Desarrollo de apps con Android

3. Depurador Android Studio

Errores

- Resultados incorrectos o inesperados, valores inválidos.
- Fallos de memoria, excepciones, la app se bloquea…
- Motivos:
  - Errores de implementación o diseño humano -> Corregir el código.
  - Fallo en las librerías software -> Trabajar con limitaciones.
  - Fallo o limitaciones del Hardware -> Hacer que la app funcione con lo que esté disponible.

Depuración (Debugging)

- Encontrar y corregir errores.
- Corregir comportamientos inesperados o no deseables.
- Las pruebas unitarias ayudan a identificar errores y a prevenir regresión.
- Las pruebas de usuario ayudan a identificar errores de interacción.

Herramientas depuración Android Studio

Android Studio proporciona herramientas que ayudan a:

- Detectar problemas.
- Encontrar en qué parte del código fuente se ha creado el problema.

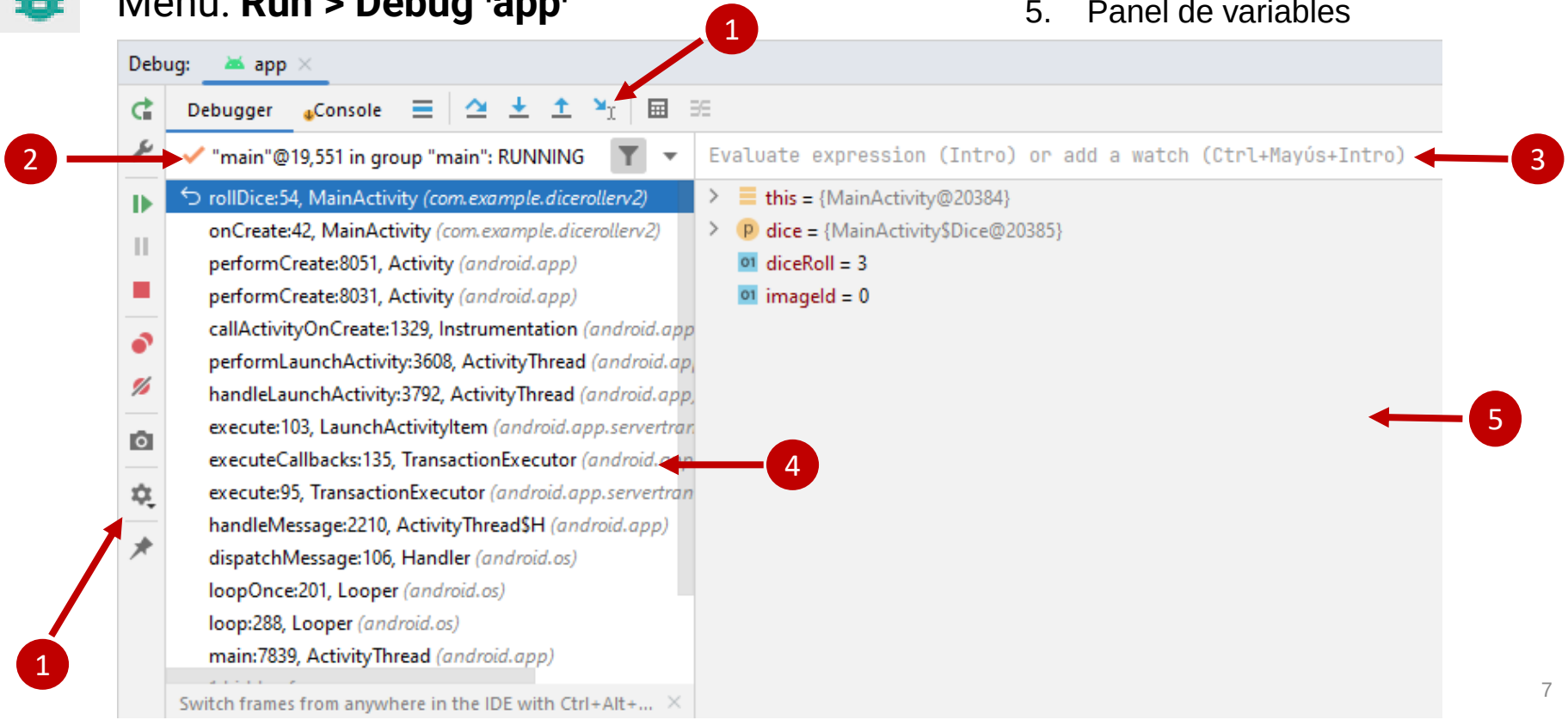
Qué se puede hacer

- Ejecutar la app en modo depuración (*debug*).
- Establecer y configurar los puntos de interrupción (*breakpoints*)
- Parar la ejecución en los puntos de interrupción.
- Inspeccionar la pila de ejecución y los valores de las variables.
- Cambiar los valores de las variables.
- Ejecutar el código línea a línea.
- Pausar y reanudar la ejecución de la app.

Ejecutar en modo depuración (debug)

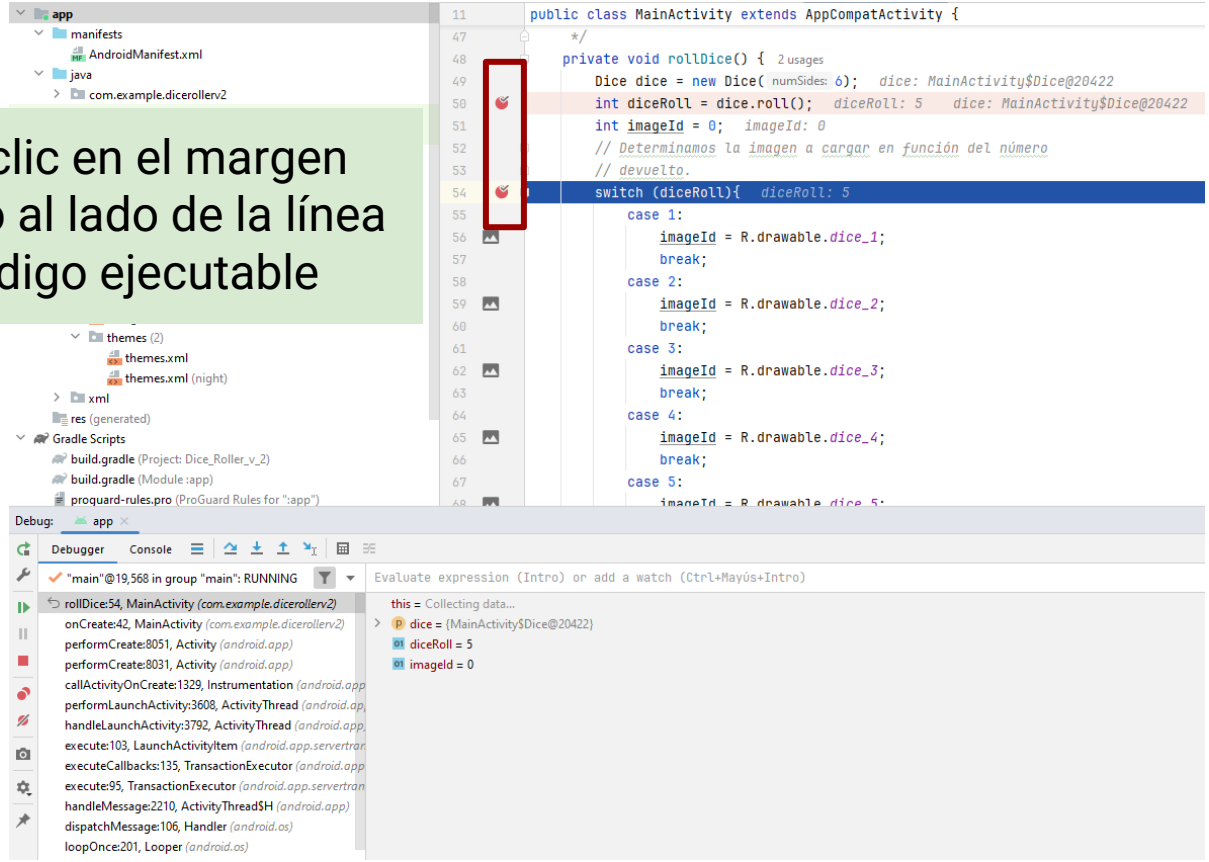
1. Barra de navegación y ejecución
2. Selector del hilo (thread)
3. Variables de inspección
4. Pila de ejecución
5. Panel de variables

Menú: **Run > Debug 'app'**

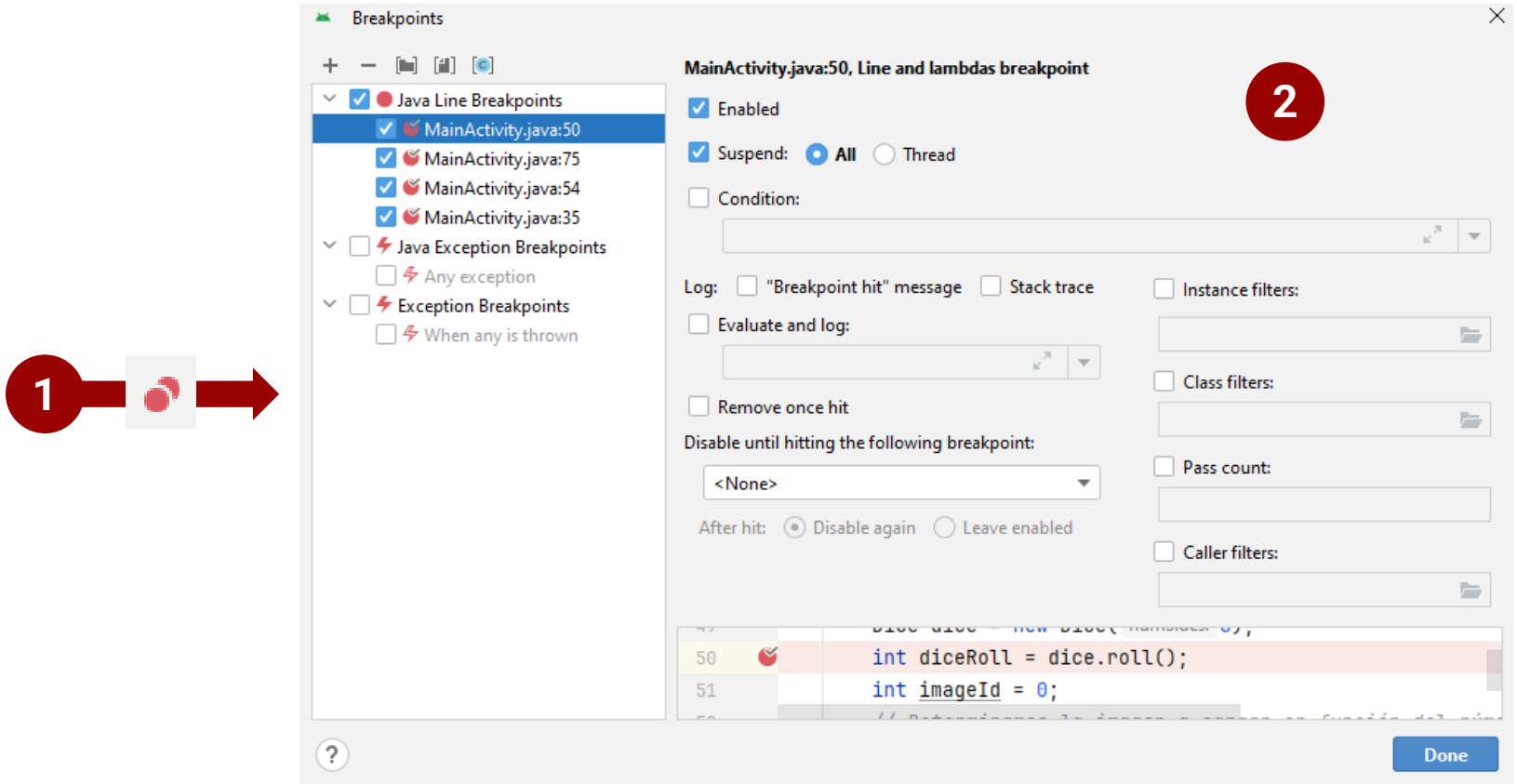


Establecer puntos de interrupción

Hacer clic en el margen izquierdo al lado de la línea de código ejecutable

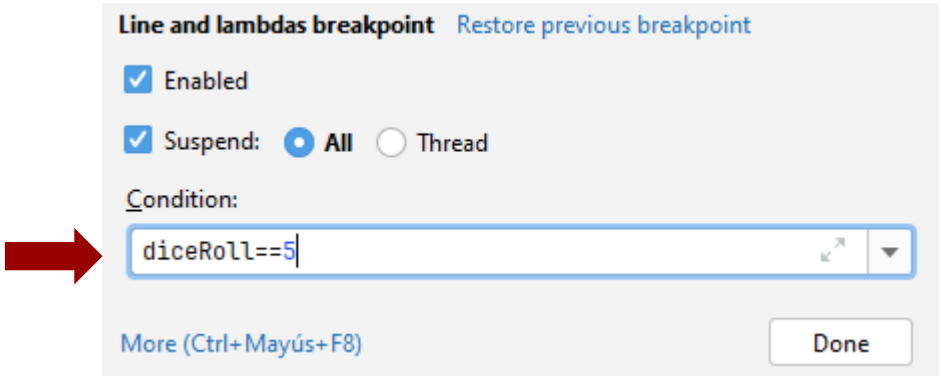


Editar las propiedades de puntos de interrupción

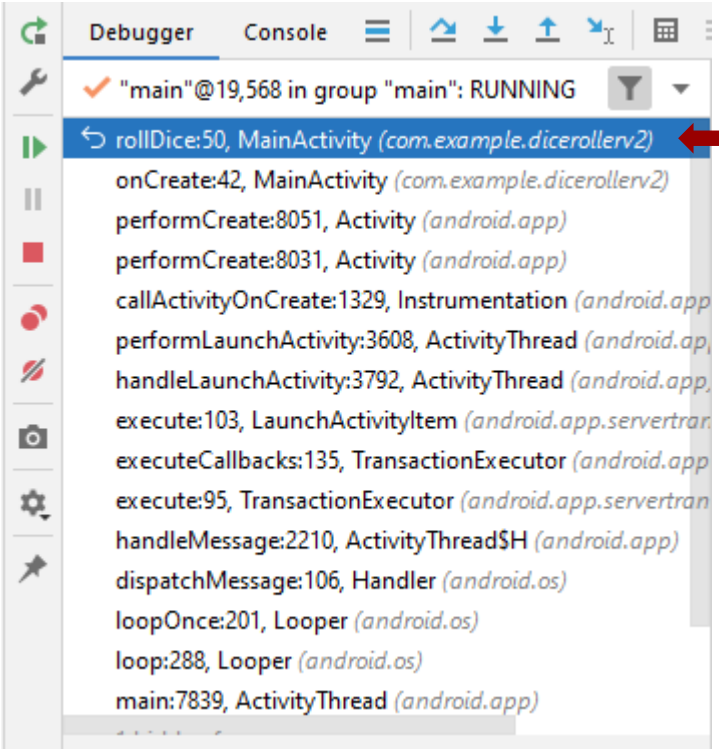


Crear puntos de interrupción condicionales

- En la ventana de propiedades o hacer clic-derecho en un punto de interrupción existente.
- Cualquier expresión en Java que devuelva un booleano.
- El autocompletado de código ayuda a escribir condiciones.

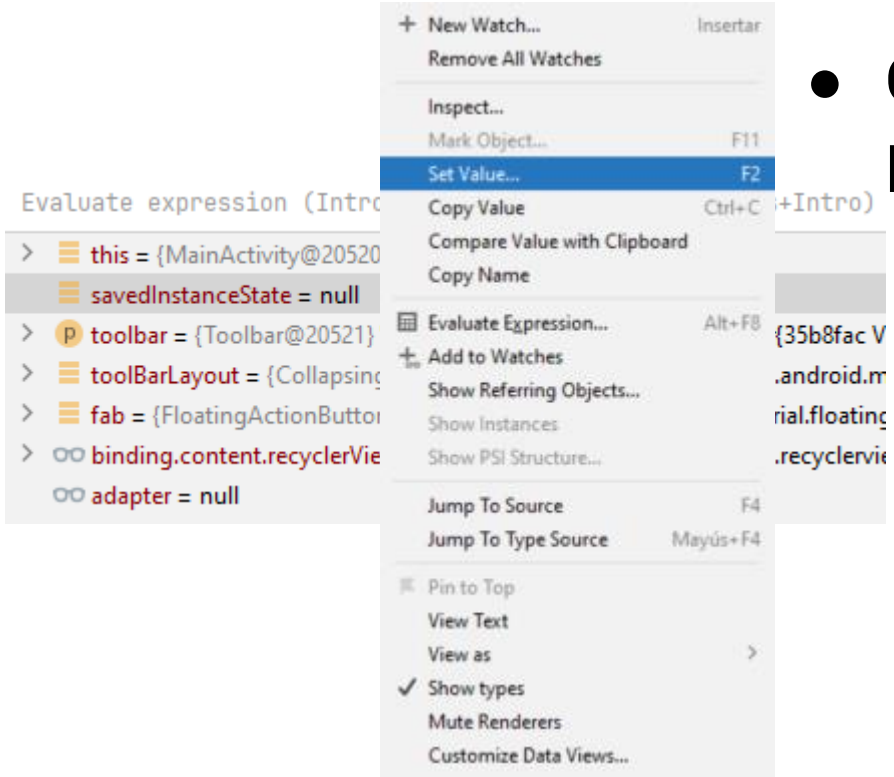


Inspeccionar marcos de pila (frames)



Marco de pila *(frame)* superior donde se ha parado la ejecución del código de la app.

Inspeccionar y editar variables



- **Clic-derecho** sobre una variable para ver el menú de opciones.

Navegar a través del código

Mostrar punto de ejecución

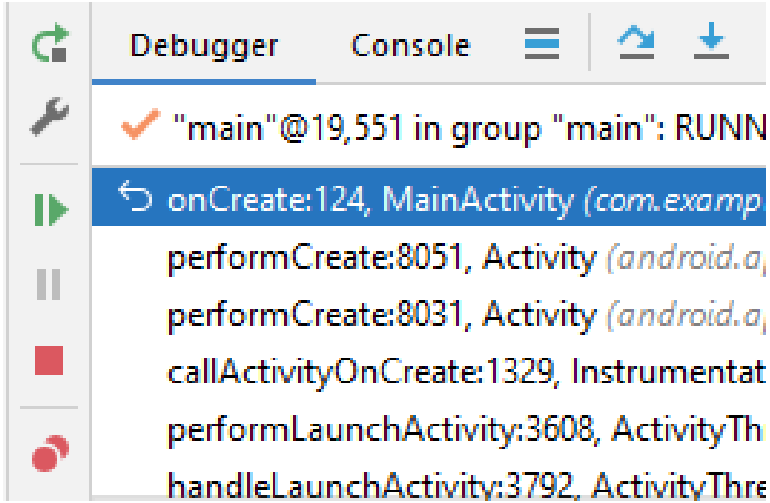
Para navegar por el código e ir examinando cómo se ejecuta.

Reanudar y pausar

Reanudar

Pausar

Inhabilitar todos los puntos de interrupción



Logging con Android Studio

Mecanismo de logging en una app

- Otra forma de poder depurar el sistema.
- Cuando la app se ejecuta, el panel **Logcat** muestra información.
- Añadir declaraciones de logging en la app para que se muestren en este panel.
- Establecer filtros en el panel **Logcat** para visualizar solamente lo que se considere importante.
- Buscar utilizando etiquetas.

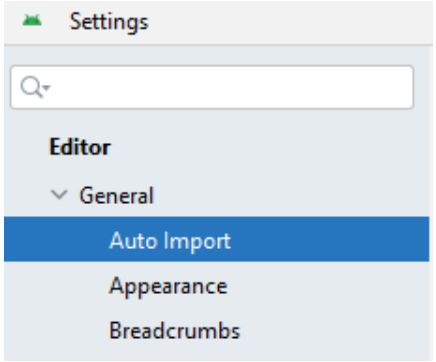
Niveles de Log

- **Verbose (v) –** Logs de detalle del funcionamiento de la app.
- **Debug (d) –** Logs de depuración.
- **Info (i) –** Logs de información.
- **Warning (w) –** Logs de advertencia, no supone error.
- **Error (e) –** Logs de comunicación de errores.

Declaración de Logging en una app

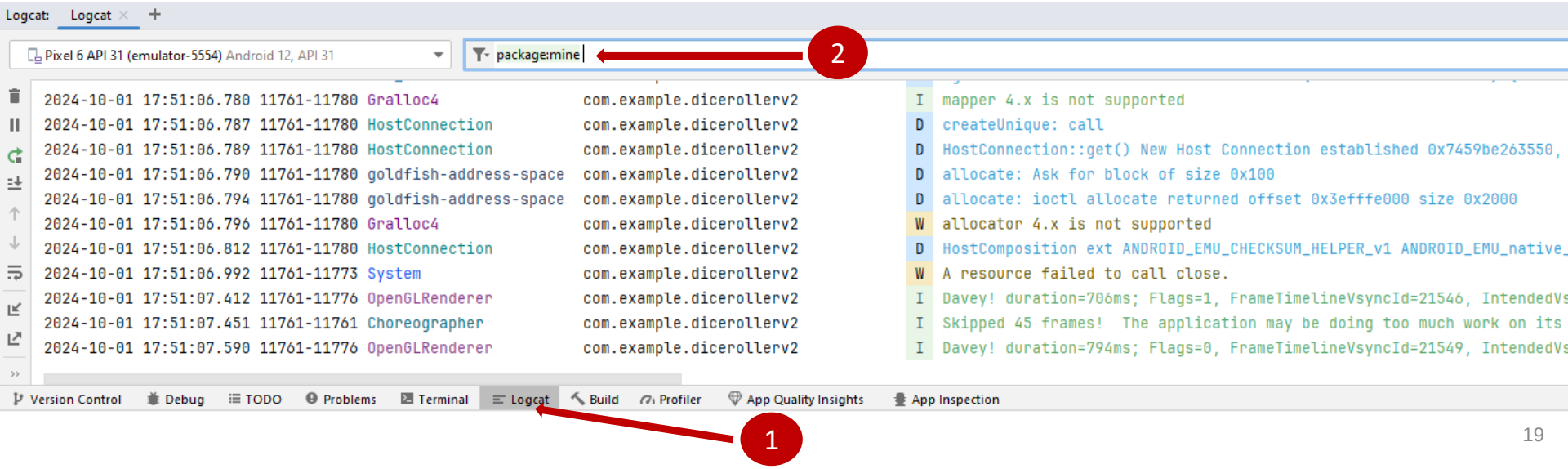
```
import android.util.Log;
...
// Aquí se utiliza el nombre de la propia
// clase como etiqueta.
private static final String TAG =
    MainActivity.class.getSimpleName();
...
// Mostrar el mensaje en el panel logacat
// Log.<log-level>(TAG, "<Message>");
Log.d("MainActivity", "Realizando la búsqueda…");
```

Control de importación automática de librerías: **File > Settings > Editor > General > Autoimport**

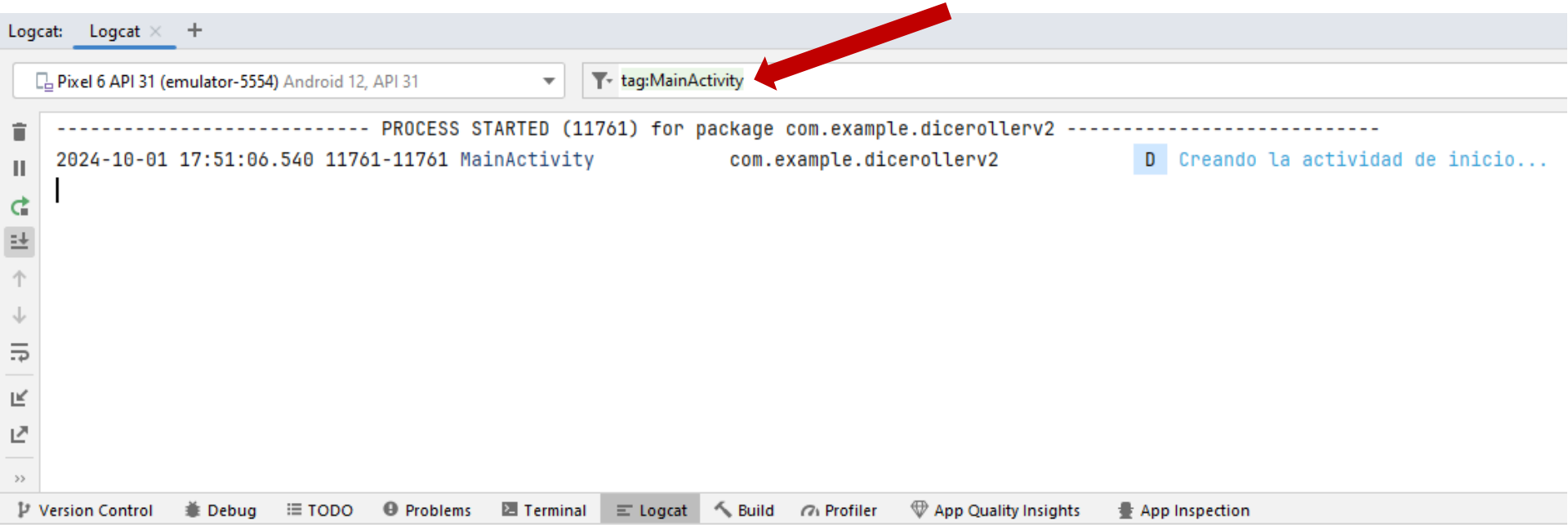


El panel Logcat (1/3)

1. Pestaña **Logcat** para mostrar el panel.
2. Menú para el [nivel de log](#) que se desee.



El panel Logcat (2/3)



El panel Logcat (3/3)