Supongamos una aplicación que inicia con una ventana de ***login***, donde el usuario debe ingresar sus credenciales. La aplicación las valida contra algo (una base de datos ***SQLite***, una base de datos externa, algo, no importa qué), y si son aceptadas, le muestra una nueva ventana para que registre determinada información.

Luego de ese ingreso, tiene tres opciones más de ingreso, cada una de las cuales muestra más información a mostrar. Y tiene opciones para ver listados, gráficos, fotos, o lo que sea. O sea, una aplicación típica, con varias pantallas, en las que el usuario pasa de una a otra, y la aplicación envía y recibe información entre cada una de ellas.

Nada que no conozcamos. Cada pantalla será una ***Activity*** (que tendrá su propio ***Layout***, como siempre). Gracias a nuestro amigo ***Jose Intent*** podremos ir saltando de ***Activity*** en ***Activity*** cuando nos venga en gana. Y gracias a su fiel asistente ***Rigoberto Bundle***, el ***Intent*** podrá transportar información de una ***Activity*** a otra. Todo muy lindo.

—*Sí, todo muy lindo* —dijo uno de los creadores del Android, poco antes de parir la versión 3.0—, *pero tenemos que hacer algo mejor que esto. Tenemos que inventar una forma que permita a las aplicaciones tener una única* ***Activity****, y poder hacer que esa única* ***Activity*** *pueda mostrar bloques funcionales independientes*.

—*Bloques funcionales independientes? Qué es eso?* —le preguntó otro, mientras saboreaba su Caramel Frappuccino, y pensaba por qué será que “alto” es menos que “grande”.

—*Si. Estaría chetísimo que tuvieras una única* ***Activity****, en la que definieras uno o más sectores dentro de su* ***layout****, y en cada uno de ellos pudieras mostrar el resultado visible de un fragmento de código. Por ejemplo: tenés un fragmento de código que muestra una lista de algo. Su código puede ser tan complejo como quieras, pero su resultado visible es una lista de elementos. Entonces, ese fragmento muestra esa lista. Después, tenés otro fragmento de código cuyo resultado visible es un* ***TextView****, un* ***EditText****, y un Botón. Y así, cada fragmento de la aplicación se formaría por código que genera un proceso, y alguna clase de resultado visible de ese código. Y la* ***Activity*** *principal (en realidad, la única* ***Activity*** *que tendría la aplicación), funcionaría como anfitriona de cada uno de esos fragmentos de código.*

—*¡Copadísimo mal, chabón!* —gritó otro mientras escupía parte de su Big Mac encima del teclado de su ídem—. *Entonces, como cada uno de esos fragmentos de código son invocados siempre desde la* ***Activity*** *anfitriona, podrían interactuar entre sí muy fácilmente: basta con que en la* ***Activity*** *anfitriona haya ciertas variables y métodos públicos, para que cualquiera de esos fragmentos de código puedan guardar o leer información en ellos. De esa forma, no tienen que andar pasándose información entre ellos, sino que simplemente la información está toda ahí, en la* ***Activity*** *anfitriona*.

—*¡Mejor todavía, chavales! —*exclamó otro mientras comía una galleta de arroz con queso blanco sin sal y entraba a la WEB para descargarse el sabor*—. Siendo que cada fragmento de código estará programado sin saber dónde va a ser mostrado, sino solamente sabiendo qué es lo que debe mostrar, podríamos hacer que una misma aplicación, para su* ***Activity*** *anfitriona, tenga distintos* ***layouts****: uno para la pantalla en vertical, otro para la pantalla en horizontal, otro para tablets chiquitas, o para tablets grandes, otro para* ***Android TV****, y así. De esa forma, la misma aplicación, sin tocarle el código, se adaptaría automáticamente a cada dispositivo, aprovechando al máximo su pantalla. Eso sí: es fundamental que cada uno de estos fragmentos funcionales de código esté programado de forma que sea completamente independiente de los otros. Es decir, no debe tener que depender de los otros para su funcionamiento, debe depender únicamente de la* ***Activity*** *anfitriona, pero no de otros fragmentos funcionales.*

*—Yo tengo una duda —*dijo otro, rogando que no le respondieran “la podonga”*—. ¿Cómo sugieren que llamemos a este concepto que consiste en fragmentar la aplicación en pequeños módulos funcionales independientes, o sea, a un conjunto de fragmentos de código que forman la aplicación completa?*

**Fragments**

El concepto no es complicado, pero podría prestarse a confusión al principio, en especial por ser bastante diferente a lo que veníamos haciendo. Así que, como siempre, hagamos un ejemplo de código, muy paso a paso, mientras tenemos al mismo tiempo un ojo en este tutorial y otro ojo en el Android Studio (como hacía Néstor Kirchner).

Nuestra primera aplicación de ejemplo deberá mostrar un campo de texto, un ***checbox*** y un botón. Al presionar el botón, deberá mostrarse en una nueva pantalla, la palabra “***Sr.***” o “***Sra.***” (según el ***checkbox*** esté tildado o no), y el nombre ingresado. A diferencia de lo habitual, NO lo haremos con dos ***activities*** comunicadas por un ***Intent*** que lleva un ***Bundle***, sino con ***fragments***.

El primer paso es simple: creamos una aplicación común y corriente, con nuestra ***ActividadPrincipal*** de siempre, con su ***layout actividad\_principal.xml*** habitual, reemplazando el ***RelativeLayout*** por nuestro ***LinearLayout***. Y, con un elemento nuevo dentro: un ***FrameLayout***.

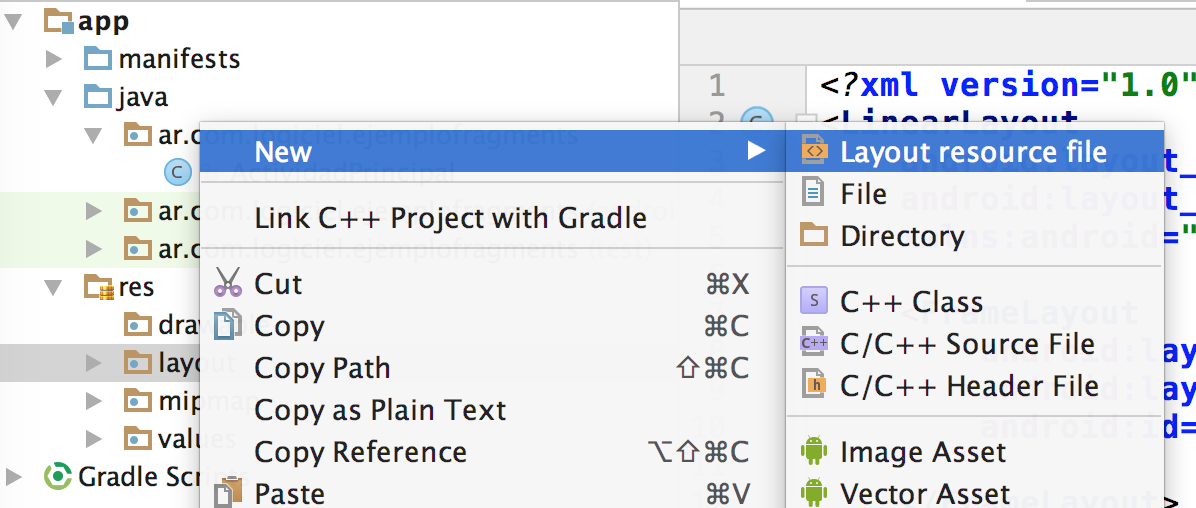


Qué es ese ***FrameLayout***? Es el sector de la ***ActividadPrincipal*** en el que iremos alojando cada uno de los ***fragments*** que se ejecutarán. Es decir, la ***ActividadPrincipal***, al iniciar (o sea, en el ***onCreate***), dirá que un determinado ***fragment*** se aloje ahí. Y luego, en base a ciertos eventos, irá cambiando cuál ***frament*** se alojará ahí.

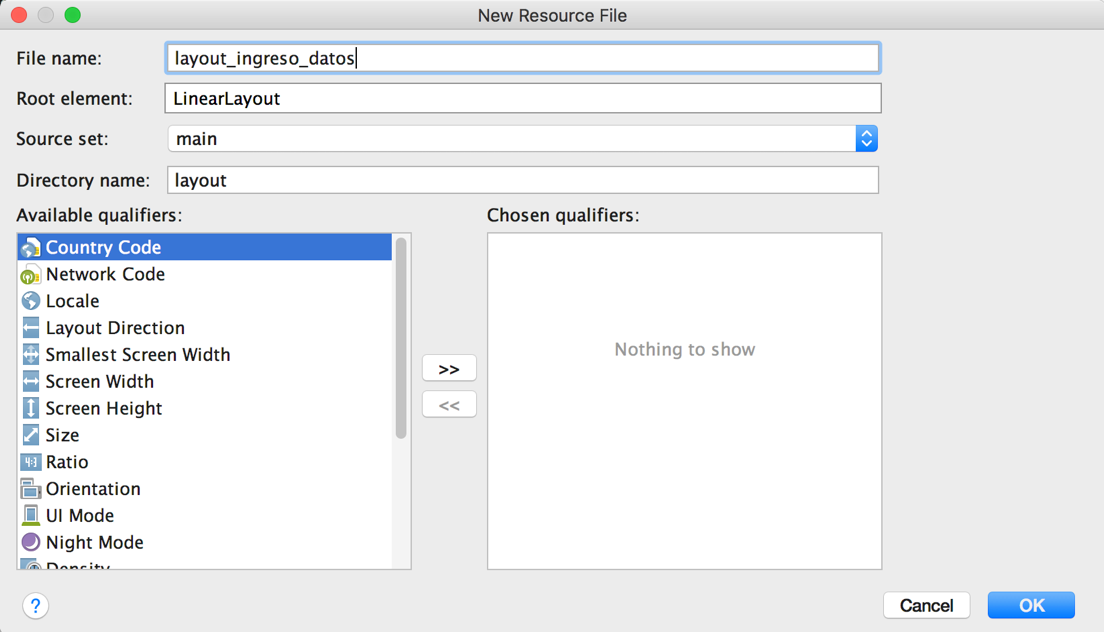
Una aclaración importante: podríamos tener varios ***holders*** (alojadores) de ***fragments*** en ese ***layout*** simultáneamente. Obviamente, en ese caso no podrían tener “***match\_parent***”, sino alguna medida que permitiera que varios quepan en la pantalla al mismo tiempo. En mi ejemplo estoy poniendo solo uno, para facilitar su comprensión.

Ahora, el paso siguiente, es definir el ***layout*** del ***fragment*** que se utilizará para pedir el ingreso de los datos. Ojo acá: hasta ahora, nosotros siempre usábamos un ***layout*** asociado a una ***Activity***. Ya no más. Ahora vamos a usar un ***layout*** suelto, no asociado a nada, independiente (perdón, Albi).

Cómo creamos un ***layout*** suelto?



Click derecho sobre ***\res\layout***, y ***New – Layout resource file***. En la ventana que aparece completamos el nombre del nuevo ***layout***.

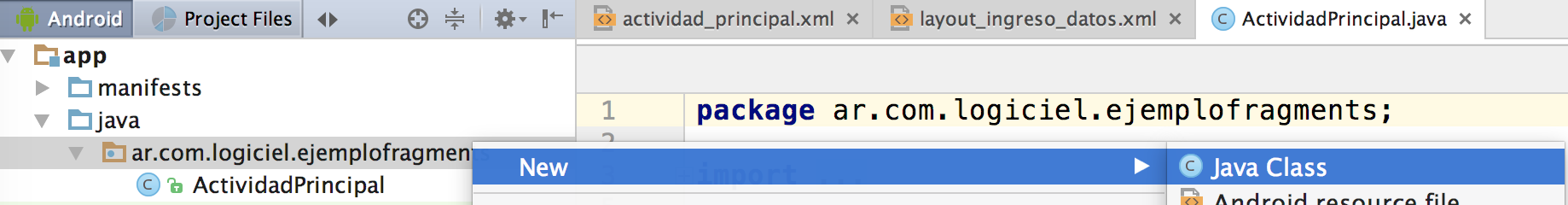


Podemos decirle también que su elemento raíz (***Root element***) sea un ***LinearLayout***, simplemente para que nos aparezca escrito y no tengamos que escribirlo nosotros.

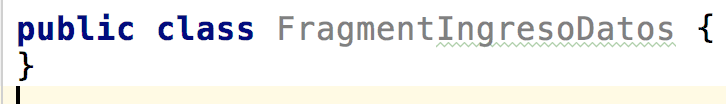
Qué tendrá ese ***layout***? Tendrá lo que queremos que muestre el fragmento funcional de ingreso de datos. Tendrá entonces un ***TextView***, un ***EditText***, un ***CheckBox*** y un botón.



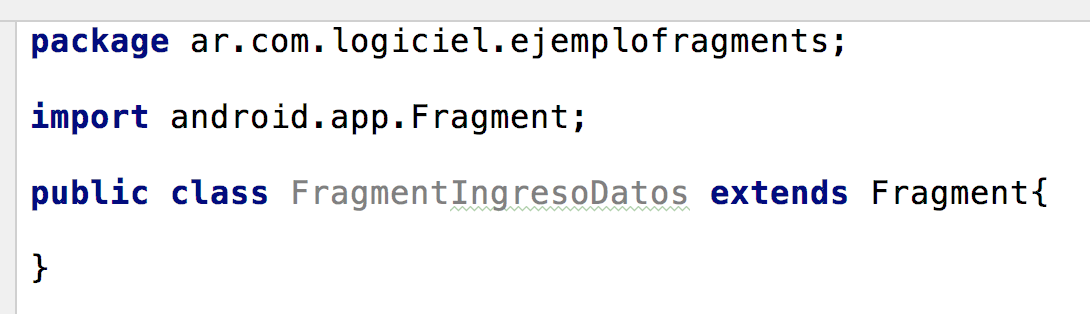
El paso siguiente nos lleva a un terreno nuevo y desconocido: crear la lógica de funcionamiento de este ***fragment***, para ser ejecutada cuando alguna ***Activity*** anfitriona así nos lo requiera. Para eso, creamos una clase:



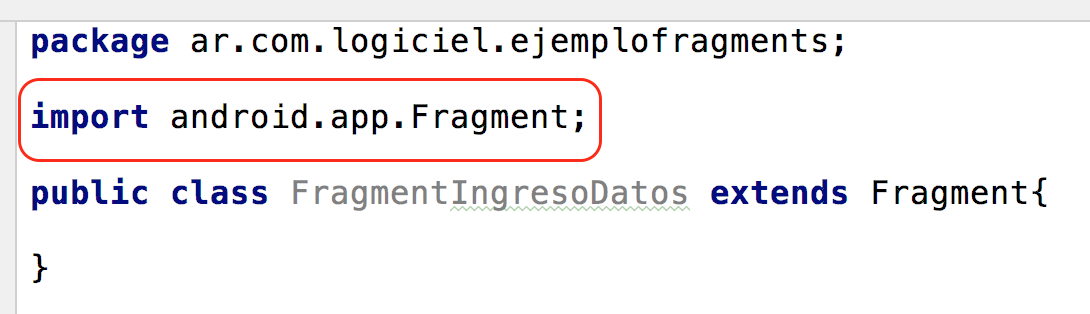
En mi ejemplo la llamé ***FragmentIngresoDatos***:



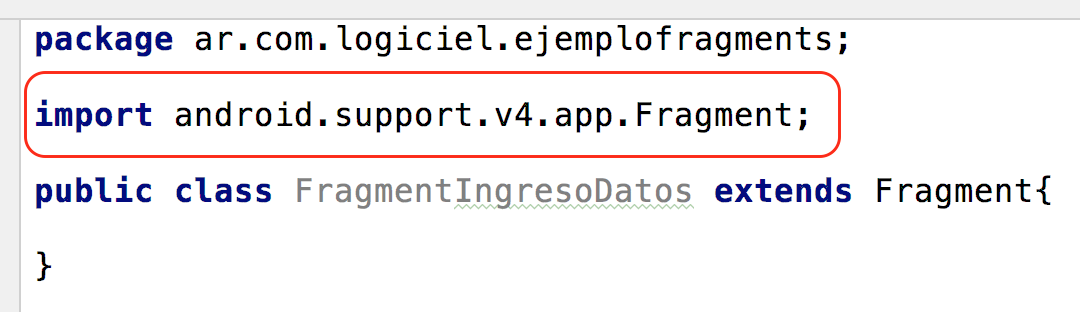
E indicamos que esa clase hereda de ***Fragment***



Antes de seguir, debemos hacer un pequeño ajuste manual. Resulta que los ***fragments*** aparecieron en la versión 3.0 de Android, por lo que los desarrolladores tuvieron que hacer un pequeño injerto para lograr compatibilidad retroactiva. Como resultado, hay dos versiones de ***fragments***, y nosotros no queremos usar la que viene por default sino la otra. Entonces, el cambiecito será reemplazar esto:



Por esto:



Ahora sí, vamos a codear el funcionamiento de ese ***fragment***. Como sabemos, ese ***fragment*** cobrará vida en el instante en que una ***Activity*** anfitriona lo coloque en algún ***holder*** de su ***layout***. En ese momento, el ***fragment*** intentará ejecutar un método llamado ***onCreateView***. Si no existe, no pasa nada, pero no veremos nada. Entonces, debemos crearlo, e indicarle a ese ***onCreateView*** que debe leer, en tiempo de ejecución, el ***layout*** que preparamos para él (lo habíamos llamado ***layout\_ingreso\_datos***).

Veamos el código, y luego lo explicaremos un poco:

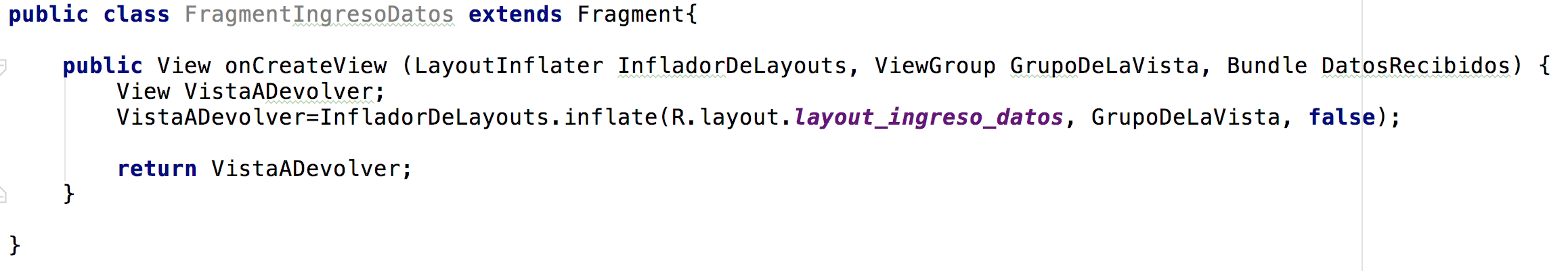


El ***onCreateView*** espera recibir tres parámetros obligatorios. El primero, de tipo ***LayoutInflater***, lo llamamos ***InfladorDeLayouts***, y será el elemento encargado de leer y expandir en tiempo de ejecución (lo que Android llama “***Inflar***”) el ***layout*** que usará este ***fragment***.

El segundo, de tipo ***ViewGroup***, lo llamamos ***GrupoDeLaVista***, y será un elemento necesario para inflar el ***Layout***. El tercero no lo usamos por ahora, así que nos ahorramos la explicación.

También vemos que la llave de cierre está en rojo, y entendemos que es porque esa función debe devolver un ***View***. Pero, qué es ese ***View***?

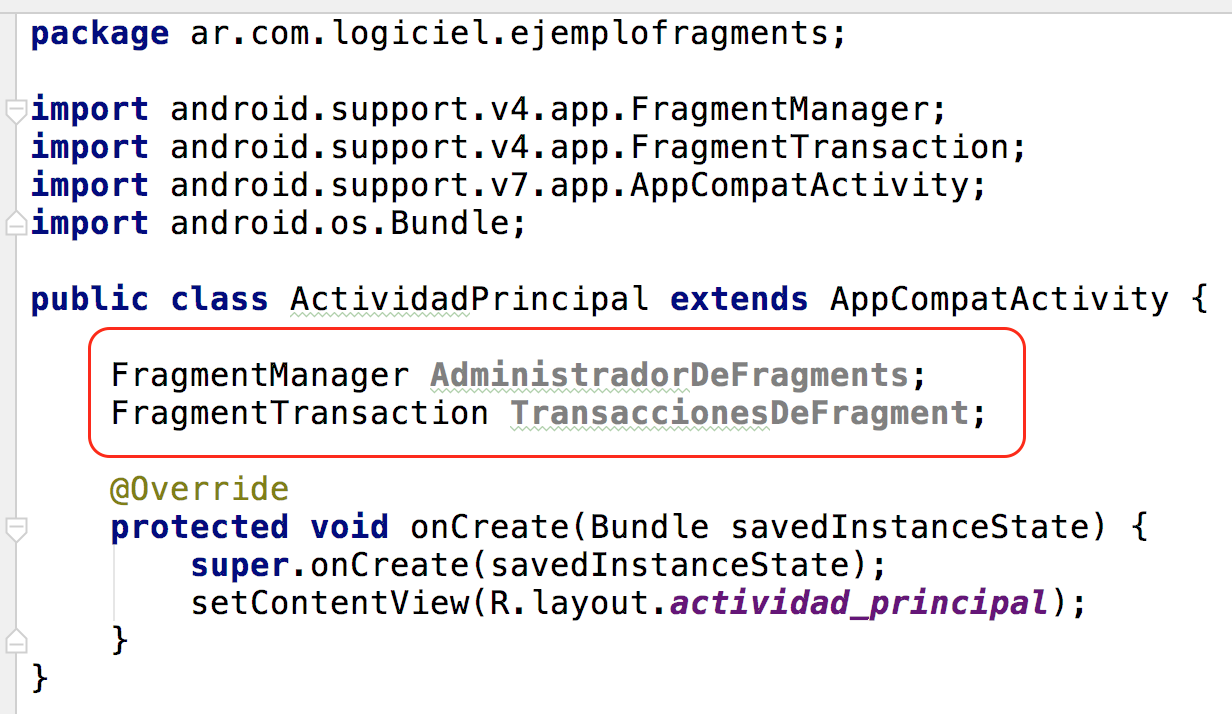
Ese ***View*** es la representación gráfica visible del ***layout***. Justamente, lo que vamos a hacer es leer y expandir (inflar) el ***layout*** que preparamos, y hacer que el ***onCreateView*** devuelva esa Vista.



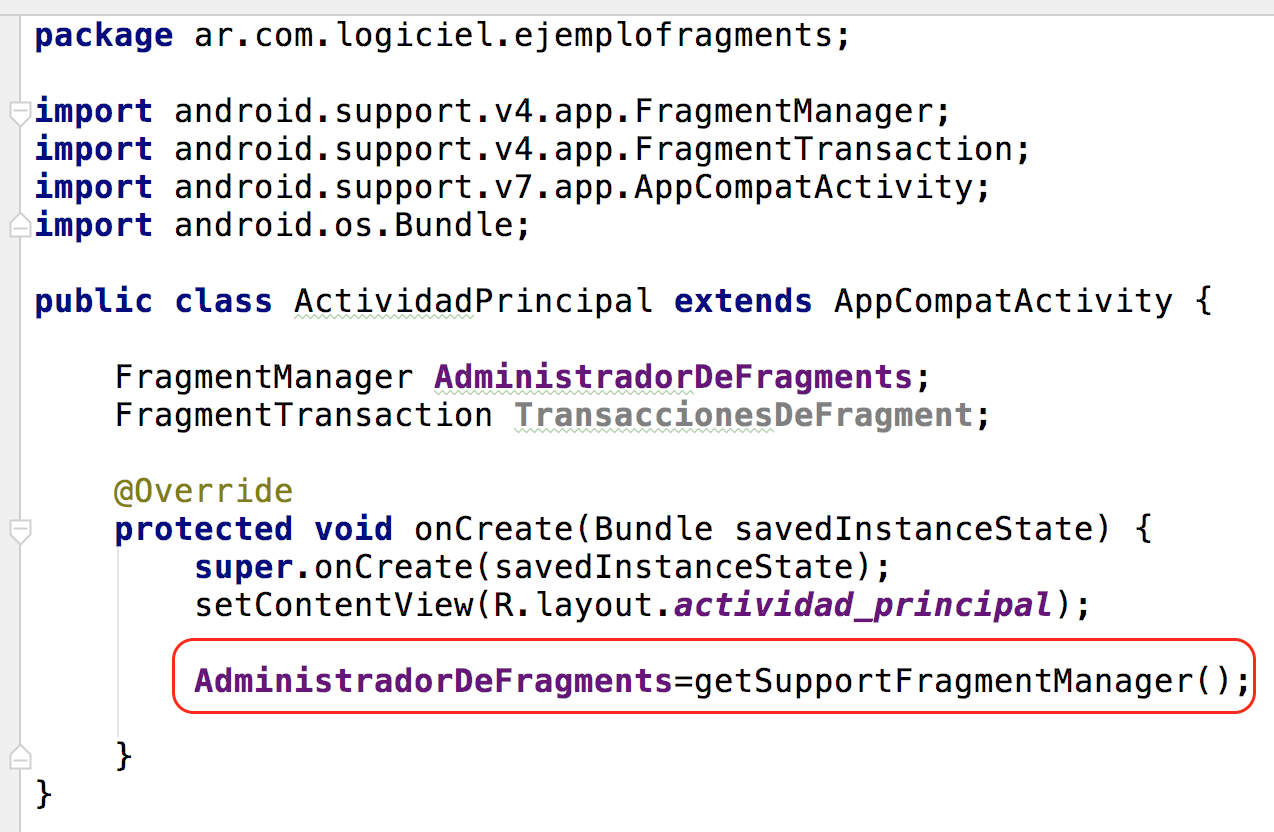
Por más que aún no entendamos del todo lo que estamos haciendo, conviene seguir avanzando para que las cosas se aclaren un poco más. El ***fragment*** está listo para ser usado.

Ahora, volvemos a la ***ActividadPrincipal***, la única, la anfitriona. En ella, en su ***onCreate***, vamos a decirle al ***FragmenIngresosDatos*** que se ejecute alojado en el ***holder*** que habíamos puesto en su ***layout***.

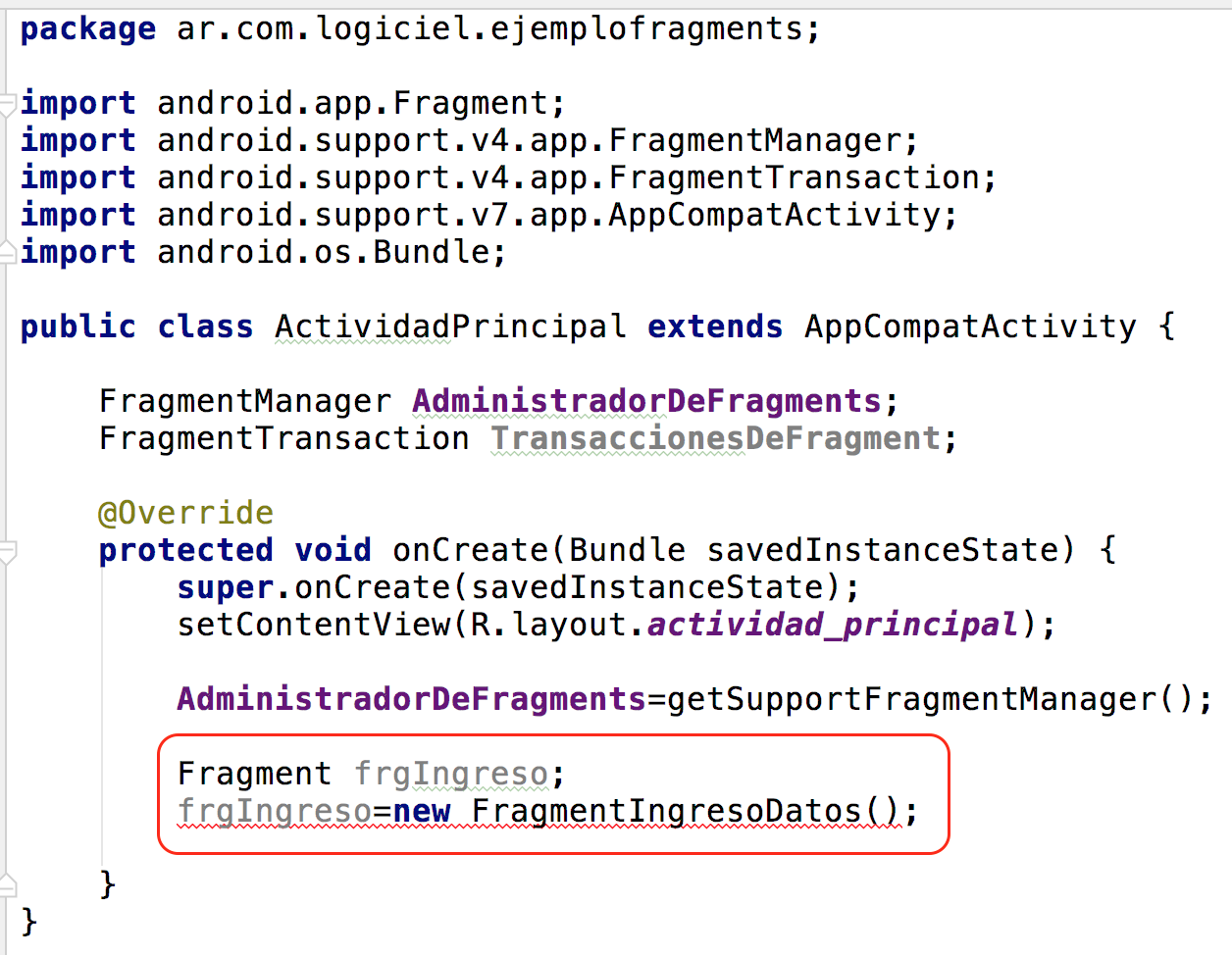
Para eso, necesito dos elementos: un ***FragmentManager***, y un ***FragmentTransaction***. Los declaro globales, para poder usarlos en cualquiera de sus métodos.



Inicializo solo el ***FragmentManager***, por ahora.

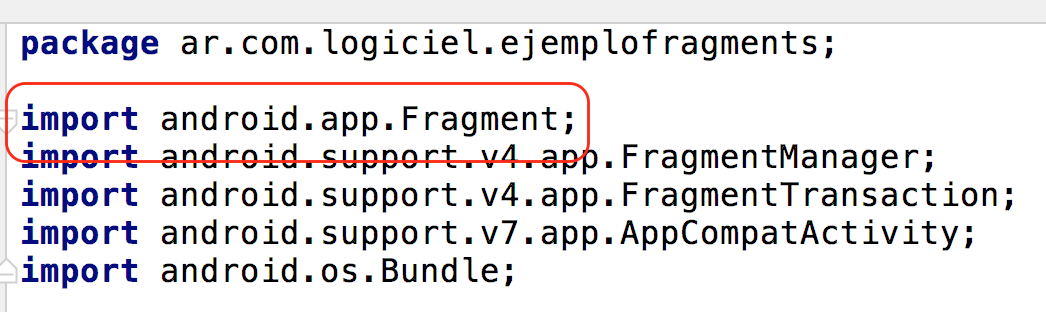


Y ahora llega el momento de la verdad: primero, voy a declarar un objeto de tipo ***fragment***, y lo voy a instanciar del tipo de la clase que creamos.

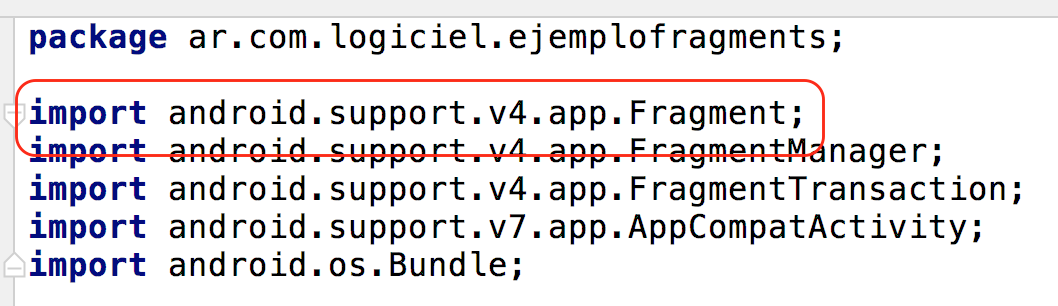


Perfecto, pero por que en rojo? Qué es lo que no le gusta? Android nos odia?

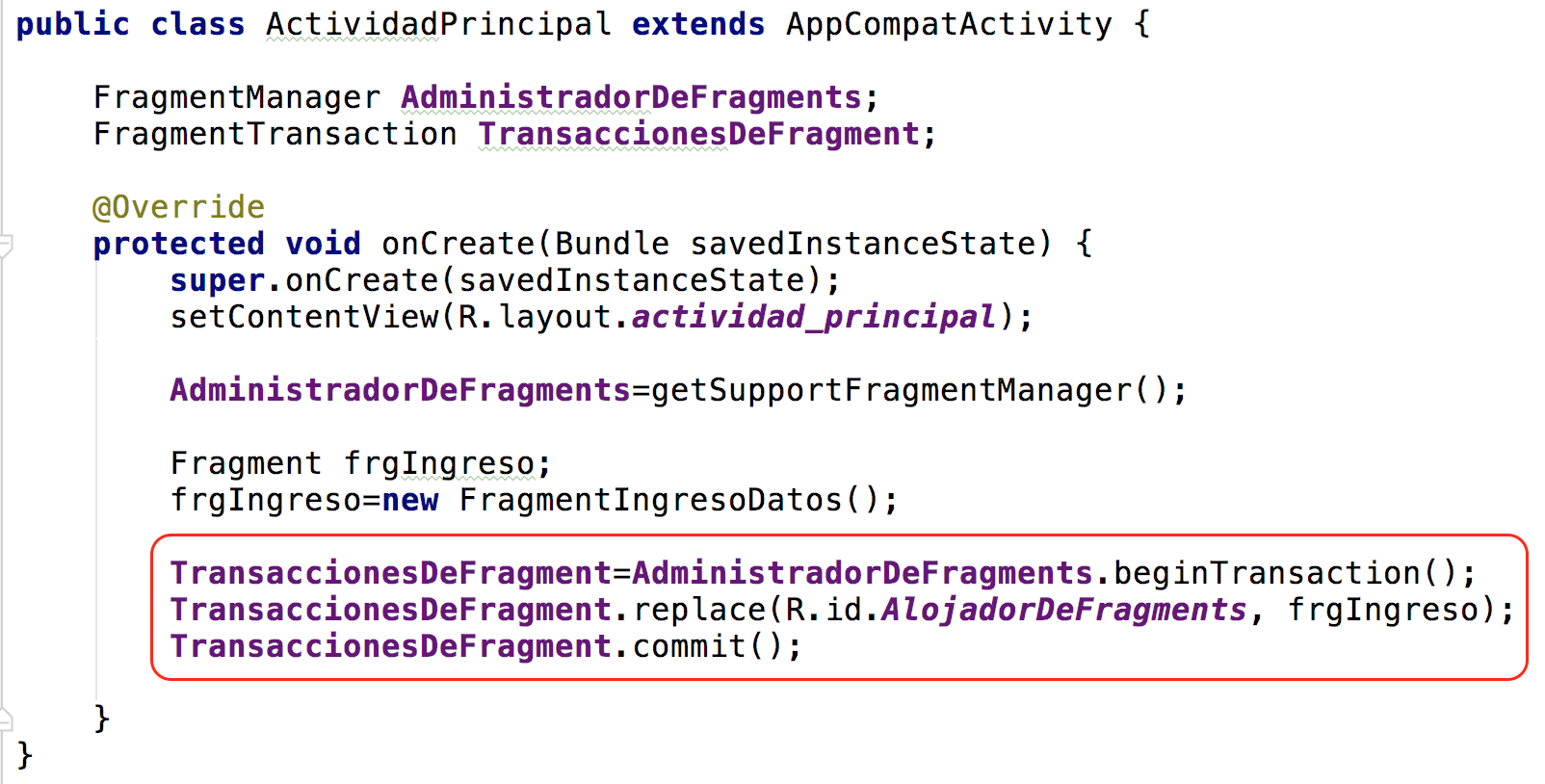
No, nos odia, no ama, ya lo sabemos. Nos faltó un pasito simple, el mismo que antes. Cambiar esto:



Por esto:



Genial, genial. Ya casi. Nos falta una sola cosa: decirle a Android que queremos que muestre en el ***holder*** (alojador) que preparamos con tanto cariño en el ***layout*** de la ***ActividadPrincipal***, el resultado de este ***fragment***. Así:



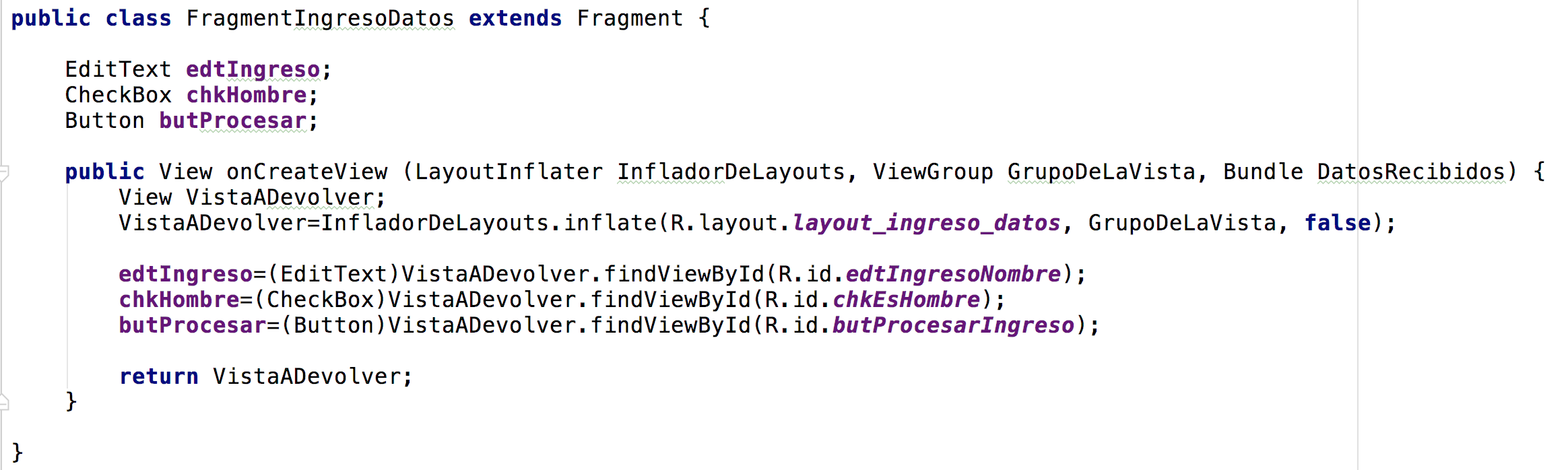
Le estamos diciendo:

Iniciá un proceso transaccional de asignación de ***fragments*** en ***holders***, alojá en el ***holder*** llamado ***AlojadorDeFragments*** (qué habíamos creado en el ***layout actividad\_principal*** allá lejos y hace tiempo), el ***fragment*** llamado ***frgIngreso***.

Es un excelente momento para compilar y confirmar que todo ande hasta acá.

Seeeee, vamo’ carajo! Funciona. Bien, avancemos entonces.

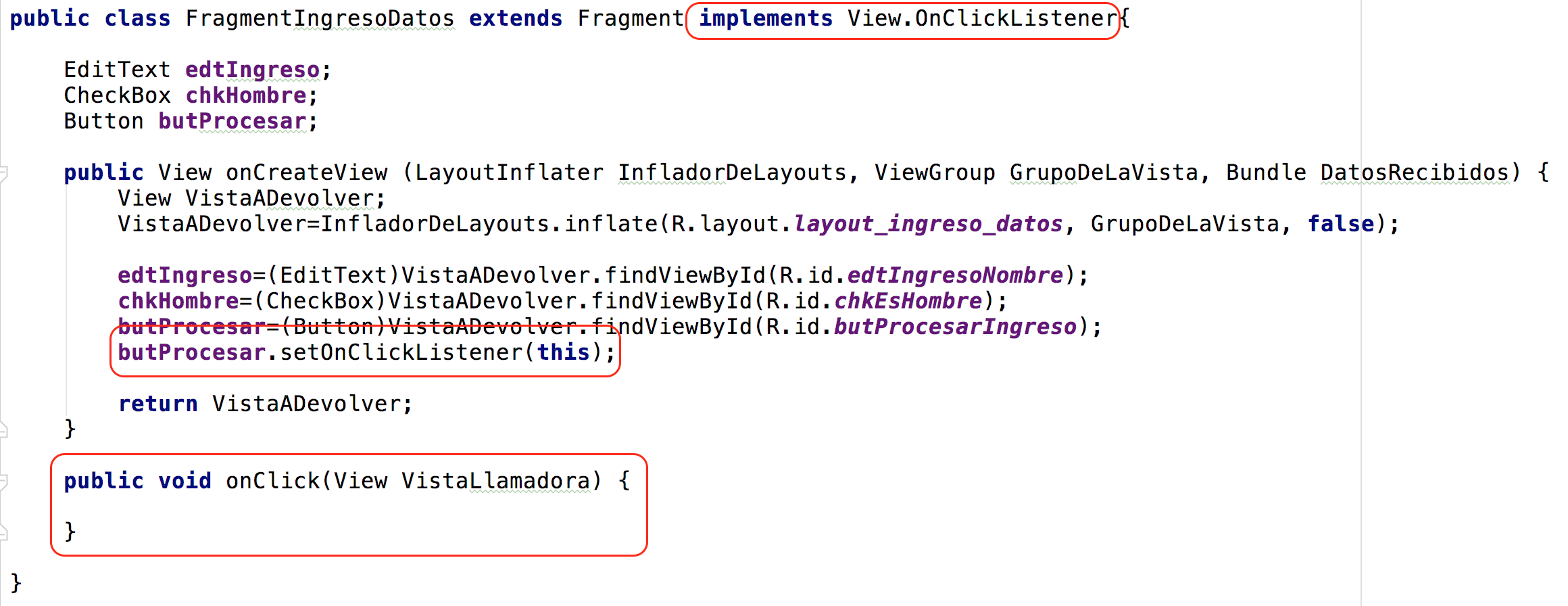
Paso siguiente, de vuelta en el ***fragment***: asignar los objetos relacionados para leer los valores. Pero ojo, que no es igual que siempre, es parecido:



Qué notamos de distinto? Pues claro, amiguitos: que el ***findViewById*** no va sueltito, como antes, sino que va “atado” a ***VistaADevolver***. Es decir, NO lee directamente del ***layout*** que está “guardado en el disco”, sino que lee del ***layout*** que se leyó en tiempo de ejecución y se expandió (infló) en memoria, llamado ***VistaADevolver***.

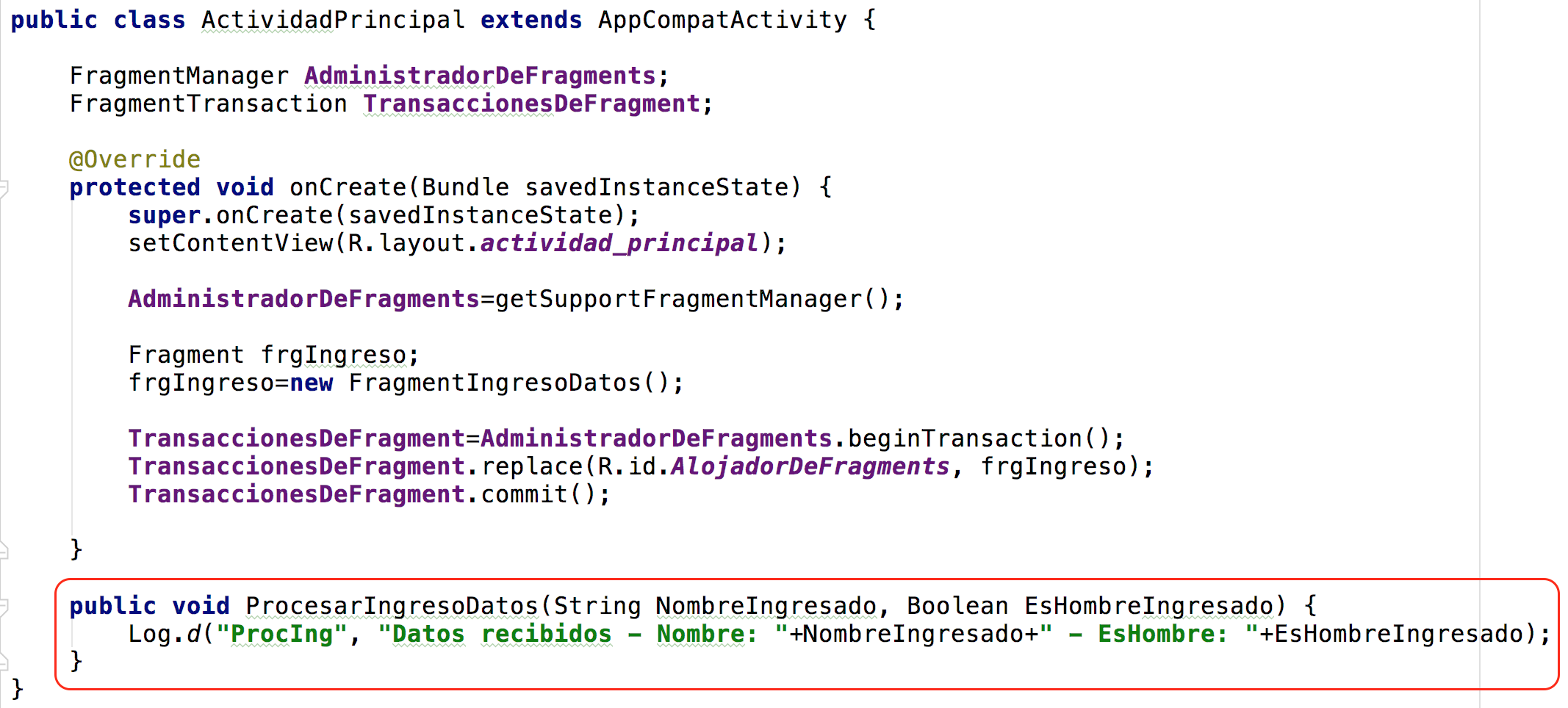
Paso siguiente: asignarle ***onClick*** al botón, para que dispare un evento al ser presionado. Por qué no lo hacemos con el ***onClick*** en el ***layout***, como siempre? Mismo motivo: ese ***layout*** en estado puro no nos sirve, sino que debemos usar su representación en memoria en tiempo de ejecución.

Entonces:

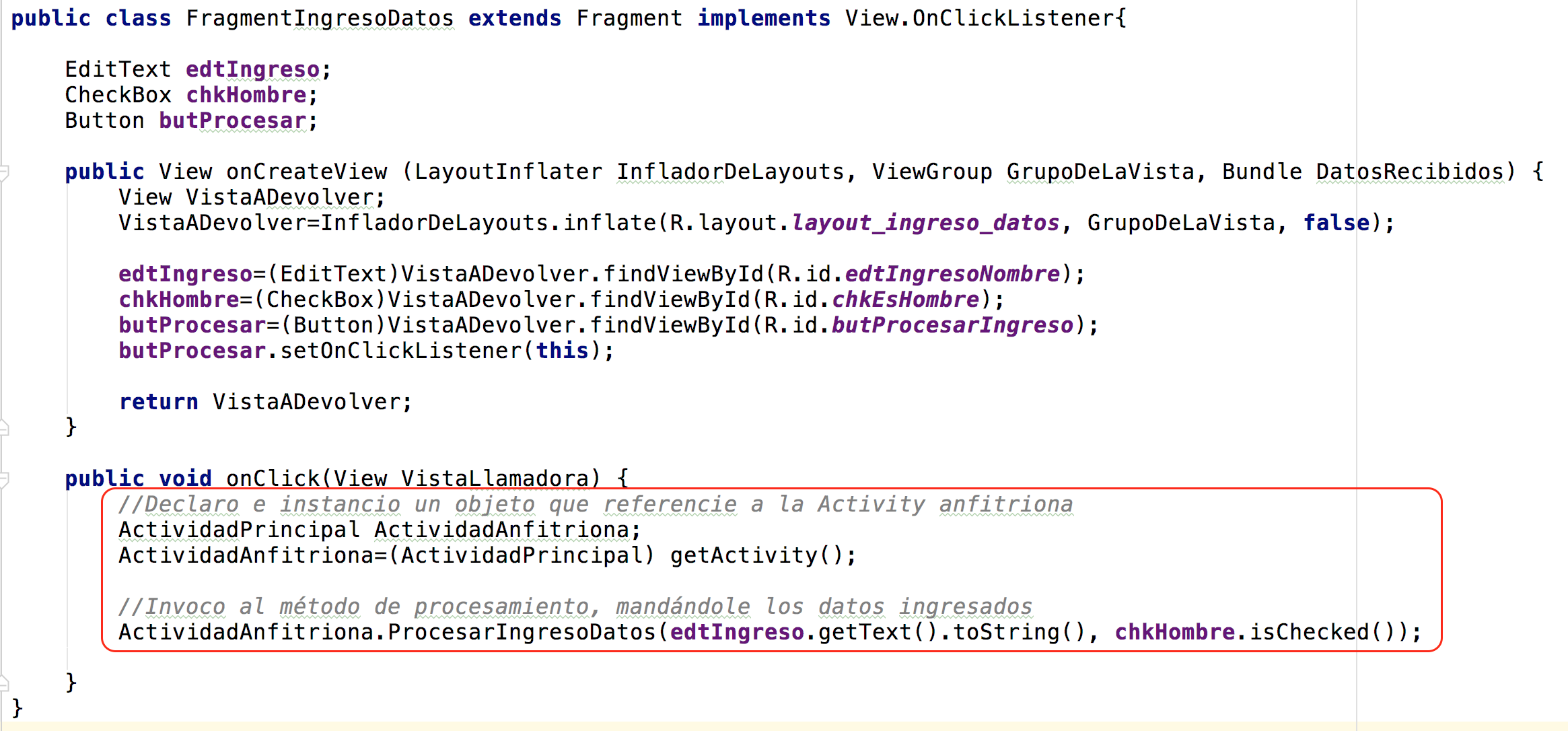


Qué nos falta? Nos falta hacer algún método en la ***ActividadPrincipal***, la anfitriona, que podamos invocar desde este ***fragment***.

Lo hacemos, aunque de momento solo reciba los datos y los muestre en un ***Log***:

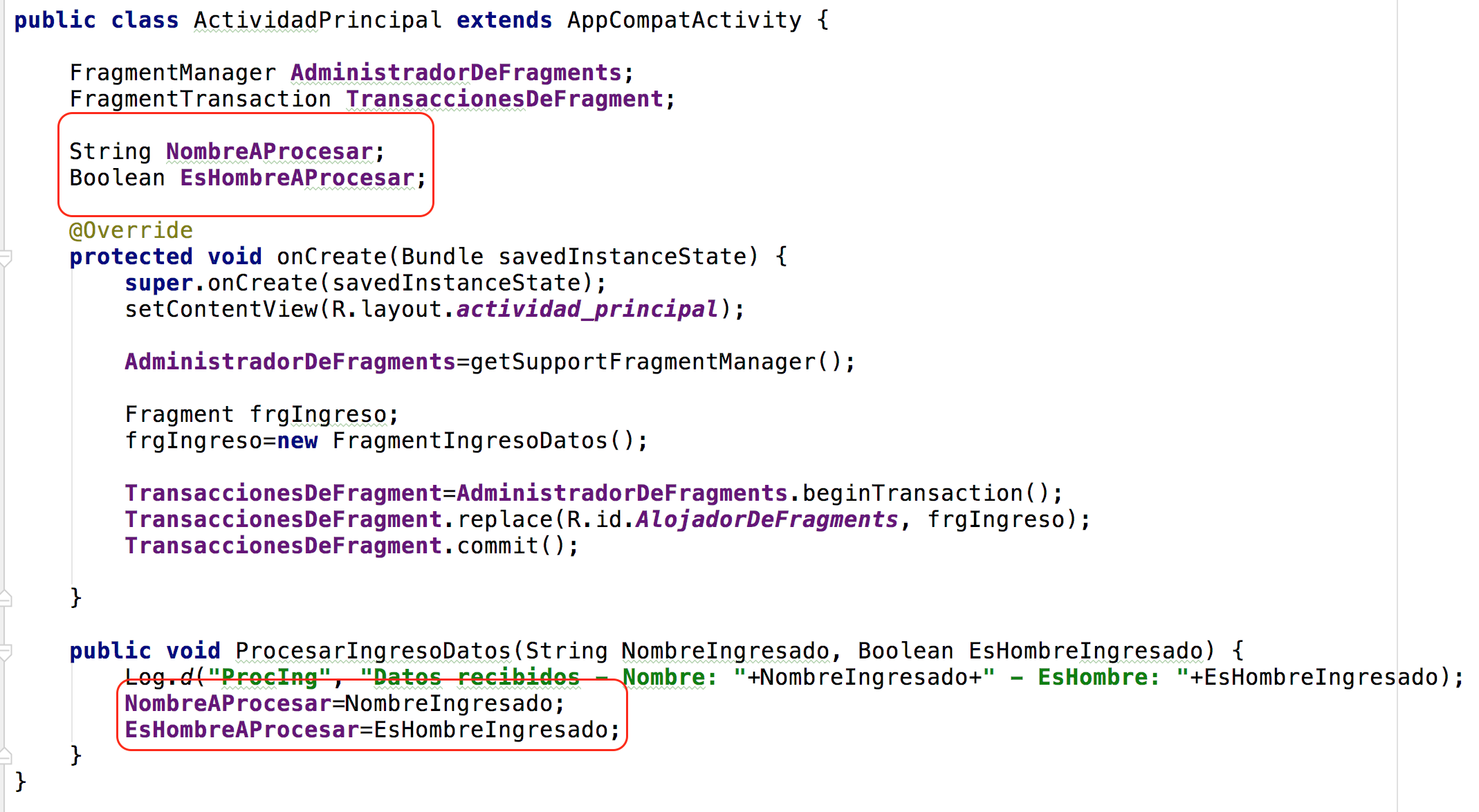


Ahora si, en el ***fragment***, podemos hacer que se invoque a este método de la clase anfitriona:



Momento para ejecutar y mirar el Log, para ver que todo ande como debe.

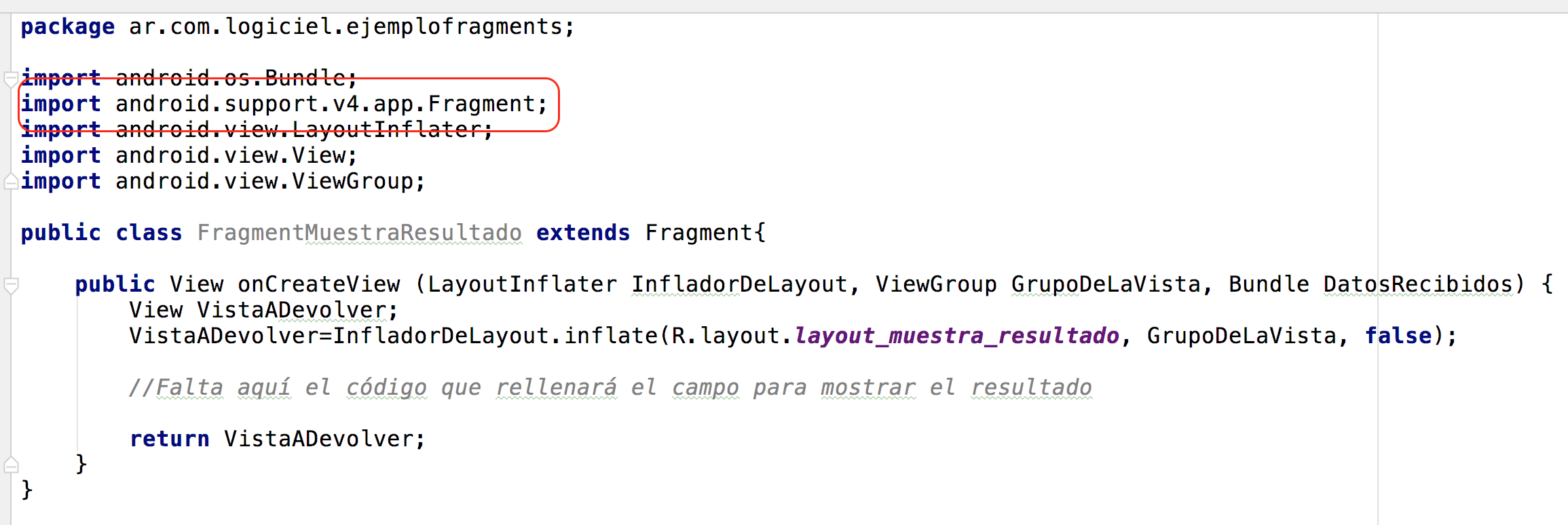
Seguimos. Ahora vamos a guardar los datos recibidos en variables públicas, de forma que puedan ser accedidas desde fuera de esta clase.



Vamos ahora a preparar un segundo ***fragment***. Al igual que antes, primero hacemos su ***layout***, creándolo en forma independiente. En este caso será muy simple, ya que solo tendrá un ***TextView*** para mostrar resultados.



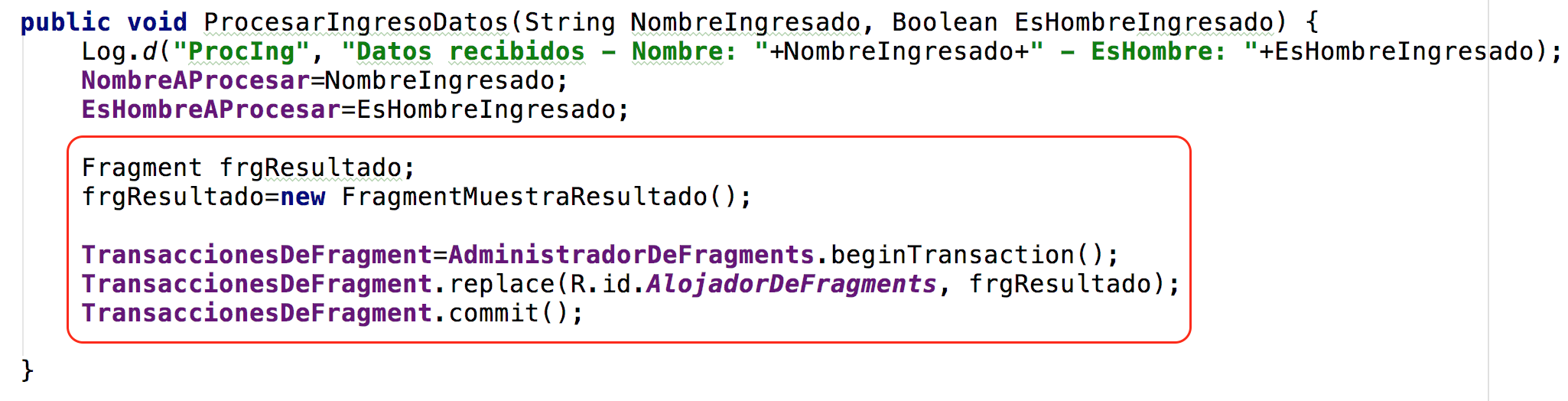
Creamos ahora la clase con la lógica de programación de este segundo ***fragment***. Recordemos cambiar el ***import*** para el ***fragment***.



Como vemos, la lógica es muy parecida a la del primer ***fragment***. Vamos a completar el código.



Qué nos falta? Nos falta un último paso: en la ***ActividadPrincipal***, indicar que queremos mostrar este segundo ***fragment*** en el ***holder*** que antes ocupaba el primero.



Ahora si, debiéramos ver que la ActivityPrincipal, la única, cambia el contenido que muestra en tiempo de ejecución.

Bienvenidos los ***fragments*** a nuestra vida.