

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)



Menú: **Run > Debug 'app'**

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

Debug: **app** x

Debugger Console

1 "main"@19,551 in group "main": RUNNING

2 rollDice:54, MainActivity (com.example.dicerollerv2)

3 Evaluate expression (Intro) or add a watch (Ctrl+Mayús+Intro)

4 onCreate:42, MainActivity (com.example.dicerollerv2)

5

1

2

3

4

5

Switch frames from anywhere in the IDE with Ctrl+Alt+...

Establecer puntos de interrupción

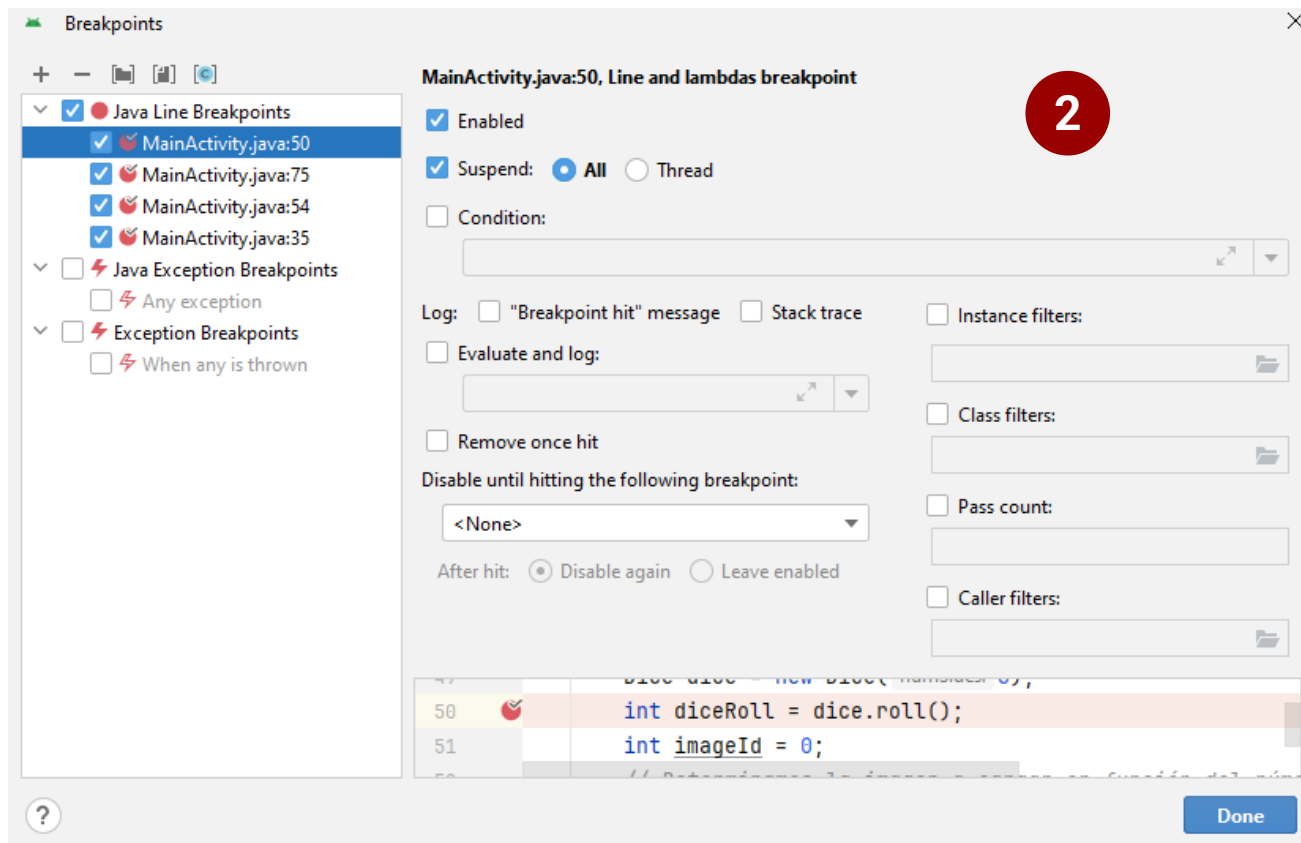
Hacer clic en el margen izquierdo al lado de la línea de código ejecutable

The screenshot displays the Android Studio interface. On the left, the 'Project' view shows the file structure of an Android application, including 'manifests', 'java', 'themes', 'xml', 'res', and 'Gradle Scripts'. The main editor shows the 'MainActivity.java' file. A red square highlights the left margin next to line 54, which contains the start of a switch statement. The code in the editor is as follows:

```
11 public class MainActivity extends AppCompatActivity {
47     */
48     private void rollDice() { 2 usages
49         Dice dice = new Dice( numSides: 6);  dice: MainActivity$Dice@20422
50         int diceRoll = dice.roll();  diceRoll: 5  dice: MainActivity$Dice@20422
51         int imageId = 0;  imageId: 0
52         // Determinamos la imagen a cargar en función del número
53         // devuelto.
54         switch (diceRoll){  diceRoll: 5
55             case 1:
56                 imageId = R.drawable.dice_1;
57                 break;
58             case 2:
59                 imageId = R.drawable.dice_2;
60                 break;
61             case 3:
62                 imageId = R.drawable.dice_3;
63                 break;
64             case 4:
65                 imageId = R.drawable.dice_4;
66                 break;
67             case 5:
68                 imageId = R.drawable.dice_5;
```

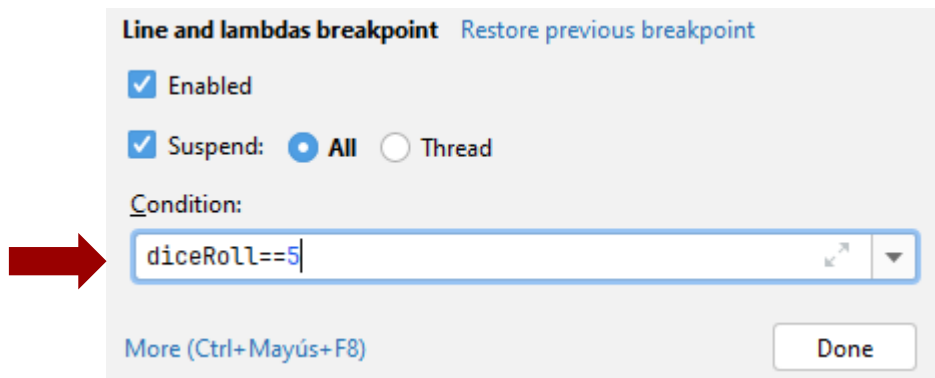
At the bottom, the 'Debugger' window is open, showing the 'main' thread running. The 'Call Stack' pane lists the sequence of method calls, with 'rollDice:54, MainActivity (com.example.dicerollerv2)' at the top. The 'Variables' pane shows the current state of variables: 'this' is 'Collecting data...', 'dice' is '(MainActivity\$Dice@20422)', 'diceRoll' is '5', and 'imageId' is '0'.

Editar las propiedades de puntos de interrupción

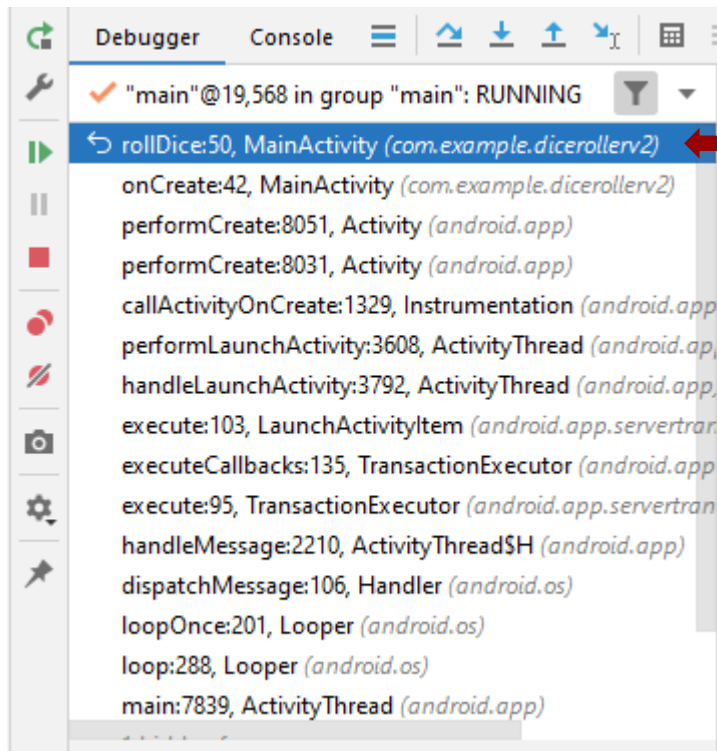


Crear puntos de interrupción condicionales

- 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.



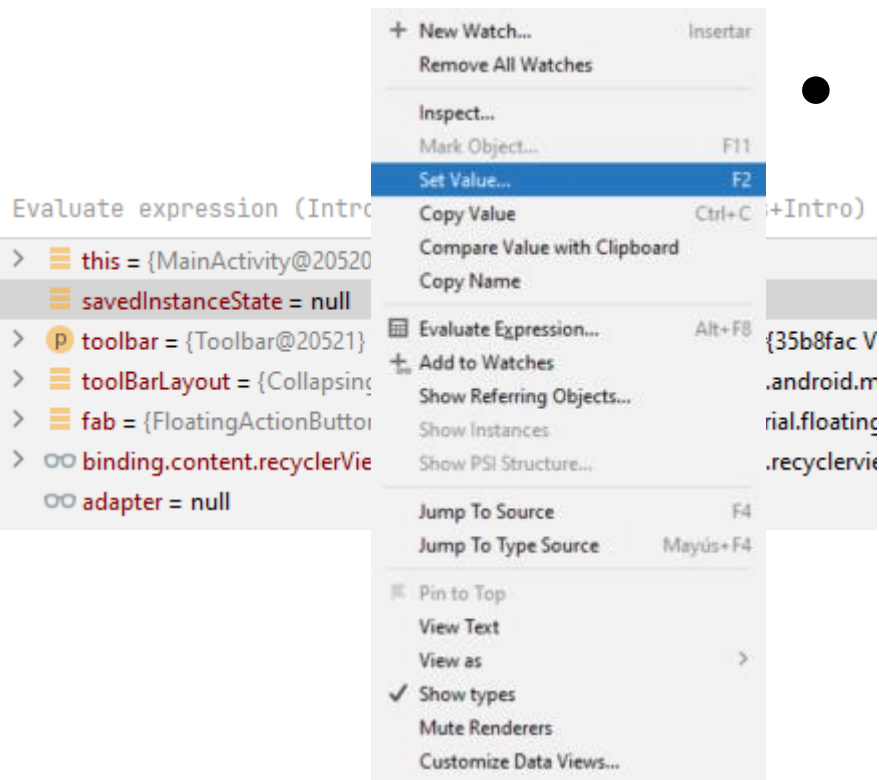
Inspeccionar marcos de pila (frames)



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.



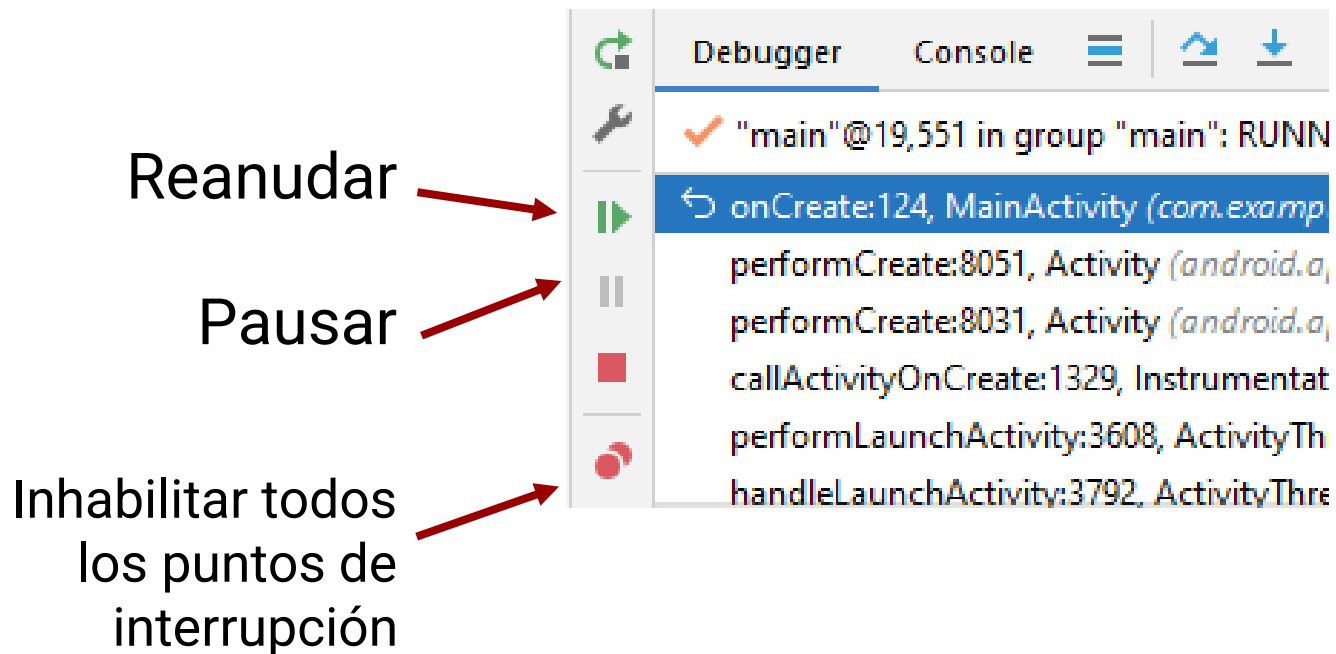
Navegar a través del código

Mostrar punto de ejecución



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

Reanudar y pausar



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.

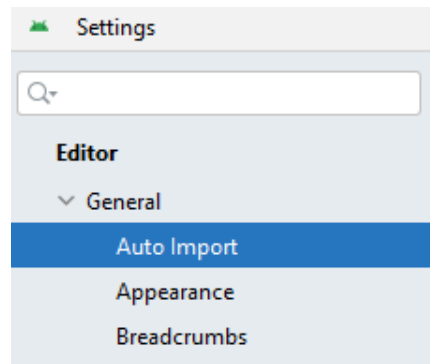
Niveles de Log

- **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

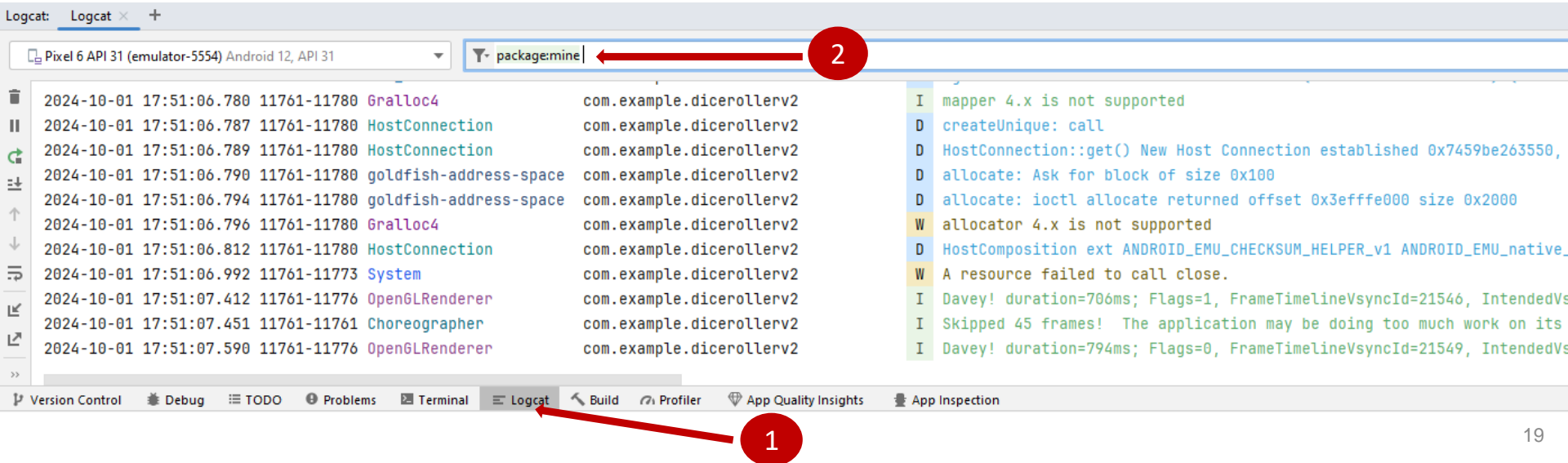
```
import android.util.Log;
...
// Aquí se utiliza el nombre de la propia
// clase como etiqueta.
private static final String TAG =
    MainActivity.class.getSimpleName();
...
// Mostrar el mensaje en el panel logcat
// Log.<log-level>(TAG, "<Message>");
Log.d("MainActivity", "Realizando la búsqueda...");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

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

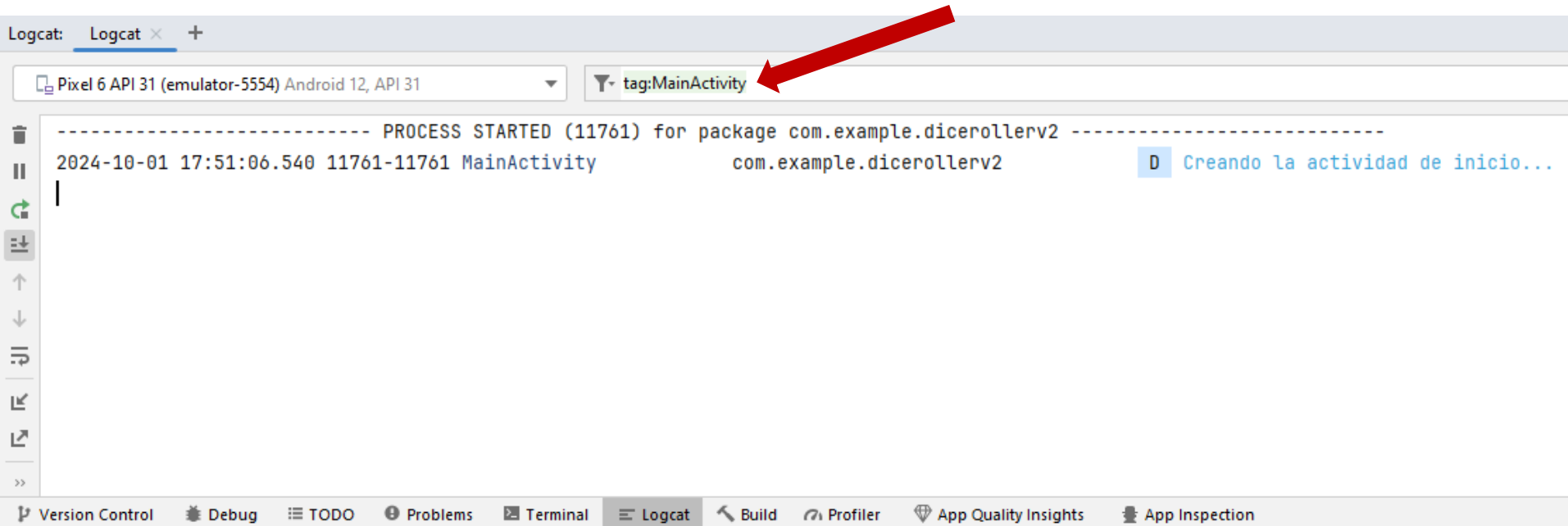


El panel Logcat (1/3)

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)

