que exhibe datos (el modelo) y órdenes de usuario de las rutas (eventos) al presentador para actuar sobre los datos.

**Presentador:** está representado por la clase LoginPresenter.java. Se encarga de recibir las órdenes del usuario y de solicitar los datos al modelo para comunicárselos a la vista.

## Estructura de carpetas (5)

El archivo Manifest Contiene el manifiesto de la aplicación principal, esto es, la declaración de cada componente de aplicación (activity, services, etc.), los permisos, etc.



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.nicoc.scraping_guarani">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />

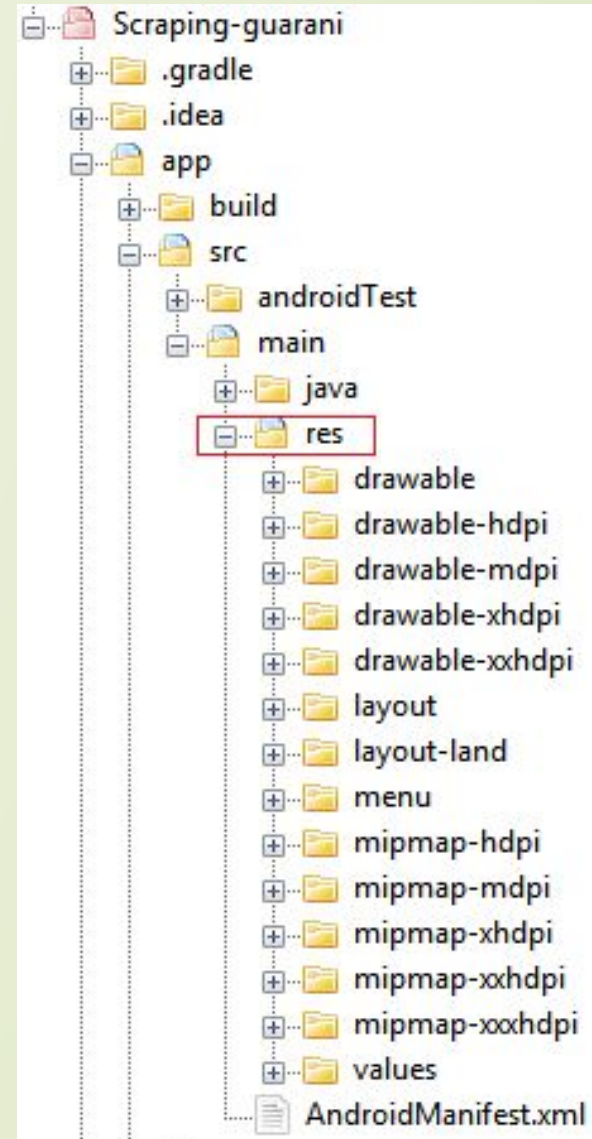
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".Login.LoginActivity"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".Alumno.AlumnoActivity"
            android:screenOrientation="sensor" />
        <service
            android:name=".ServicioIntent"
            android:exported="false" />
        <receiver
            android:name=".ConectividadBroadcastReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
                <action android:name="android.net.wifi.WIFI_STATE_CHANGED" />
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

# Estructura de carpetas (6)

La **carpeta “res”** es la que contiene todos los archivos de recursos (resources) utilizados por la aplicación: imágenes, videos, cadenas de texto, etc.

**res/drawable:** Contiene los elementos gráficos para las distintas resoluciones y densidades de pantallas. Dentro de esta carpeta nosotros hemos puesto el logo de la aplicación llamado “logo.png”.

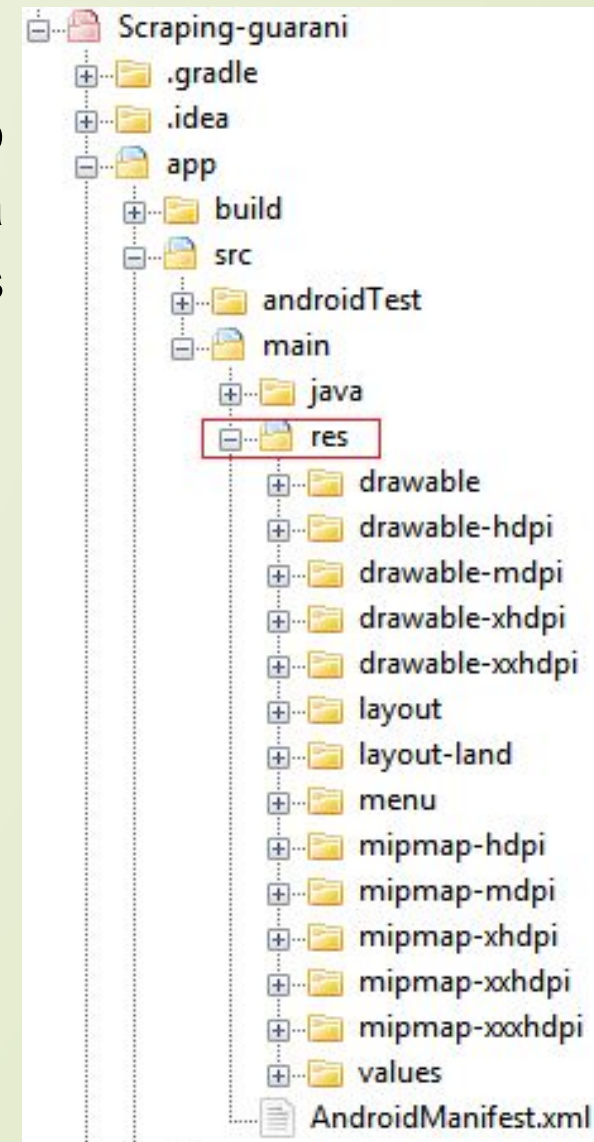
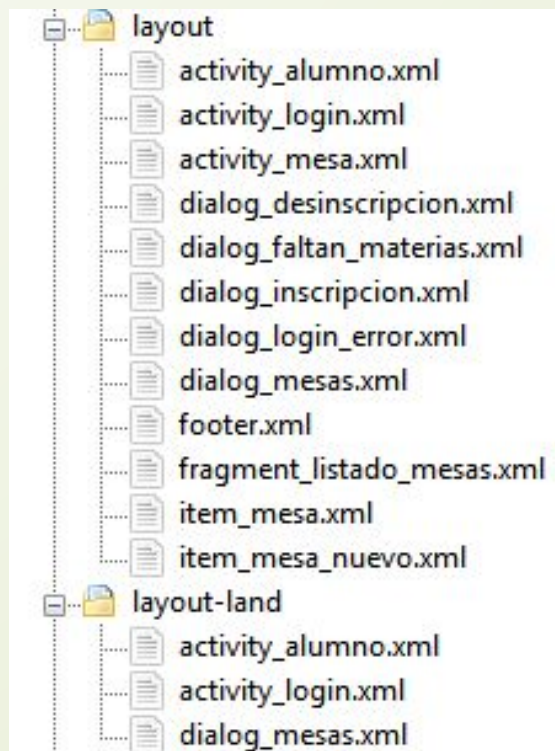
**res/mipmap:** Contiene los iconos para las distintas densidades de pantallas. Aquí incorporamos el ícono de la aplicación llamado “ic\_launcher.png”.





# Estructura de carpetas (7)

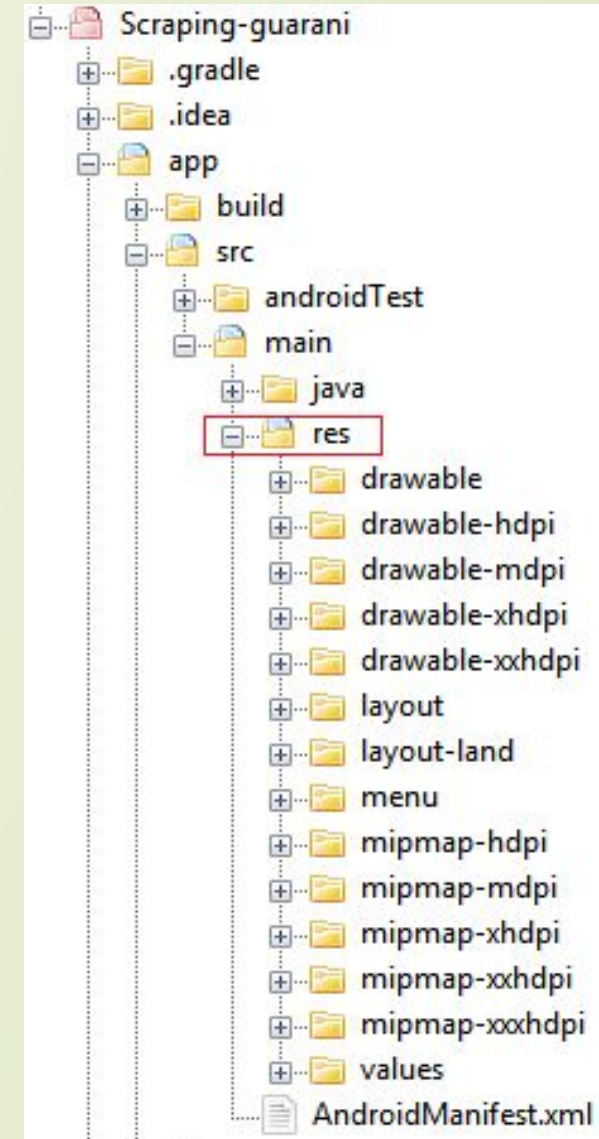
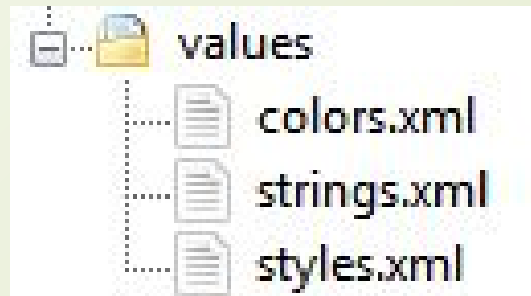
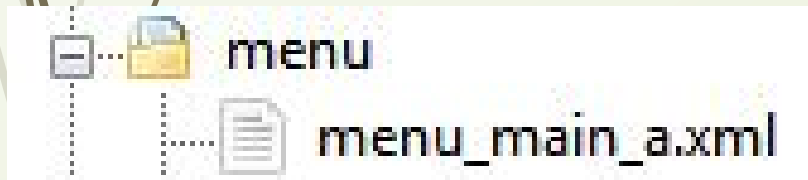
**res/layout:** Contiene los ficheros de definición XML de las diferentes pantallas de la interfaz gráfica. La carpeta layout incorpora los XML cuando estamos en modo Portrait (más alto que ancho), mientras que la carpeta layout-land contiene los XML cuando estamos en modo Landscape (más ancho que largo).



# Estructura de carpetas (8)

**res/menu:** Contiene la definición XML de los menús de la aplicación. Nosotros hemos definido un menú llamado menu\_main\_a.xml que lo usamos cuando estamos en la activity AlumnoActivity.java y contiene un item que nos permite cerrar sesión.

**res/values:** Contiene archivos XML de recursos de la aplicación, como por ejemplo cadenas de texto (strings.xml), estilos (styles.xml), colores (colors.xml), arrays de valores (arrays.xml), etc.



# Tecnologías y Herramientas utilizadas (1)

- Para desarrollar aplicaciones Android necesitamos:
  - ◆ Sun's Java Development Kit (JDK, v7)
  - ◆ Android Software Developer's Kit (SDK)
  - ◆ Android Studio
- **Android Studio:** Se trata del entorno de desarrollo integrado (IDE) oficial para la plataforma Android. Está basado en el software IntelliJ IDEA de JetBrains y ha sido publicado de forma gratuita a través de la Licencia Apache 2.0. Está disponible para las plataformas Microsoft Windows, macOS y GNU/Linux. Ha sido diseñado específicamente para el desarrollo de Android. Utiliza el lenguaje de programación Java.

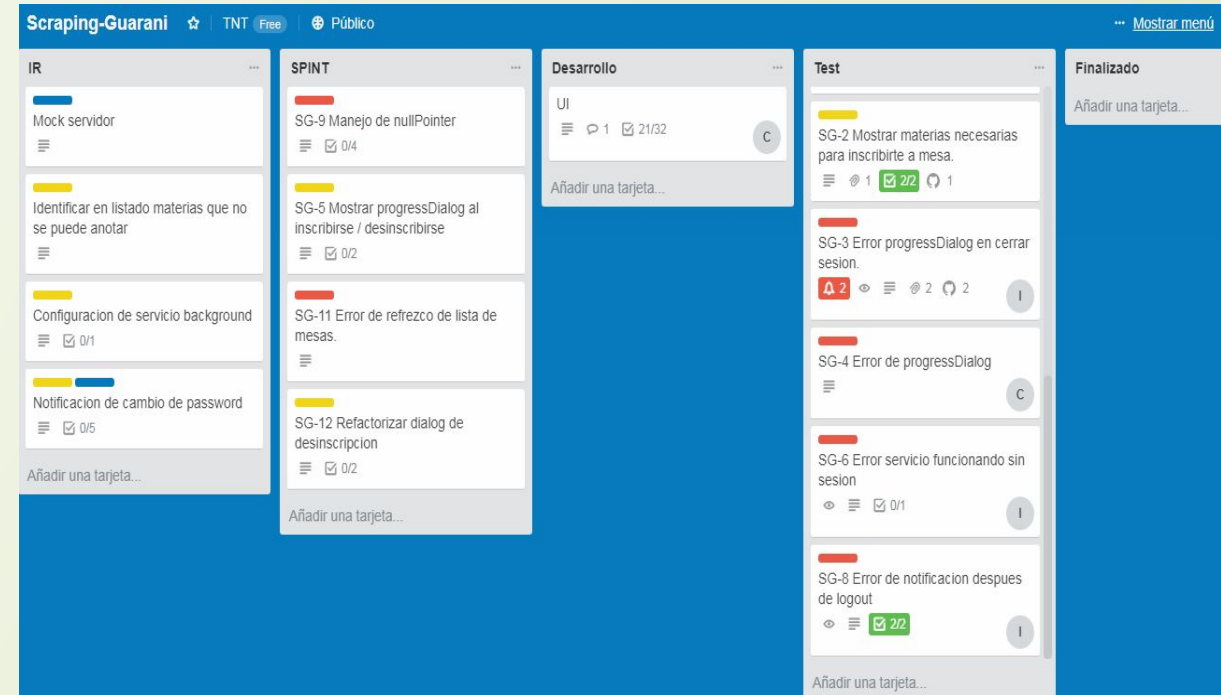
## Tecnologías y Herramientas utilizadas (2)

- **Wget:** Se trata de una herramienta libre que permite la descarga de contenidos desde servidores web de una forma simple. Su nombre deriva de World Wide Web (w), y de «obtener» (en inglés get), esto quiere decir: obtener desde la WWW. Lo utilizamos para descargarnos las páginas web necesarias de SIU GUARANÍ.
- **JSOUP:** es una librería Java que facilita mucho la labor de scrapeo. Provee una API para extraer y manipular datos. La manera de seleccionar los elementos del DOM es similar a jQuery. Permite:
  - ◆ Scrpear y parsear HTML desde una URL, archivo o String.
  - ◆ Buscar y extraer datos, usando DOM o selectores CSS.
  - ◆ Manipular los elementos, atributos y texto HTML.
  - ◆ Entre otros.



## Tecnologías y Herramientas utilizadas (3)

**Trello:** Para especificar la funcionalidad de la app y ver qué es lo que está haciendo cada uno en un determinado momento utilizamos la herramienta Trello es un software de administración de proyectos. Emplea el sistema kanban, para el registro de actividades a través de tarjetas virtuales. Permite organizar tareas, agregar listas, adjuntar archivos, etiquetar eventos, agregar comentarios y compartir tableros. Trello es un tablero virtual en el que se pueden colgar ideas, tareas, imágenes o enlaces.



El tablero de Trello se puede ver en:  
<https://trello.com/b/b76gl2lC/scraping-guarani>

## Tecnologías y Herramientas utilizadas (4)

**ButterKnife:** es una librería desarrollada y mantenida por Jake Wharton (Square Inc.) que nos facilita la tarea de relacionar los elementos de las vistas con el código en las aplicaciones Android.

Nos evita tener que utilizar findViewById y simplifica el código. No solo podemos utilizar ButterKnife para inyectar vistas, también ofrece otro tipo de anotaciones para manejar eventos como onClick(), onLongClick(), etc.

```
1 public class MainActivity extends Activity{
2
3     @Bind(R.id.my_textview) TextView myTextView;
4
5     @Override
6     protected void onCreate(Bundle savedInstanceState) {
7         super.onCreate(savedInstanceState);
8         setContentView(R.layout.main_activity);
9         ButterKnife.bind(this);
10
11         myTextView.setText("Texto de ejemplo");
12     }
13
14     @OnClick(R.id.my_textview)
15     public void submit() {
16         Toast.makeText(this, "Pulso sobre el TextView", Toast.LENGTH_LONG).show();
17     }
18 }
```

Para instalar Butter Knife solo tendremos que añadir la siguiente dependencia al archivo build.gradle del proyecto:

*compile 'com.jakewharton:butterknife:7.0.1'*

## Tecnologías y Herramientas utilizadas (2)

**GSON:** es conocido como Google Gson, es una biblioteca de código abierto para Java que permite la serialización y deserialización entre objetos Java y su representación en notación JSON. Nosotros usamos esta librería para manejar los objetos (alumnos, mesas, inscripciones, etc.) y guardarlos en las preferencias como JSON.

JSON es el acrónimo de JavaScript Object Notation, es un formato de texto ligero para el intercambio de datos. JSON es un formato de texto para la serialización de datos estructurados.

Para utilizar GSON simplemente deberemos añadir una línea en el archivo .gradle para incluirlo en nuestro proyecto:

compile 'com.google.code.gson:gson:2.3.1'

```
String mesas_json_guardadas = preferences.getString("mesas", "");  
Type collectionType = new TypeToken<ArrayList<Mesa>>(){}.getType();  
ArrayList<Mesa> mesas_guardadas = new Gson().fromJson(mesas_json_guardadas,  
collectionType);
```

## Web Scraping desde la aplicación (1)

- Como hemos mencionado anteriormente, el Scraping es una técnica que se usa para obtener el contenido (datos) que hay en páginas web a través de su código HTML.
- Toda la funcionalidad de scraping hacia la página de SIU GUARANÍ está encapsulada dentro de la clase java Guarani. También encapsula comportamiento para loguearse/desloguearse e inscribirse/desinscribirse a una mesa de examen en SIU GUARANÍ a través de peticiones HTTP, por lo que deben ejecutarse en tareas en segundo plano (ya que son tareas pesadas que bloquearán el UI THREAD).
- La clase Guarani es un singleton (es decir, la idea es que haya una única instancia Guarani y proporcionar un punto de acceso global a ella), con métodos synchronized para evitar inconsistencias entre hilos.



## Web Scraping desde la aplicación (2)

- La clase Guarani presenta una API que nos permite:
  - ◆ Realiza una inscripción a una mesa, para lo cual requiere una instancia del Alumno que se inscribirá a la mesa, instancia de la Mesa a la que se va a inscribir, el tipo de inscripción (regular o libre).
  - ◆ Devolver un listado de las inscripciones de un alumno. Representan inscripciones que pueden ser canceladas.
  - ◆ Desinscribirse de una mesa de examen, identificándola por código de carrera y materia.
  - ◆ Obtener todas las mesas de examen habilitadas para inscripción.
  - Loguearse/Desloguearse. Si existe una sesión activa, cerramos sesión contra el servidor guarani.
  - ◆ Obtener los datos del alumno, nombre, carreras y respectivas materias.

# El Servicio de la aplicación (1)

- Problema: scrapear datos es una tarea pesada. Dialog ANR si se ejecuta en UI THREAD.
- Solución: Uso de Intent Service (IS) en ServiceIntent.java.
- Servicio comparte proceso de ejecución con la Aplicación, IS no.
- IS diseñado para tareas pesadas: subir archivos, geofencing, scraping.
- Comentario: prueba de ejecución Service vs IS.
- Cuando se instancia un IS se ejecuta `onHandleIntent(Intent intent)`.
- Tareas que realiza nuestro IS:
  - ◆ Verifica si hay conexión (WiFi, 3G, 4G). Si no hay, termina.
  - ◆ Mientras haya conexión y el usuario esté logueado, se realiza scraping a SIU GUARANÍ para obtener las mesas de examen y las inscripciones.
  - ◆ Comunicar al usuario cuando haya nuevas mesas e inscripciones.

## El Servicio de la aplicación (2)

- Para consultar conectividad se necesitan 2 permisos:
  - ◆ `<uses-permission android:name="android.permission.INTERNET" />`
  - ◆ `<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />`
- **ConnectivityManager:** es la clase que se encarga de responder consultas sobre el estado de la conectividad de red. Además notifica a las aplicaciones cuando la conectividad de red cambia.
- **Comunicación:** Para comunicarle al alumno la existencia de mesas/inscripciones, utilizamos notifications y broadcast.
- **Notification:** se muestran en el área de notificaciones del dispositivo. Poseen mínimamente, un título, un mje y un ícono. Nosotros le agregamos la capacidad de vibrar el dispositivo y agregarle un PendingIntent.
- **Notification Manager:** gestiona las notificaciones que se deben mostrar en la barra de notificaciones del dispositivo.



## El Servicio de la aplicación (3)

- Cuando comunicamos a través de broadcasts, lo que hacemos es crear una instancia de `LocalBroadcastManager`, al cual le pasamos un `Intent` con un nombre, y se encargará de difundir a toda la aplicación el intent que le hemos agregado. Si hay una instancia de `AlumnoActivity` escuchando, capturará dicho `Intent`, y realizará las acciones pertinentes.
- **LocalBroadcastManager:** es la clase que ayuda a registrar o enviar broadcasts de intents a objetos locales dentro de nuestro proceso. Ventajas sobre enviar un broadcast global a partir del método `sendBroadcast(Intent)`:
  - ◆ Los datos que estamos broadcasteando no sobrepasarán la aplicación (privacidad de datos).
  - ◆ Es imposible para otras aplicaciones enviar estos broadcast hacia mi aplicación (seguridad).
  - ◆ Es más eficiente que enviar un broadcast global a través del sistema.

# El Servicio de la aplicación (4)



## La Alarma de la aplicación (1)

- El Intent Service nos resuelve el problema de las tareas pesadas. Debemos ejecutar ese servicio con una cierta frecuencia para poder scrapear datos de las mesas e inscripciones. Esto lo podemos realizar utilizando la clase **AlarmManager**.
- Esta clase proporciona acceso a los **servicios de alarma del sistema**. Esto permite programar aplicaciones que se ejecuten en algún momento en el futuro. Cuando se activa una alarma, el Intent que se ha registrado en ella se transmite por el sistema, iniciando automáticamente la aplicación de destino si no se está ejecutando. Las alarmas registradas se conservan mientras el dispositivo está dormido, pero se borrará si se apaga y se reinicia.

## La Alarma de la aplicación (2)

- Si bien esta clase tiene varios métodos, nosotros hemos utilizado uno llamado: **setRepeating(...)**
- Éste método permite programar una alarma repetitiva. Esta alarma continuará repitiéndose hasta que se elimine explícitamente con el método `cancelar(AlarmManager.OnAlarmListener)`. Si el tiempo de disparo establecido está en el pasado, la alarma se activará de inmediato
- Parámetros:
  - ◆ Tipo de Alarma
  - ◆ Tiempo en milisegundos a partir del cual la alarma se activará. Ej: a las 15:00 PM.
  - ◆ Tiempo en milisegundos que indicará la frecuencia de repetición. Ej: cada 1 hs.
  - ◆ `PendingIntent`: Acción a realizar cuando la alarma se dispara.
- Nosotros configuramos una alarma a las 12:00 PM, y se repetirá cada 12 hs, ejecutando el servicio definido en `ServiceIntent`. Esto se encuentra en `Alarma.java` la cual extiende de `Thread`.

## Los Broadcast Receivers de la aplicación (1)

- Nuestra aplicación está compuesta por dos Broadcast Receivers: `BootBroadcastReceiver.java` y `ConectividadBroadcastReceiver.java`.
- **BootBroadcastReceiver**: se encarga de reiniciar la alarma cuando el dispositivo se apaga o se reinicia, ya que ante estos dos casos la alarma muere, por lo que el servicio no se ejecutará nunca más. Para poder hacer esto, debemos definir el siguiente permiso dentro del `AndroidManifest.xml` de la aplicación:  
`<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />`

## Los Broadcast Receivers de la aplicación (2)

- **ConectividadBroadcastReceiver**: se encarga simplemente de monitorear constantemente la conectividad de la red (WiFi, 3G, 4G). Pero sólo lo va a hacer cuando nosotros queremos que lo haga. La necesidad de esto surge en la clase ServicioIntent. Éste, antes de scrapear, pregunta si hay conectividad. En caso de que no haya, utiliza las preferencias y setea el valor de la preferencia “aviso” a “true” y luego muere. El broadcast, detecta este cambio del valor de la preferencia de falso a verdadero y se activa. Cuando detecta conectividad, lo que hace es reiniciar de nuevo el servicio.



## Los AsyncTask de la aplicación (1)

- Todos los componentes de una aplicación Android, tanto las actividades como los broadcast receivers se ejecutan **en el mismo hilo de ejecución**, llamado hilo principal, main thread o GUI thread. Es el hilo donde se ejecutan todas las operaciones que gestionan la interfaz de usuario de la aplicación.
- Cualquier operación larga o costosa que realicemos en este hilo va a bloquear la ejecución del resto de componentes de la aplicación así como también la interfaz, produciendo al usuario un efecto de lentitud, bloqueo, o mal funcionamiento.
- **Android monitorea las operaciones** realizadas en el hilo principal y detecta aquellas que superen los **5 segundos**, en este caso se muestra el famoso mensaje de “Application Not Responding” (ANR) y el usuario debe decidir entre forzar el cierre de la aplicación o esperar a que termine la ejecución de la misma. Es por eso que utilizamos procesos en segundo plano.



## Los AsyncTask de la aplicación (2)

- Existen algunas alternativas a la hora de ejecutar tareas en segundo plano en Android:
  - ◆ Crear nosotros mismos de forma explícita un **nuevo hilo** para ejecutar nuestra tarea.
  - ◆ Utilizar la clase auxiliar **AsyncTask** proporcionada por Android.
  - ◆ Utilizar la clase **IntentService** proporcionada por Android.
- Android provee la posibilidad de realizar tareas asincrónicas a través de la clase AsyncTask.

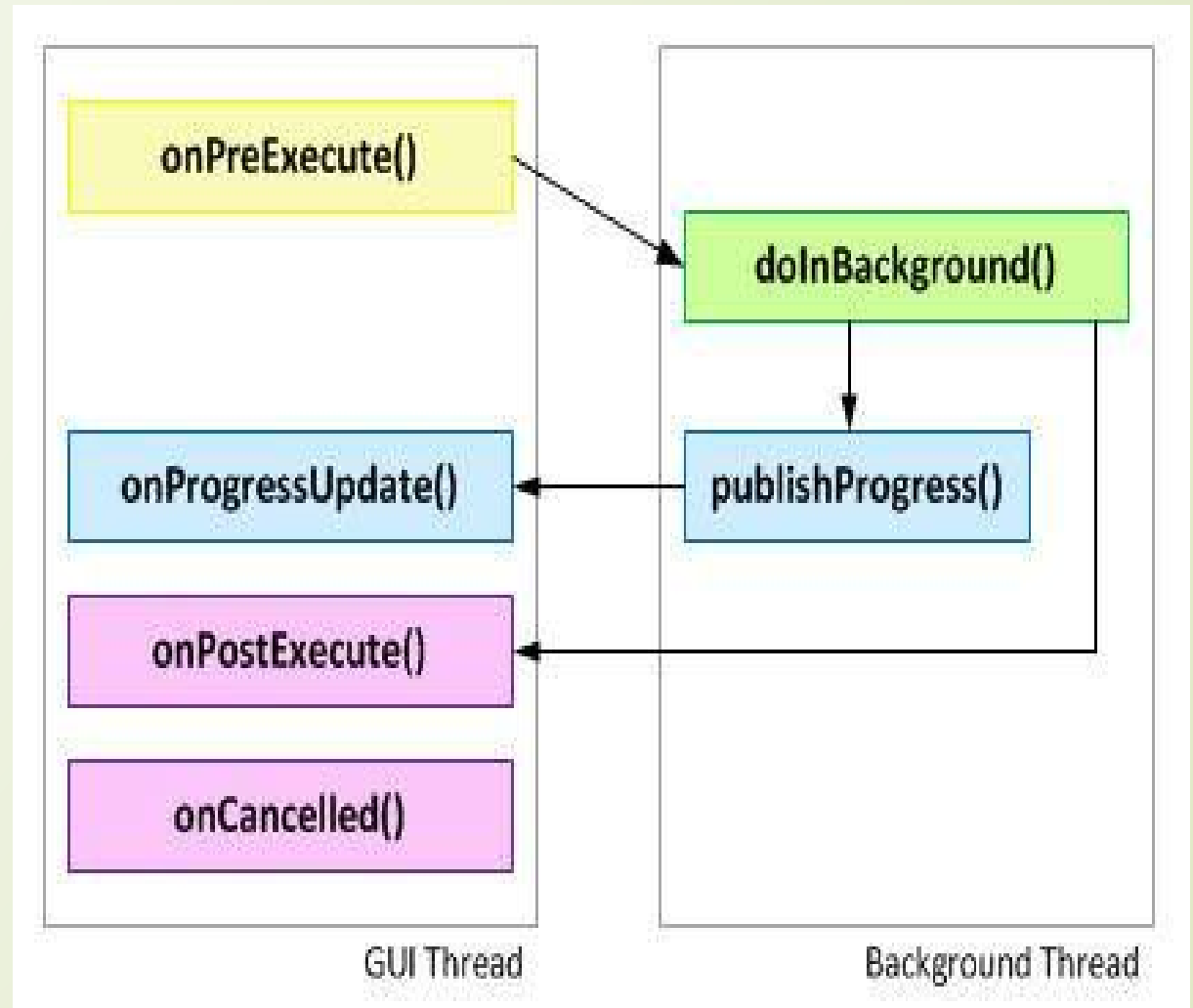
Nos permite realizar **operaciones en segundo plano y luego utilizar sus resultados en el hilo principal**. Esto evita que debamos controlar los hilos de ejecución por nosotros mismos.

## Los AsyncTask de la aplicación (3)

- La forma básica de utilizar la clase AsyncTask consiste en crear una nueva clase que extienda de ella y sobrescribir varios de sus métodos entre los que repartiremos la funcionalidad de nuestra tarea. Estos métodos son los siguientes:
- ◆ `onPreExecute()`. Se ejecutará antes del código principal de nuestra tarea. Se suele utilizar para preparar la ejecución de la tarea, inicializar la interfaz, etc.
  - ◆ `doInBackground()`. Contendrá el código principal de nuestra tarea.
  - ◆ `onProgressUpdate()`. Se ejecutará cada vez que llamemos al método `publishProgress()` desde el método `doInBackground()`.
  - ◆ `onPostExecute()`. Se ejecutará cuando finalice nuestra tarea, o dicho de otra forma, tras la finalización del método `doInBackground()`.
  - ◆ `onCancelled()`. Se ejecutará cuando se cancele la ejecución de la tarea antes de su finalización normal.

## Los AsyncTask de la aplicación (4)

- El método `doInBackground()` se ejecuta en un hilo secundario (por lo tanto NO podremos interactuar con la interfaz). Sin embargo todos los demás se ejecutan en el hilo principal, lo que quiere decir que dentro de ellos podremos hacer referencia directa a nuestros controles de usuario para actualizar la interfaz.
- Dentro de `doInBackground()` tendremos la posibilidad de llamar periódicamente al método `publishProgress()` para que automáticamente desde el método `onProgressUpdate()` se actualice la interfaz si es necesario.



## Los AsyncTask de la aplicación (5)

→ Al extender una nueva clase de AsyncTask tendremos que indicar tres parámetros de tipo:

```
private class MyTask extends AsyncTask<A, B, C> { ... }
```

- ◆ A: indica tipo de parámetros que recibirá la tarea para ser ejecutada. Dato de entrada en `doInBackground()`.
- ◆ B: indica el progreso, es decir, el tipo de datos con el que actualizaremos el progreso de la tarea. Dato de entrada en `onProgressUpdate()` y `publishProgress()`.
- ◆ C: indica el tipo de resultado. Será el tipo de retorno del método `doInBackground()` y el tipo del parámetro recibido en el método `onPostExecute()`.

→ No todas las tareas necesitan de las tres cosas. Utilizamos Void si algo no es necesario:

```
private class MyTask extends AsyncTask<Void, Void, Void> { ... }
```

## Los AsyncTask de la aplicación (6)

- Cabe mencionar que todo lo que podemos hacer con una AsyncTask lo podemos hacer con un Intent Service (IS).
- A diferencia de las AsyncTask, un IS **no proporciona métodos que se ejecuten en el hilo principal** de la aplicación y que podamos aprovechar para “comunicarnos” con nuestra interfaz durante la ejecución. Éste es el motivo principal de que los IntentService sean una opción menos utilizada a la hora de ejecutar tareas que requieran cierta vinculación con la interfaz de la aplicación.
- Sin embargo tampoco es imposible su uso en este tipo de escenarios ya que podremos utilizar por ejemplo **mensajes broadcast o notificaciones** para comunicar eventos al hilo principal (como ya hemos mencionado), como por ejemplo la necesidad de actualizar controles de la interfaz o simplemente para comunicar la finalización de la tarea ejecutada.

## Los AsyncTask de la aplicación (7)

→ En nuestra aplicación hemos utilizado varias tareas asincrónicas:

- ◆ **AsyncLogin.java:** recibe como parámetros el par usuario y contraseña (como Strings), inicia sesión en Guaraní y devuelve una instancia de Alumno con los datos ya seteados.
- ◆ **AsyncDesloguear.java:** no recibe nada como parámetro. Se encarga de cerrar sesión en Guaraní. No devuelve nada.
- ◆ **AsyncInscribirse.java:** recibe como parámetros una instancia de Object, al cual lo tratamos como un arreglo, que contiene el alumno, la mesa y el tipo de inscripción (regular/libre), y con esos datos hace una petición post a Guaraní para inscribirse a una mesa. Devuelve como resultado un booleano, el cual indica verdadero cuando se pudo inscribir correctamente a la mesa, o falso si ha ocurrido algún error.
- ◆ **AsyncDesinscribirse.java:** recibe como parámetro el par código de la carrera y código de la materia (como un String) y con ellos hace una petición a Guaraní para desinscribirse de una mesa de examen. Devuelve como resultado un booleano, que indica verdadero o falso según el éxito que tuvo al realizar la transacción.



## Los Preferencias en la aplicación (1)

- La aplicación que hemos desarrollado NO utiliza una Base de Datos.
- Utilizamos las Preferencias de Android como forma de persistir datos.
- Podremos acceder a las preferencias guardadas a través del manager de preferencias PreferenceManager. Las preferencias son manipuladas a través de un objeto SharedPreferences.

## Los Preferencias en la aplicación (2)

- Por ejemplo: si queremos guardar un String “Valor” cuya clave es “clave\_para\_el\_valor”, podemos codificar lo que está en la imagen:
- Ese par clave/valor va a persistir por más que nuestro dispositivo se reinicie o se apague. Para obtener este par, debemos hacer lo que está en la imagen:
- Con estas líneas de código hemos almacenado las mesas de examen, las inscripciones, los datos de la cuenta del alumno y otras variables que necesitábamos persistir para el buen funcionamiento de la aplicación.

```
SharedPreferences.Editor editor = sharedPref.edit();  
editor.putString("clave_para_el_valor", "Valor");  
editor.commit();
```

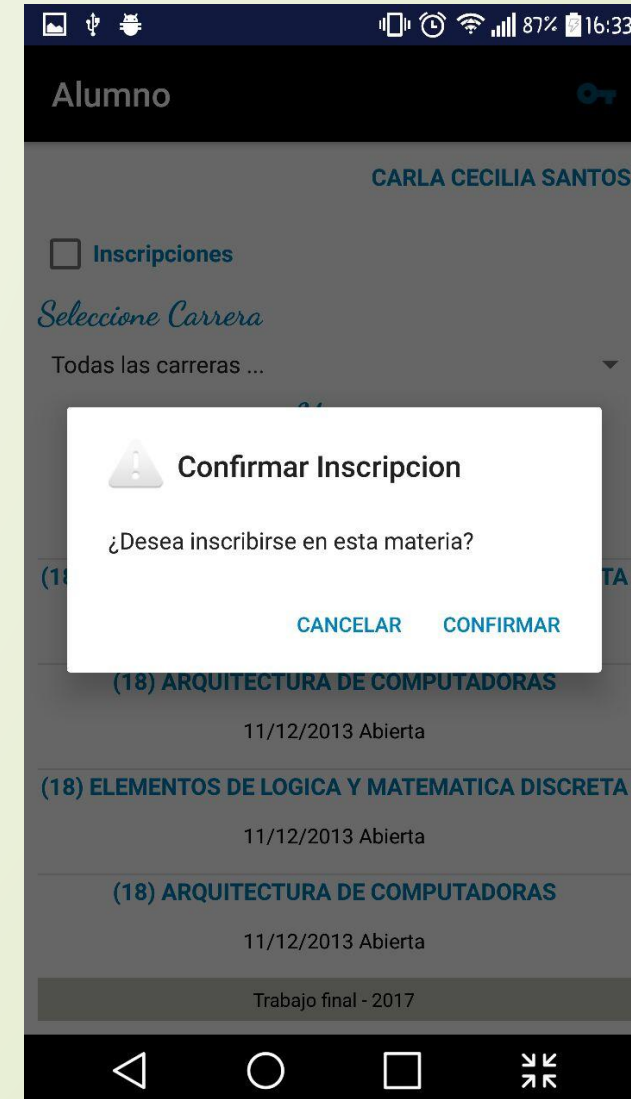
```
SharedPreferences preferences =  
context.getSharedPreferences("loginPrefs", MODE_PRIVATE);  
String valor =  
preferences.getString("clave_para_el_valor", "");
```

## Los Dialogs de la aplicación (1)

- La plataforma Android nos provee distintos modos de brindar información al usuario, las notificaciones (que ya las hemos mencionado) son uno de estos modos. Otra forma son los AlertDialogs. Estos muestran una ventana que toma el foco de la aplicación a la espera de que el usuario haga algo, normalmente confirmar, aceptar o cancelar una acción.
- En principio, los diálogos de Android los podremos utilizar con distintos fines, en general:
  - ◆ Mostrar un mensaje.
  - ◆ Pedir una confirmación rápida.
  - ◆ Solicitar al usuario una elección (simple o múltiple) entre varias alternativas.

## Los Dialogs de la aplicación (2)

- Para construir nuestros diálogos de alerta utilizamos la clase `AlertDialog.Builder` que es muy flexible y nos da la posibilidad de construir diálogos tan complejos y específicos como necesitemos.
- La clase `AlertDialog.Builder` ya nos provee todo lo necesario para mostrar un `AlertDialog` simple con unas pocas líneas de código, con un título, un mensaje y un botón, como por ejemplo:



## Los Dialogs de la aplicación (3)

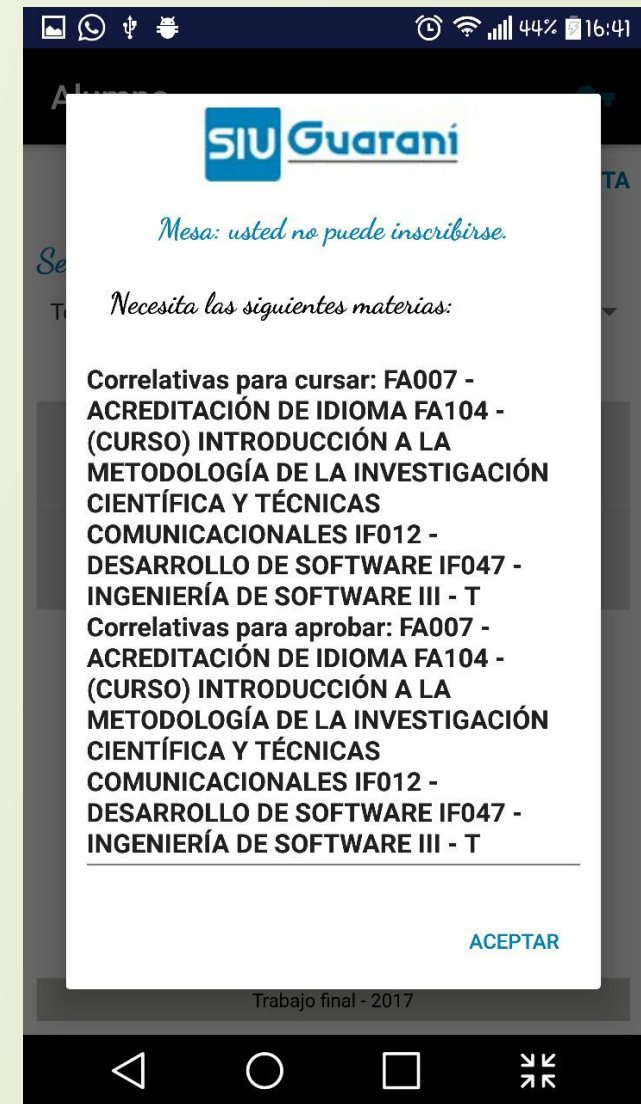
- Nosotros hemos definido Dialogs más complejos, primero definiendo parte de su estructura en un archivo layout.xml, para luego inflarlo a partir del mismo.
- Realizar un 'inflate' en Android es instanciar un xml para poder añadirlo a una jerarquía de vistas. El LayoutInflater es un servicio propio de Android que nos facilita el mecanismo de "inflado" de los XML. Básicamente lo que queremos decir con "Inflar" es agregar una vista (.xml) a nuestros dialogs en tiempo de ejecución.

## Los Dialogs de la aplicación (4)

- ◆ `dialog_desinscripcion.xml`: lo usamos para mostrar un AlertDialog que permite desinscribirte de una mesa de examen, mostrando los datos de inscripción.
- ◆ `dialog_faltan_materias.xml`: lo usamos para mostrar un AlertDialog que le indica al usuario que no puede inscribirse a una mesa de examen porque le faltan materias correlativas.
- ◆ `dialog_inscripcion.xml`: lo usamos para mostrar un AlertDialog que permite inscribirte de una mesa de examen. Muestra los datos de dicha mesa.



## Los Dialogs de la aplicación (5)



## Uso de Fragments en nuestra Aplicación (1)

- Se puede definir a un fragment como porciones modulares y cohesivas de interfaz con su propio ciclo de vida.
- Utilizamos fragments cuando queremos aprovechar mejor las dimensiones de la pantalla.
- Los fragments, siempre son hospedados por actividades. Cada fragment especificará su propio layout pudiendo este a su vez ser responsivo.
- El ciclo de vida de un fragment está fuertemente relacionado al ciclo de vida de la actividad que lo hospeda, pero recibe sus propios callbacks en la medida que va cambiando de estado, condicionado por el estado de la actividad y por la naturaleza de su interfaz y la interacción del usuario con ella.

## Uso de Fragments en nuestra Aplicación (2)

→ Nosotros hemos definido un fragment que contiene el listado de mesas.



```
interface ViewFragment{  
    public void updateList();  
}  
  
interface ViewContainer{  
    public void onItemSelectedInFragment(Mesa mesa);  
    public void mostrarError(String error);  
}
```



# **SIU GUARANÍ MÓVIL MEJORAS FUTURAS**

## Mejoras Futuras y Nuevas Funcionalidades (1)

- La aplicación que hemos desarrollado podría expandirse para incorporar nuevas funcionalidades o mejorar las que ya tiene, entre ellas:
  - ◆ Desarrollar un servidor que simule al servidor Guaraní para poder realizar peticiones, ya sea de login/logout, inscripción/desinscripción etc.
  - ◆ Actualmente el servicio de sincronización con el servidor Guarani es configurable, pero no para el usuario. La idea es permite al usuario cambiar los horarios de sincronización con el servidor guarani.



## Mejoras Futuras y Nuevas Funcionalidades (2)

- La aplicación que hemos desarrollado podría expandirse para incorporar nuevas funcionalidades o mejorar las que ya tiene, entre ellas:
- ◆ El servicio en background efectúa las consultas de mesas usando username y password definidas en preferencias. Puede quedar inconsistente el password almacenado con el registrado en GUARANÍ y esto puede generar un error. La solución sería la siguiente:
    - El servicio en background envíe una notificación al usuario indicando que el password es incorrecto.
    - Se lleva al usuario a una activity donde pueda ingresar su nueva password. Se verifica que nuevo password sea correcto.
    - Se actualizan las preferencias.
    - En ningún momento se eliminan los datos del alumno almacenados.



¿Preguntas?



**¡MUCHAS GRACIAS!**  
**POR SU ATENCIÓN...**

