Diagrama de Actividades

Descripción General:

El **diagrama de actividades** representa el flujo de navegación de la aplicación. Cada pantalla o fragmento se muestra como un nodo, y las acciones de navegación entre ellos son flechas que indican el destino correspondiente. Este modelo incluye:

- **Inicio (`start`)**: Representa el punto de partida de la aplicación.
- **Decisiones (`if`)**: Representan bifurcaciones condicionales en la navegación, como "¿El usuario completó el login?".
- **Acciones (`:`)**: Representan una actividad que lleva a un estado o fragment específico.
- **Fin (`stop`)**: Indica el término del flujo en algunos casos.

Detalles Clave:

1. **Inicio:**
    - El flujo comienza en el `LoginFragment`. Este nodo representa la pantalla inicial donde el usuario debe autenticarse.
2. **Condición:**
    - Si el login es exitoso (`Login Completo?`), la navegación continúa hacia el `IntroFragment`.
    - Si el login falla, el flujo se detiene (`stop`).
3. **Navegación:**
    - Desde el `IntroFragment`, la navegación pasa al `MainFragment`, la pantalla principal.
    - En el `MainFragment`, hay una decisión condicional:
        - **Si el usuario selecciona ir al menú**, navega al `MenuFragment`.
        - **Si el usuario selecciona otra opción**, navega a otras secciones según el flujo.
4. **Opciones en el Menú:**
    - En el `MenuFragment`, el usuario puede:
        - Navegar a la pantalla de créditos (`CreditFragment`).
        - Navegar a la lista de elementos (`ItemListFragment`).
5. **Favoritos y Detalles:**
    - Desde el `ItemListFragment`, si el usuario selecciona favoritos, el flujo lo lleva al `FavItemListFragment` y luego al `DetailItemFragment`.
    - Alternativamente, puede ir directamente al `DetailFavItemFragment`.

Diagrama de Clases

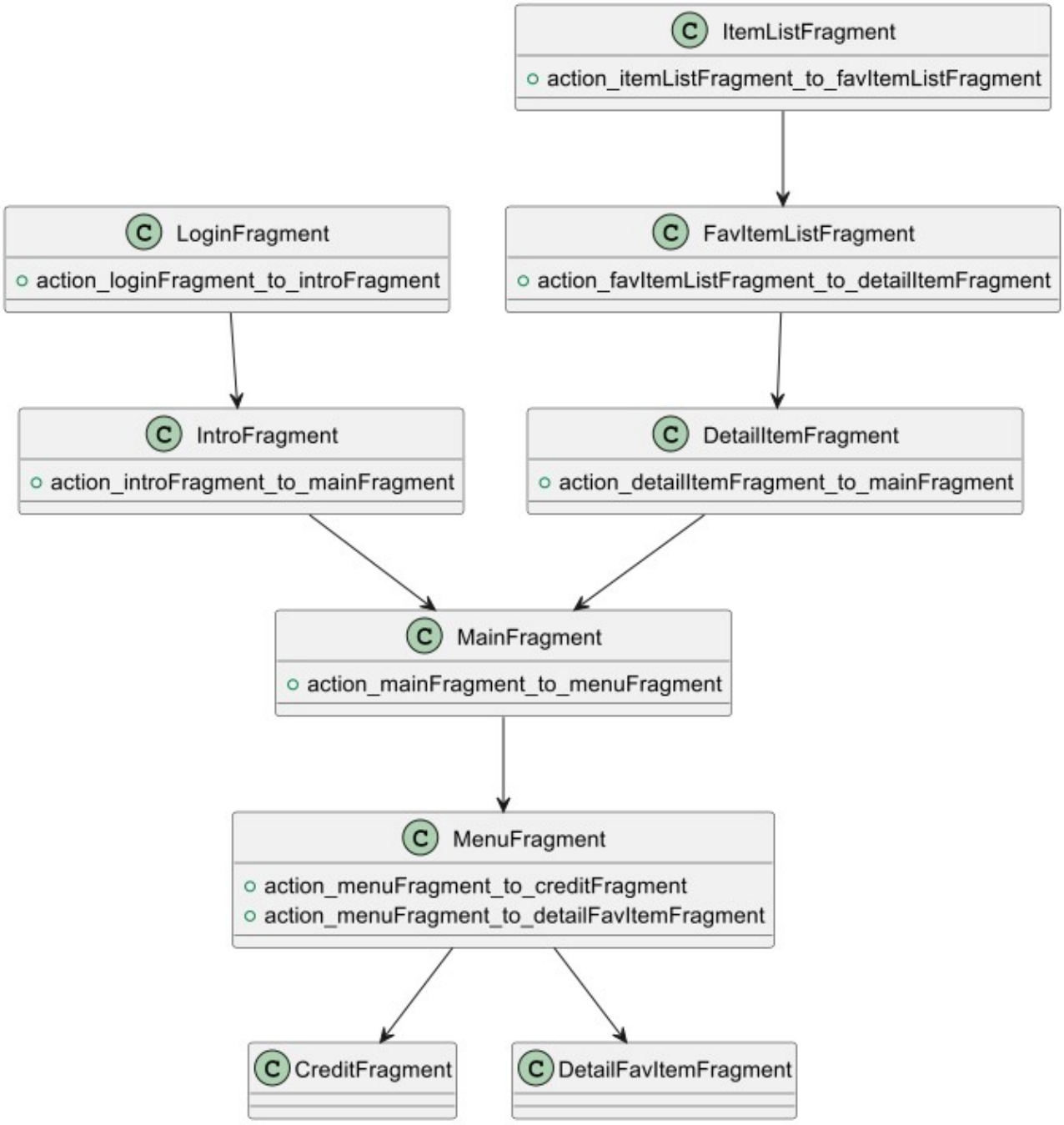
Descripción General:

El **diagrama de clases** representa una estructura estática de los **fragments** como entidades principales de la aplicación y sus **acciones** como métodos que conectan dichas entidades. Aquí se observa:

- **Clases principales:** Cada fragmento es representado como una clase.
- **Acciones:** Cada acción definida en el archivo XML es modelada como un método que lleva a otro fragmento.
- **Relaciones:** Las flechas (`-->`) indican la dirección de navegación entre los fragments.

Detalles Clave:

1. **Fragments como clases:**
    - `LoginFragment`, `IntroFragment`, `MainFragment`, y otros fragments son tratados como clases independientes que encapsulan las pantallas de la app.
2. **Acciones como métodos:**
    - Las acciones, como `action_loginFragment_to_introFragment`, son representadas como métodos de la clase origen. Estas transiciones están definidas en el archivo XML y conectan un fragmento con otro.
3. **Relaciones jerárquicas:**
    - Las flechas (`-->`) indican una relación de dependencia. Por ejemplo:
        - `LoginFragment --> IntroFragment`: El `LoginFragment` tiene un método o acción que lleva al `IntroFragment`.
        - `MenuFragment --> CreditFragment`: El `MenuFragment` contiene una acción que navega al `CreditFragment`.
4. **Agrupación lógica:**
    - Aunque todas las clases son independientes, las relaciones entre ellas modelan el flujo de navegación. Esto ayuda a identificar las dependencias y conexiones directas.



(Esta imagen esta generada con PlantUML, archivo incluido en el proyecto)

Beneficios del Diagrama

1. **Claridad en la Navegación:**
    - Ayuda a los desarrolladores a entender rápidamente cómo fluye la navegación en la aplicación, facilitando la depuración y el desarrollo de nuevas funcionalidades.

2. **Detección de errores lógicos:**
    - Visualizar las transiciones puede revelar bucles, flujos inconsistentes o fragmentos inaccesibles.

3. **Documentación del proyecto:**
    - Sirve como documentación visual para equipos de desarrollo y diseño, mejorando la comunicación y la transferencia de conocimientos.

4. **Facilidad para refactorizar:**
    - Permite identificar dependencias innecesarias o fragmentos que podrían unificarse o eliminarse.