Advanced debugger features

Line breakpoints are useful, but there are several other places where it can be helpful to stop execution:

- When the constructor is invoked.
- When a particular method is invoked. Including one that's been overridden or one with incorrect or missing source code.
- When an exception of a specific type is thrown.
- The next line of the current method, when program execution has already been stopped.
- The first line of the method that's going to be invoked and added to the call stack next.
- The next line of the caller method after the currently executing method is popped from the call stack.

Constructor and method breakpoints

To set a method breakpoint, click in the gutter against the method declaration and you will see a diamond breakpoint indicator:

To set a **constructor breakpoint**, click in the gutter against the constructor declaration or the class declaration if the class does not have an explicit constructor.

You will see a square instead of a circle, and the execution will stop right inside the constructor, which is called <init> in the bytecode:

You can set a breakpoint on a method in a similar way. When debugging compiled code, you can see decompiled classes, but there can be mismatch between line number information and the sources.

Also, when debugging Android SDK, the source code in the IDE might be a different version from the Android version running on the device. So, line breakpoints would refer to the wrong lines, but **method breakpoints** will still work.

Exception breakpoints

When an application crash occurs, you can see its stack trace, but that's not always enough. Walking through stack frames and observing application state just before the crash could be very helpful.
Pressing *Run → View Breakpoints...* enables you to see all the current breakpoints and to set **exception breakpoints**:

Click the *Java Exception Breakpoints* option. Having done this, you can choose any of the exception types available in the current project and configure

whichever details you want. For example, you can opt to stop on caught exceptions, uncaught exceptions, or both.

Done! Now, execution will stop when an exception that meets the specified conditions is thrown.

Stepping

There are also three tools highlighted in the above screenshot:

- *Step Over*
- *Step Into*
- *Step Out*

Step Over will resume execution to the next line. This can be useful if you want to examine the execution of your code line by line.

Step Into will step into the next method called by the current method. There's also *Force Step Into (*which has the same icon as *Step Into* but with a red arrow instead of a blue one). This tool makes it easy to step into library code instead of staying at the level of your own code.

Force Step Into will work even without source code, and for runtime-generated classes without bytecode available. In this case, you won't see the code being executed, but step frames will still be visible, giving you an idea about what that method does.

Step Out will suspend execution after the current method ends and return you to the context of the caller method

In addition, you can use *Run to Cursor*. If you hover over a line number, the line will be highlighted. And if you click it, the program will resume until it reaches the clicked line and then be suspended again. This means that instead of adding a breakpoint and resuming, you can simply click the desired line.

- **Breakpoint**:
  - A breakpoint is a point in the code where the execution of the program will temporarily halt, allowing the developer to inspect the program's state, variables, and execution flow.
  - Breakpoints can be set at various locations in the code, such as specific lines, methods, or even conditions.
  - When the program execution reaches a breakpoint during debugging, it pauses, and the developer can step through the code or inspect the state of the program.
- **Line Breakpoint**:
  - A line breakpoint, as the name implies, is a breakpoint set at a

specific line of code.
    - It is the most common type of breakpoint and is used to pause the program's execution when it reaches that particular line.
    - Line breakpoints are set by clicking on the gutter area (to the left of the code) in an Integrated Development Environment (IDE) or using a debugging tool.

In summary, while both terms generally refer to the same concept of pausing program execution at a specific location, "line breakpoint" specifically refers to a breakpoint set at a particular line of code, while "breakpoint" is a more general term that encompasses breakpoints set at various locations in the code.

Exploring library code

Find out which method is used to implement
both removeAll and retainAll inside ArrayList.
Use Force Step into.

Step Into
Step Out
Step Over
Resume

inside a function which is going to be invoked

after the current function returns

on the next line

on the next breakpoint