19CSI605 - Mobile Application Development

SRI KRISHNA INSTITUTIONS COIMBATORE

SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY
PERFECTION THROUGH PERFORMANCE
1998

Debugging

G. Pradeep, AP

The Android Studio Debugger

About Debugging

- Debugging is the process of finding and fixing errors (bugs) or unexpected behavior in your code.
- All code has bugs, from incorrect behavior in your app, to behavior that excessively consumes memory or network resources, to actual app freezing or crashing.

Bugs can result for many reasons:

- Errors in your design or implementation
- Android framework limitations (or bugs)
- Missing requirements or assumptions for how the app should work
- Device limitations (or bugs)

The Android Studio Debugger

Use the debugging, testing, and profiling capabilities in Android Studio to help you reproduce, find, and resolve all of these problems. Those capabilities include:

- The **Logcat** pane for log messages
- The **Debugger** pane for viewing frames, threads, and variables
- Debug mode for running apps with breakpoints
- Test frameworks such as JUnit or Espresso
- Dalvik Debug Monitor Server (DDMS), to track resource usage

The Android Studio Debugger

Running the debugger

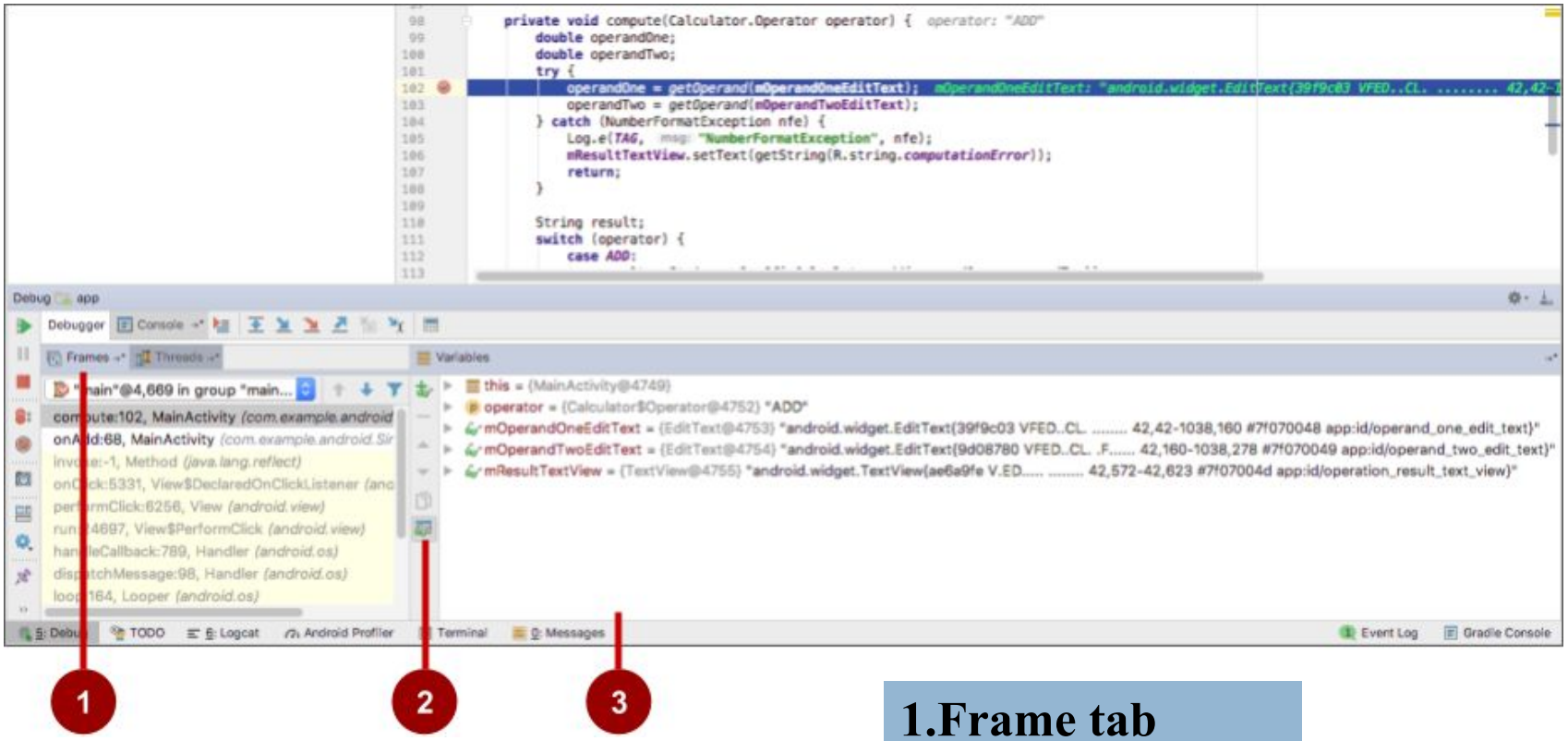
Running an app in debug mode is similar to running the app. You can either run an app in debug mode, or attach the debugger to an already-running app.

Run your app in debug mode

- To start debugging, click **Debug** in the toolbar. Android Studio builds an APK, signs it with a debug key, installs it on your selected device, then runs it and opens the **Debug** pane with the **Debugger** and **Console** tabs.

The Android Studio Debugger

Run your app in debug mode



1.Frame tab
2.Watches button
3.Variables pan

The Android Studio Debugger

Using Break Points

- Android Studio supports several types of breakpoints that trigger different debugging actions.
- The most common type is a breakpoint that pauses the execution of your app at a specified line of code.
- While paused, you can examine variables, evaluate expressions, then continue execution line by line to determine the causes of runtime errors.
- You can set a breakpoint on any executable line of code

The Android Studio Debugger

Add breakpoints

To add a breakpoint to a line in your code, use these steps:

1. Locate the line of code where you want to pause execution.
2. Click in the left gutter of the editor pane at that line, next to the line numbers. A red dot appears at that line, indicating a breakpoint. The red dot includes a check mark if the app is already running in debug mode.

As an alternative, you can choose **Run > Toggle Line Breakpoint** or press **Control-F8** (**Command-F8** on a Mac) to set or clear a breakpoint at a line.

The Android Studio Debugger

Add breakpoints

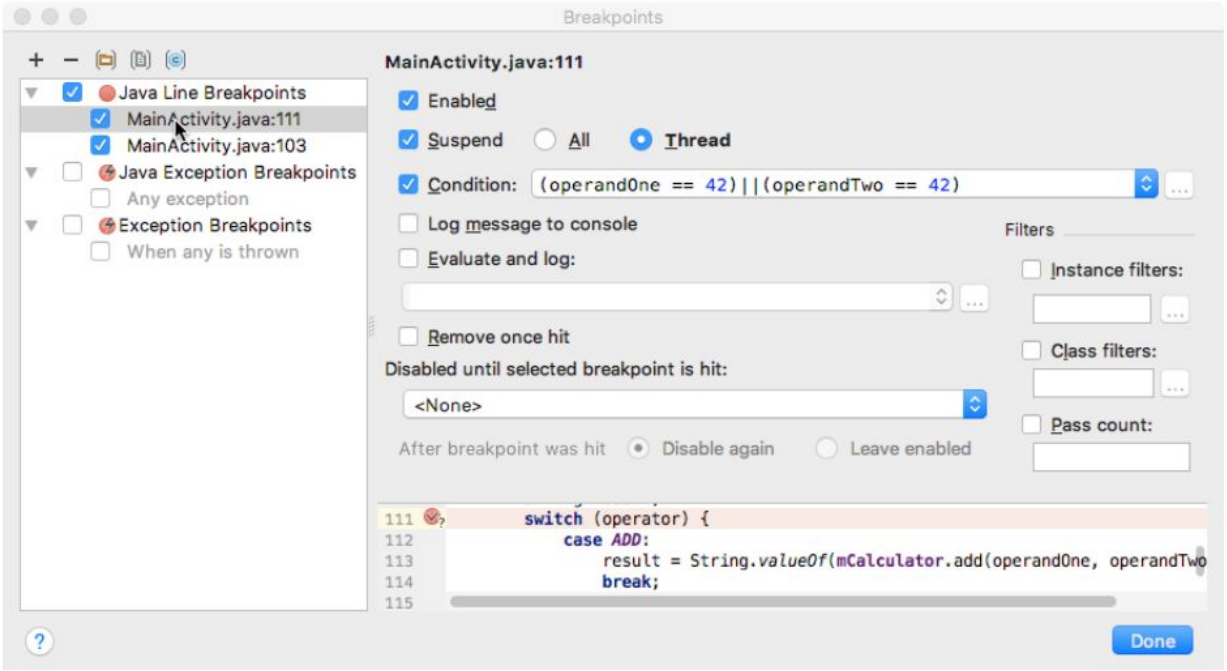
```
 98      private void compute(Calculator.Operator operator) {
 99          double operandOne;
100          double operandTwo;
101          try {
102              operandOne = getOperand(mOperandOneEditText);
103              operandTwo = getOperand(mOperandTwoEditText);
104          } catch (NumberFormatException nfe) {
105              Log.e(TAG,  msg: "NumberFormatException", nfe);
106              mResultTextView.setText(getString(R.string.computationError));
107              return;
108          }
109
110          String result;
111          switch (operator) {
```

If your app is already running, you don't need to update it to add the breakpoint.

The Android Studio Debugger

View and configure breakpoints

To view all the breakpoints you've set and configure breakpoint settings, click the **View Breakpoints** icon on the left edge of the **Debug** pane. The **Breakpoints** window appears.



The Android Studio Debugger

Disable (mute) all breakpoints

- Disabling a breakpoint enables you to temporarily "mute" that breakpoint without removing it from your code.
- If you remove a breakpoint altogether you also lose any conditions or other features you created for that breakpoint, so disabling it can be a better choice.

To mute all breakpoints, click the **Mute Breakpoints** icon. Click the icon again to enable (unmute) all breakpoints.

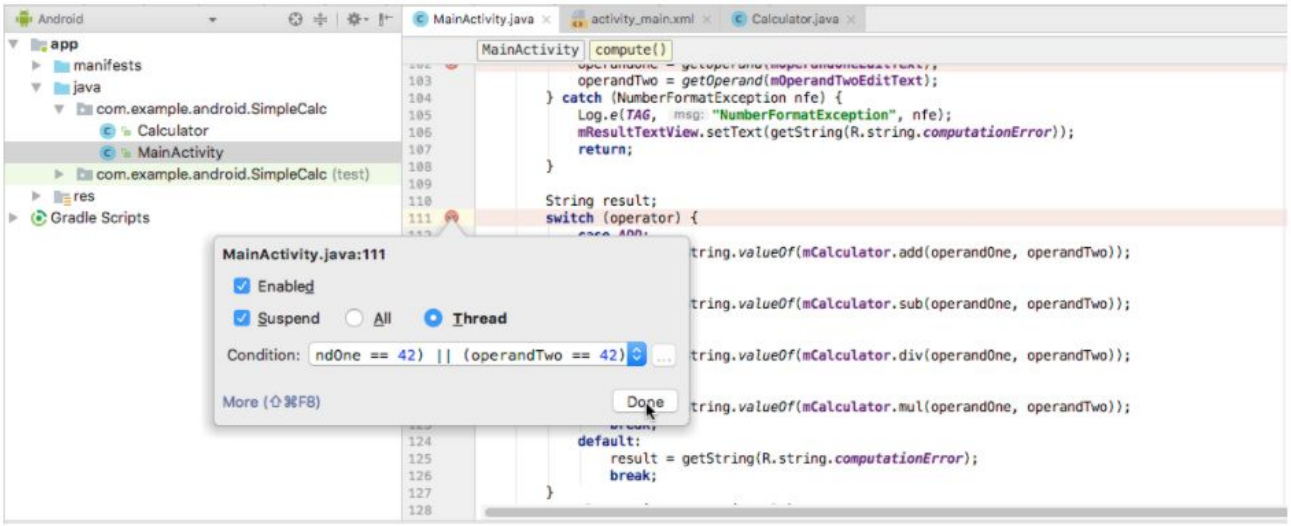
The Android Studio Debugger

Use conditional breakpoints

- Conditional breakpoints are breakpoints that only stop execution of your app if the test in the condition is true.

To define a test for a conditional breakpoint, use these steps:

- **Right-click** (or **Control-click**) a breakpoint, and enter a test in the **Condition** field.



The Android Studio Debugger

Stepping through code

- After your app's execution has stopped because a breakpoint has been reached, you can execute your code from that point one line at a time with the **Step Over**, **Step Into**, and **Step Out** functions

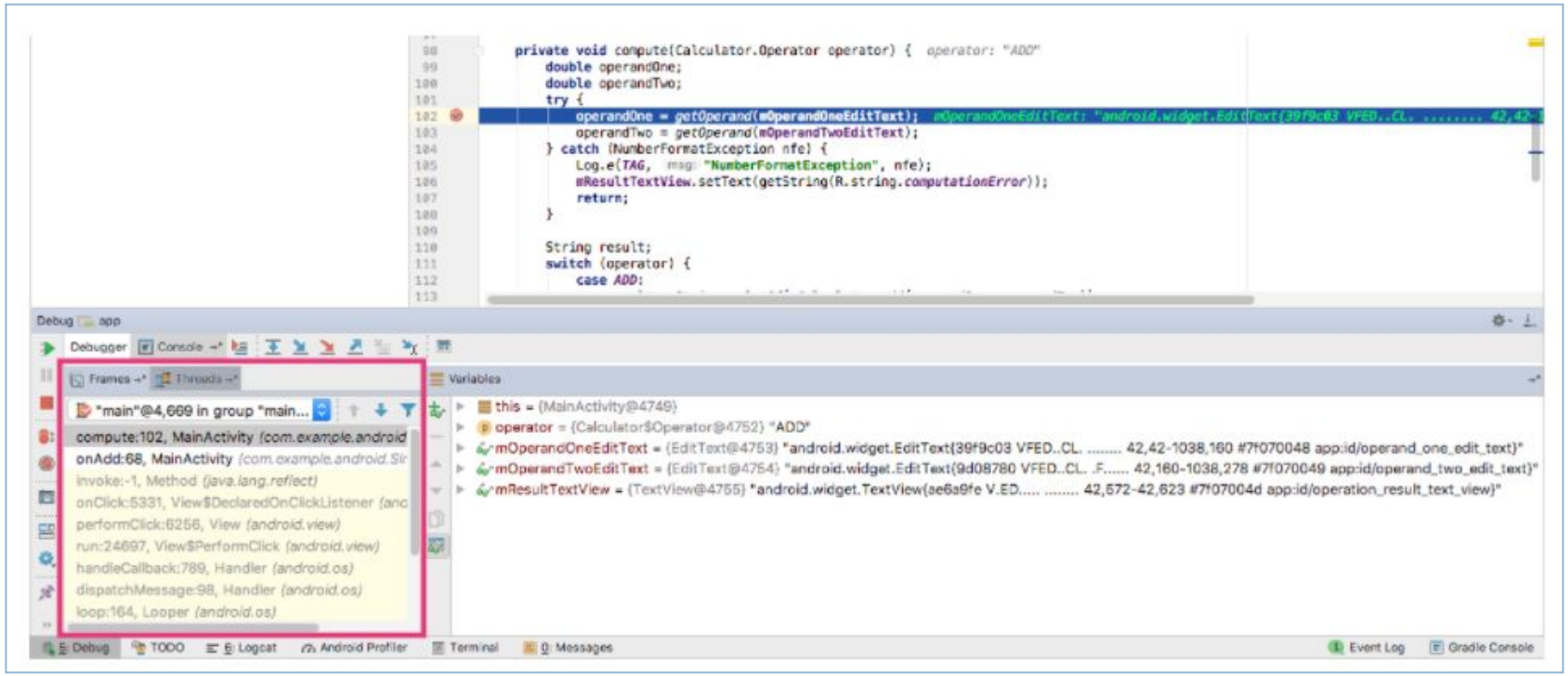
To use any of the step functions:

1. Begin debugging your app. Pause the execution of your app with a breakpoint.
2. Click the **Step Over** icon , select **Run > Step Over,** or press **F8**. (next)
3. Click the **Step Into** icon , select **Run > Step Into,** or press **F7**. **Step Into** jumps into the execution of a method call on the current line
4. Click the **Step Out** icon , select **Run > Step Out,** or press **Shift-F8**. **Step Out** finishes executing the current method and returns to the point. i.e place
5. To resume normal execution of the app, select **Run > Resume Program** or click the Resume icon.

The Android Studio Debugger

Viewing execution stack frames

- The **Frames** pane of the **Debug** pane allows you to inspect the execution stack and the specific frame that caused the current breakpoint to be reached



The Android Studio Debugger

Inspecting and modifying variables

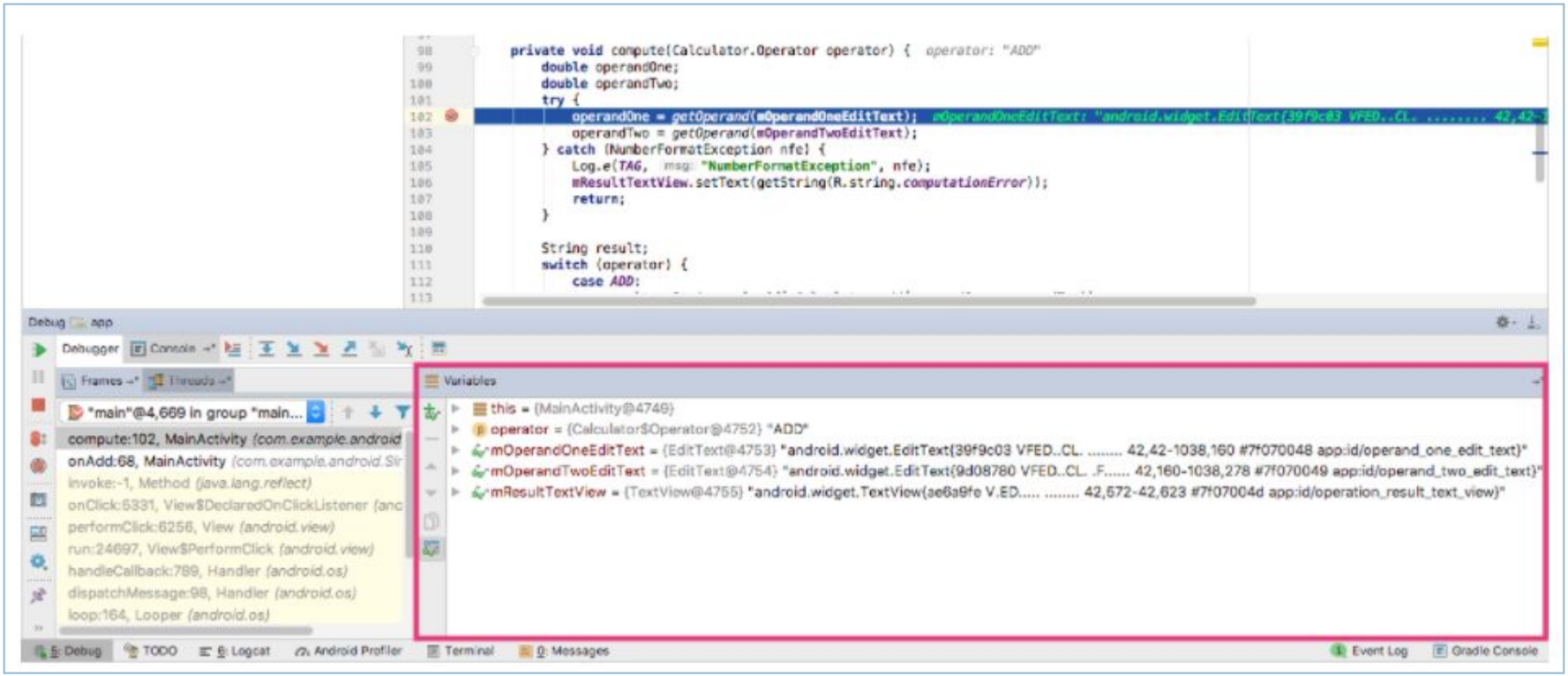
- The **Variables** pane of the **Debugger** pane allows you to inspect the variables available at the current stack frame when the system stops your app on a breakpoint.
- Variables that hold objects or collections such as arrays can be expanded to view their components.
- The **Variables** pane also allows you to evaluate expressions on the fly using static methods or variables available within the selected frame

To modify variables in your app as it runs:

- **Right-click** (or **Control-click**) any variable in the **Variables** pane, and select **Set Value**. You can also press **F2**.
- Enter a new value for the variable, and press **Return**

The Android Studio Debugger

Inspecting and modifying variables



The Android Studio Debugger

Setting watches

- The **Watches** pane provides similar functionality to he **Variables** pane except that expressions added to the **Watches** pane persist between debugging sessions.
- Add watches for variables and fields that you access frequently or that provide state that is helpful for the current debugging session.

To use watches:

1. Begin debugging your app.
2. Click the **Show Watches** icon. The **Watches** pane appears next to the **Variables** pane.
3. In the **Watches** pane, click the plus (+) button. In the text box that appears, type the name of the variable or expression you want to watch and then press **Enter**. {Use – for removal}

The Android Studio Debugger

Evaluating expressions

- Use **Evaluate Expression** to explore the state of variables and objects in your app, including calling methods on those objects.

To evaluate an expression:

1. Click the **Evaluate Expression** icon , or select **Run > Evaluate Expression**.
2. Enter any Java expression into the top field of the **Evaluate Code Fragment** window, and click **Evaluate**

THANK YOU