

Android Developer Fundamentals V2

# Testing, debugging, and using support libraries

## Lesson 3



# 3.1 The Android Studio debugger

# Contents

- All code has bugs
- Android Studio logging
- Android Studio debugger
- Working with breakpoints
- Changing variables
- Stepping through code

# All Code Has Bugs

# Bugs

- Incorrect or unexpected result, wrong values
- Crashes, exceptions, freezes, memory leaks
- Causes
  - Human Design or Implementation Error > Fix your code
  - Software fault, but in libraries > Work around limitation
  - Hardware fault or limitation -> Make it work with what's available

Origin of the term "bug" (it's not what you think)

# Debugging

- Find and fix errors
- Correct unexpected and undesirable behavior
- Unit tests help identify bugs and prevent regression
- User testing helps identify interaction bugs

# Android Studio debugging tools

Android Studio has tools that help you

- identify problems
- find where in the source code the problem is created
- so that you can fix it

# Logging with Android Studio



# Add Log messages to your code

```
import android.util.Log;

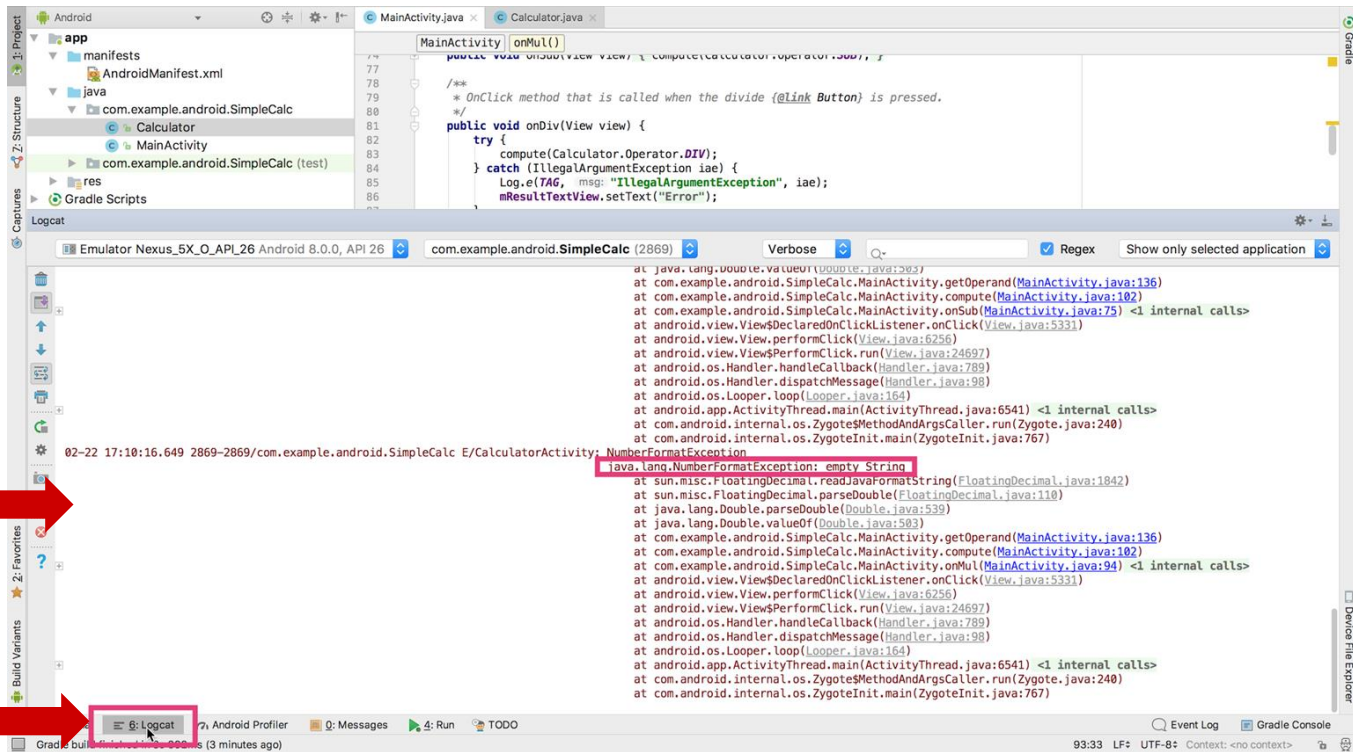
// Use class variable with class name as tag
private static final String TAG =
    MainActivity.class.getSimpleName();

// Show message in Logcat pane of Android Studio
// Log.<log-level>(TAG, "Message");
Log.d(TAG, "Hello World");
```

# Open Logcat pane

Logcat  
pane

Logcat  
tab



# Inspect logging messages

The screenshot shows the Android Studio IDE. On the left, the Project view shows the file structure of the 'HelloWorld' app. The main editor displays the `MainActivity.java` file. A red circle with the number '1' highlights the `Log.d("MainActivity", "Hello World");` line in the code. A red line extends from this line to the Logcat window at the bottom. The Logcat window shows a list of log messages, with a red circle with the number '2' highlighting the message: `09-12 14:28:07.971 4304 /com.example.android.helloworld D/MainActivity: Hello World`.

```
package com.example.android.helloworld;

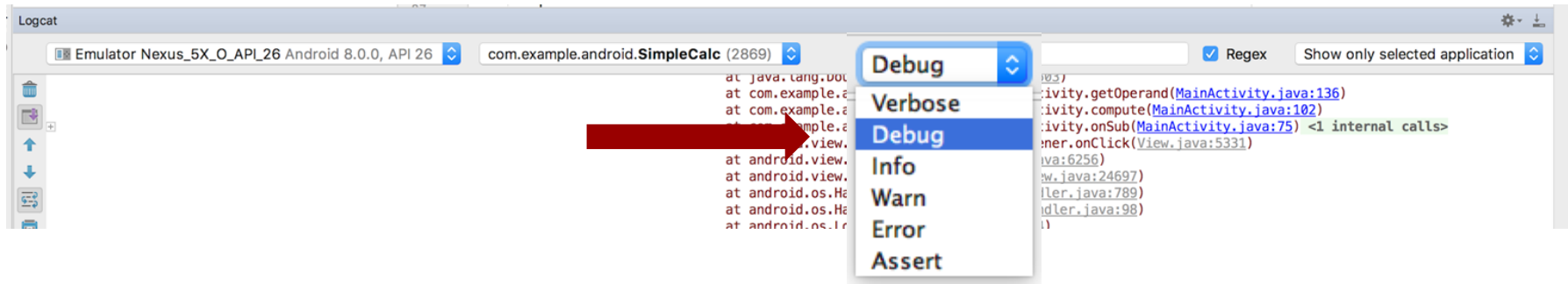
import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d("MainActivity", "Hello World");
    }
}
```

09-12 14:28:07.971 4304 /com.example.android.helloworld D/MainActivity: Hello World

# Choose visible logging level



Displays logs with levels at this level or higher

# Log Levels

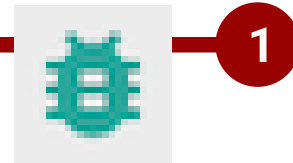
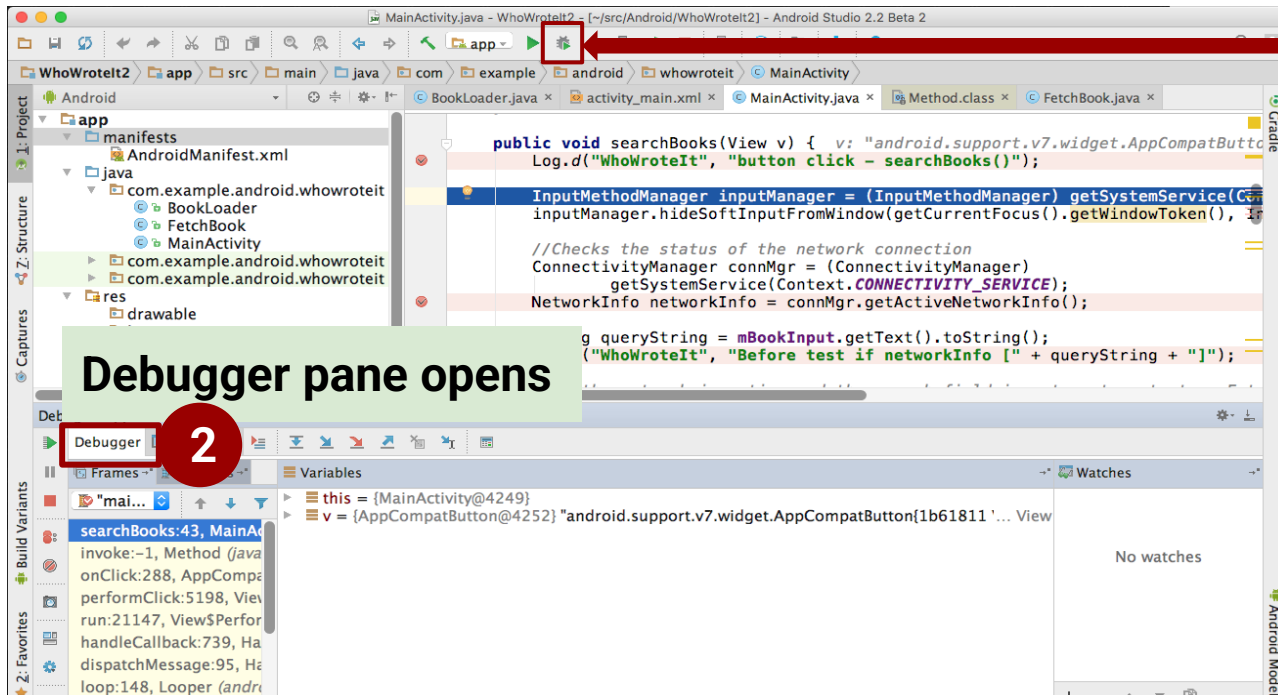
- **Verbose** - All verbose log statements and comprehensive system
- **Debug** - All debug logs, variable values, debugging notes
- **Info** - Status info, such as database connection
- **Warning** - Unexpected behavior, non-fatal issues
- **Error** - Serious error conditions, exceptions, crashes only

# Debugging with Android Studio

# What you can do

- Run in debug mode with attached debugger
- Set and configure breakpoints
- Halt execution at breakpoints
- Inspect execution stack frames and variable values
- Change variable values
- Step through code line by line
- Pause and resume a running program

# Run in debug mode

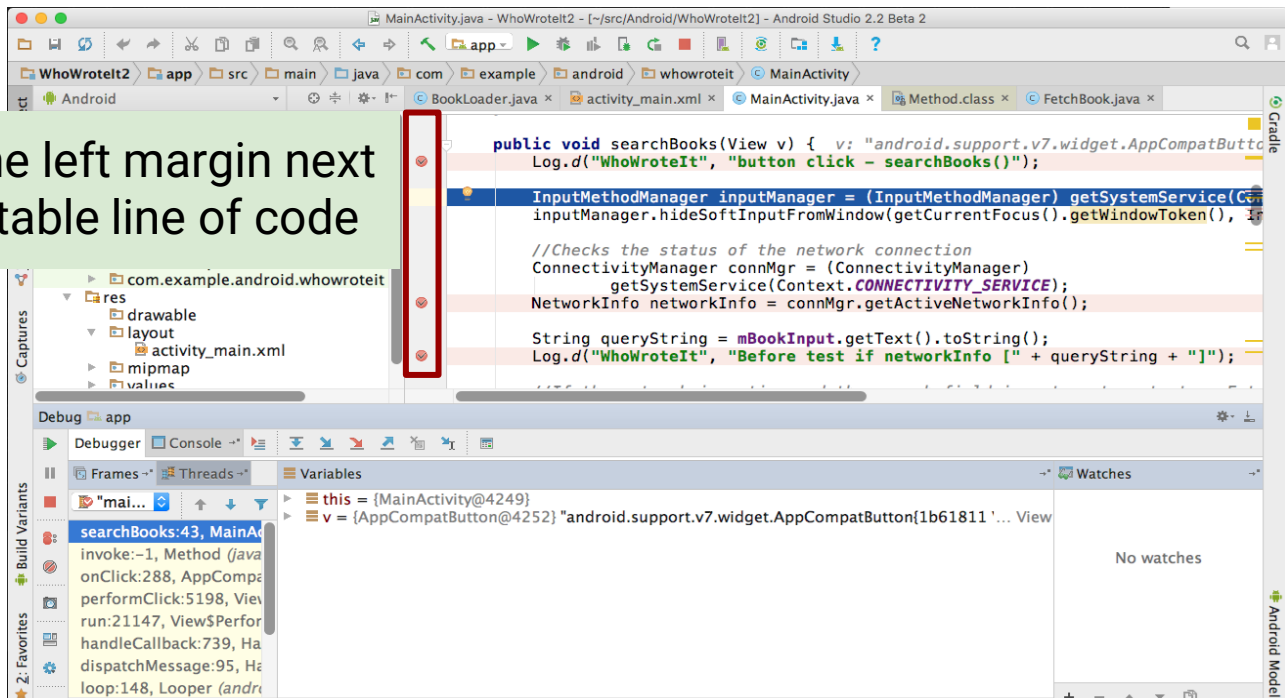


Menu:  
Run > Debug 'your app'



# Set breakpoints

Click in the left margin next to executable line of code



# Edit breakpoint properties

1

2

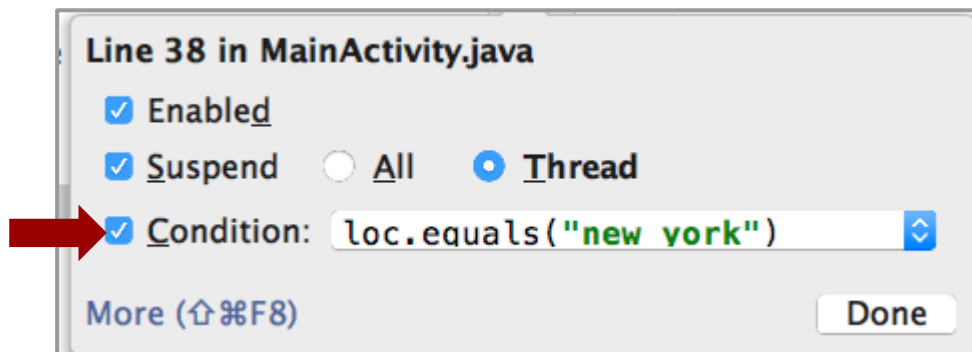
Line 47 in MainActivity.java

- ☒ Enabled
- ☒ Suspend ☐ All ☒ Thread
- ☐ Condition:
- ☐ Log message to console
- ☐ Log evaluated expression:
- ☐ Remove once hit
- Disabled until selected breakpoint is hit:
- After breakpoint was hit ☒ Disable again ☐ Leave enabled

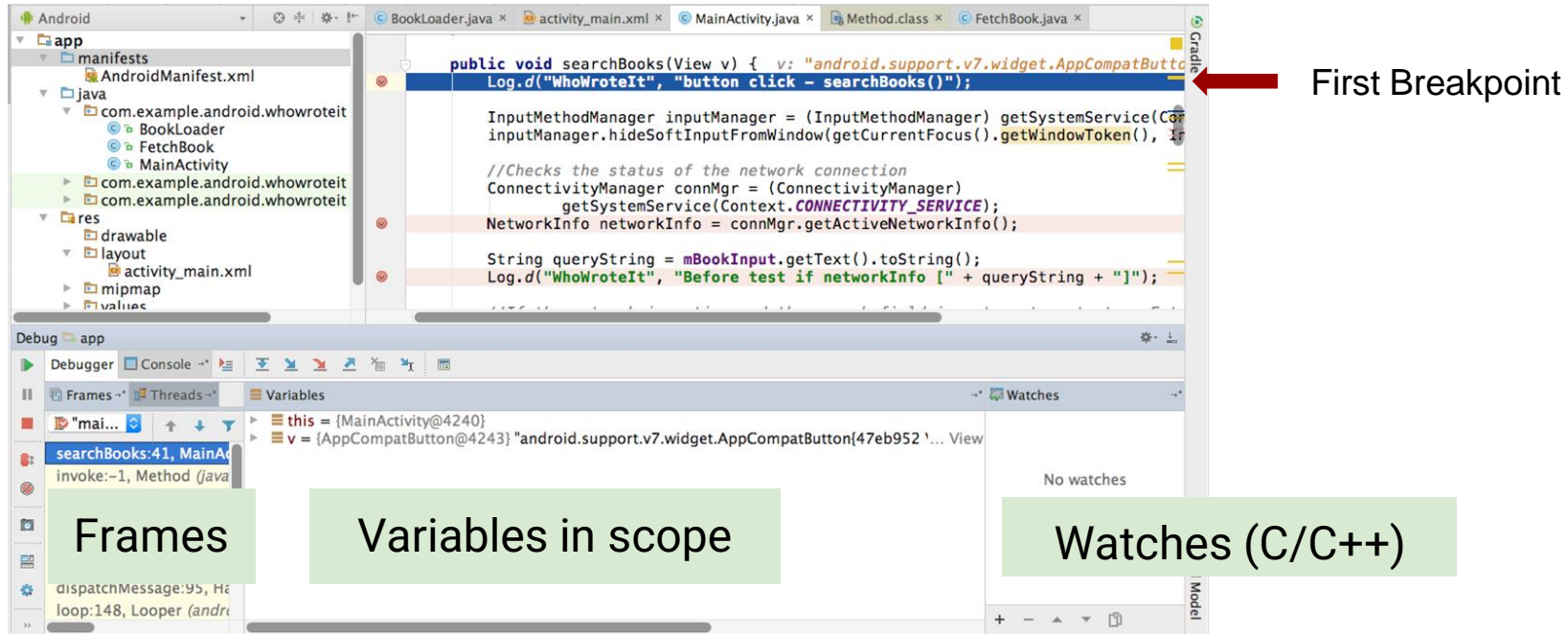
```
//Checks the status of the network connection
ConnectivityManager connMgr = (ConnectivityManager)
    getSystemService(Context.CONNECTIVITY_SERVICE);
```

# Make breakpoints conditional

- In properties dialog or right -click existing breakpoint
- Any Java expression that returns a boolean
- Code completion helps you write conditions



# Run until app stops at breakpoint



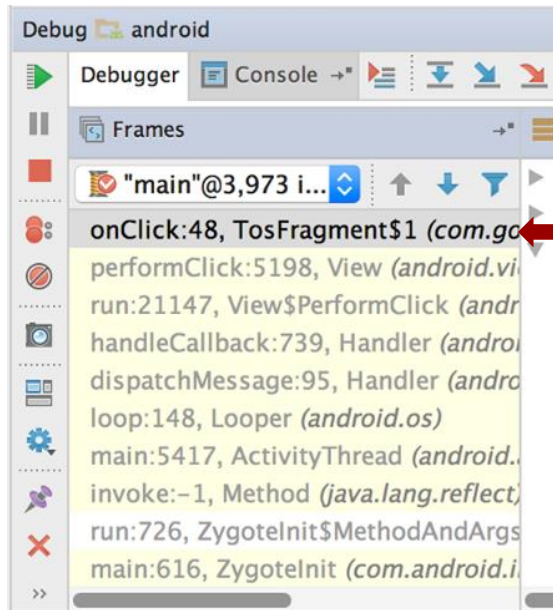
First Breakpoint

Frames

Variables in scope

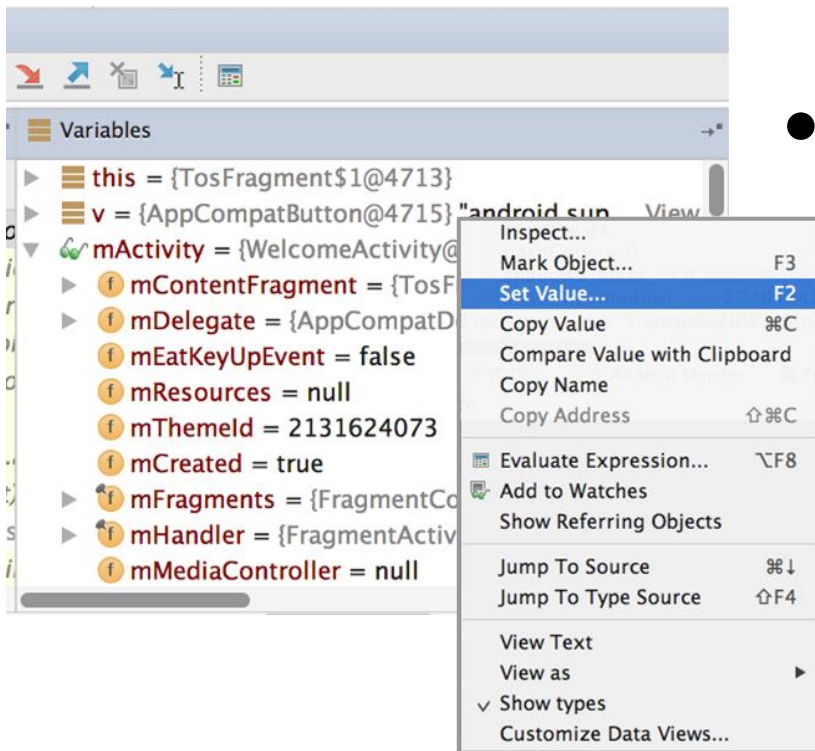
Watches (C/C++)

# Inspect frames



Top frame is where execution is halted in your code

# Inspect and edit variables



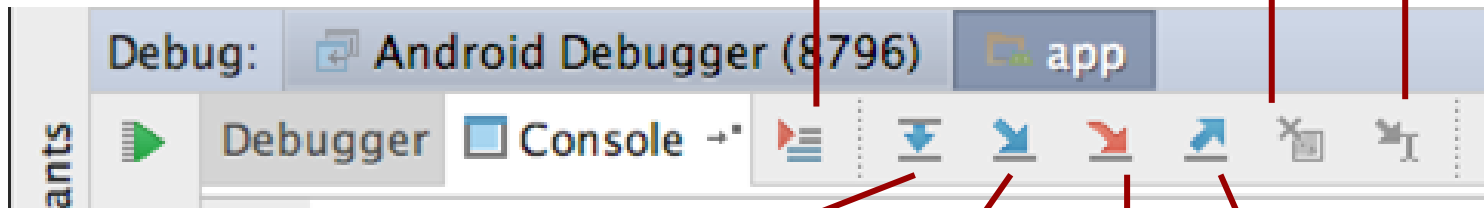
- Right-click on variable for menu

# Basic Stepping Commands

Step Over	F8	Step to the next line in current file
Step Into	F7	Step to the next executed line
Force Step Into	⇧F7	Step into a method in a class that you wouldn't normally step into, like a standard JDK class
Step Out	⇧F8	Step to first executed line after returning from current method
Run to Cursor	⇧F9	Run to the line where the cursor is in the file

# Stepping through code

Show execution point   Drop frame   Run to cursor



Step over

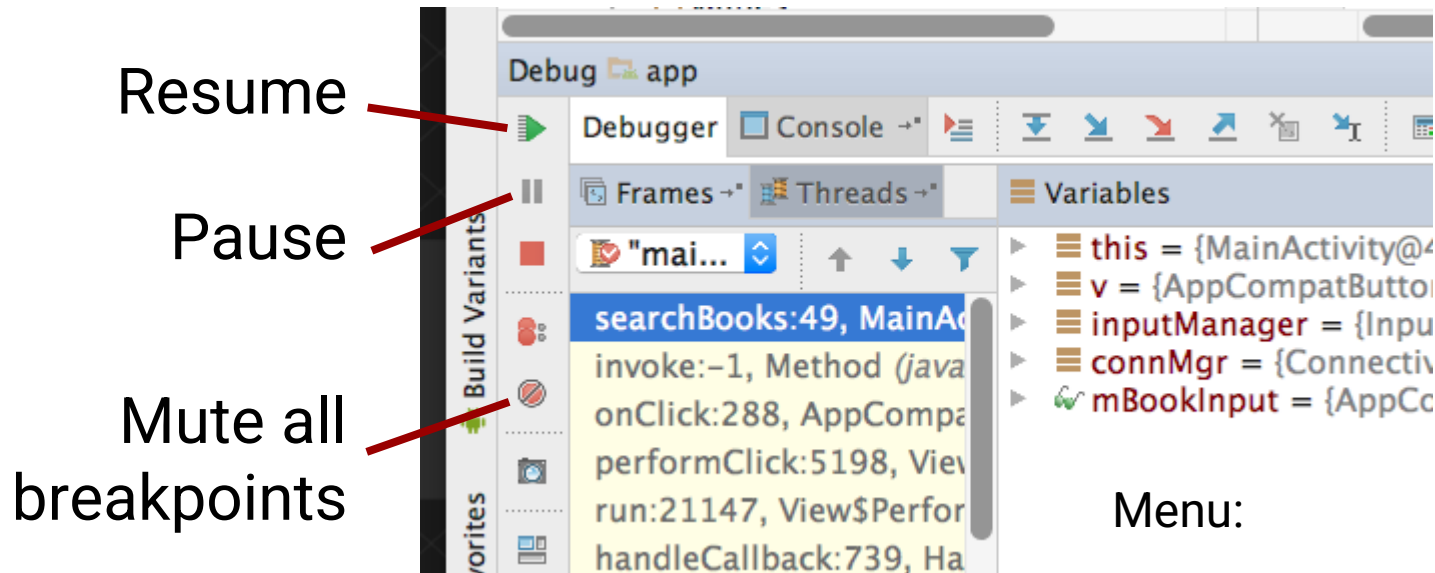
Step into

Step out

Force step into



# Resume and Pause



Menu:

**Run->Pause Program...**

**Run->Resume Program...**

# Learn more

- [Debug Your App](#) (Android Studio User Guide)
- [Debugging and Testing in Android Studio](#) (video)

# What's Next?

- Concept Chapter: [3.1 The Android Studio debugger](#)
- Practical: [3.1 The debugger](#)

# END