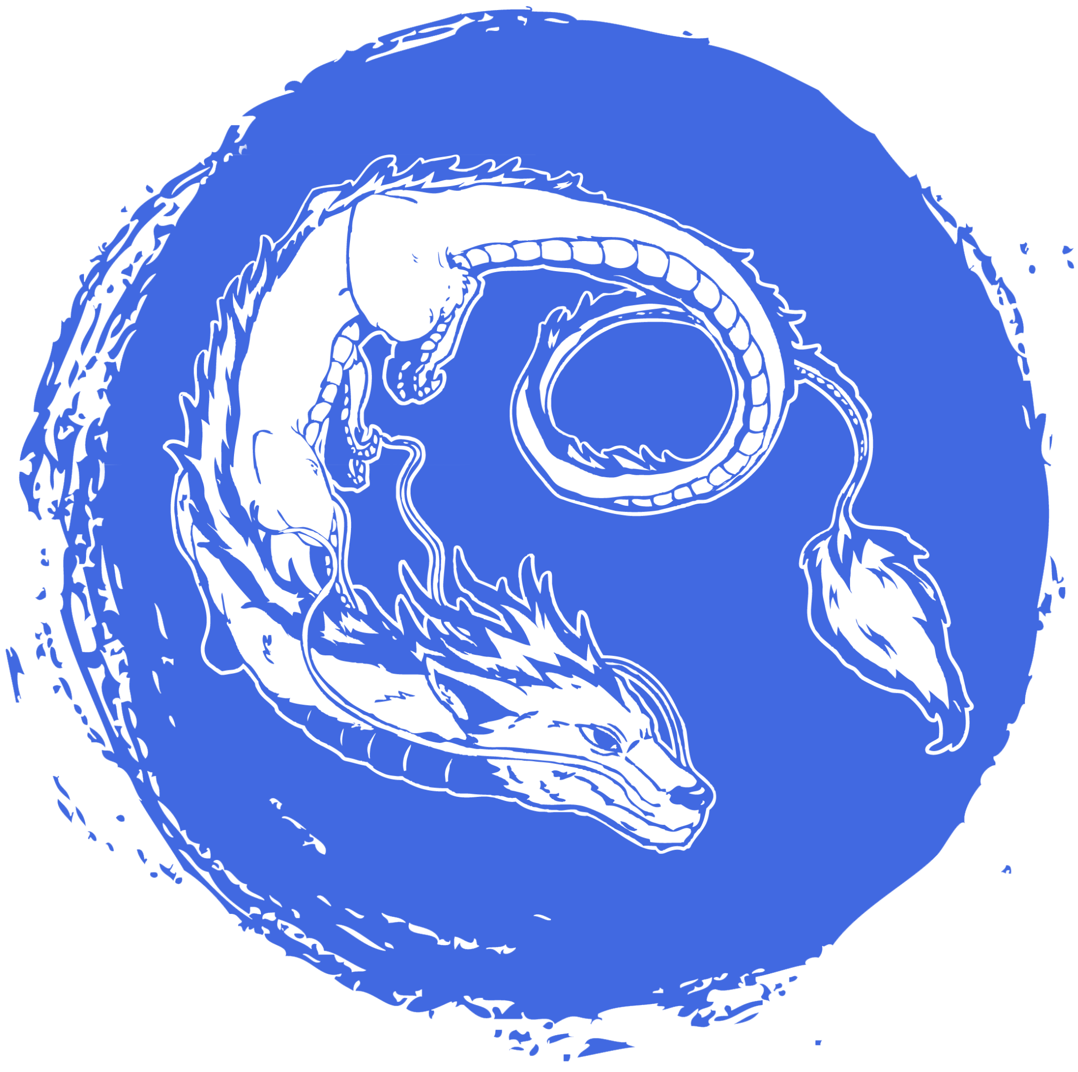
Java Workshop — Debugger

version **#v1.1.0**



Yet Another Kind of Assistants 2022 <yaka@tickets.assistants.epita.fr>

Copyright

Contents

*https://intra.assistants.epita.fr

Be careful!

The below keyboard shortcuts can vary consequently from one OS to another. Moreover, all these operations can be triggered through the universal search, using double `shift`.

1 Breakpoints

- `Ctrl+F8` or clicking the left part of your line after the line number: add or remove a breakpoint on a line of your code;
- `Ctrl+Alt+Shift+F8`: put a temporary breakpoint that will delete itself when triggered once.

When hitting a breakpoint, the execution of the whole JVM stops, and you can get information about the state of your program (call stack, local variables...).

- `F2` or right-clicking on the variable and selecting `Set Value...`: **modify local variables** during the execution.

2 Commands

- `F9`: resume execution;
- **Step over** (`F8`): same behaviour as `gdb`'s `next`, executes the current line and doesn't step into method calls, unless there is another breakpoint in it;
- **Force step over** (`Alt+Shift+F8`): eventual breakpoints located in the methods' calls will be ignored;
- **Step into** (`F7`): same behaviour as `gdb`'s `step`, steps into another method if the current executed line is a method call;
- **Smart step into** (`Shift+F7`): when several methods are called on the same line, lets you choose which one to actually step into;
- **Force step into** (`Alt+Shift+F7`): steps into a method even if it is a standard class and/or you specified in your preferences that you didn't want to step into it;
- **Step out** (`Shift+F8`): runs the whole current method, and puts next step the line after its call;
- **Run to cursor** (`Alt+F9`): runs the program and stop it when it reaches the line where your cursor is;
- **Force run to cursor** (`Ctrl+Alt+F9`): ignores every breakpoint encountered before reaching the line your cursor is hovering.

In your IDE settings, you can set some options related to the debugger, especially in `Settings > Build, Execution, Deployment > Debugger > Stepping` you can set options to tell your debugger not to step into some classes.

3 Expressions evaluation

- `Alt+F8`: opens the evaluation window. It allows you to evaluate expressions or execute parts of arbitrary code in the current context.

By default, it will suggest the evaluation of variables in the current context, but you can write any one line expression. You can click the arrows to switch expression mode (limited to one line) or the `Code fragment` mode, where you can write a full part of code. If the expression has a value, it will be displayed in the `Result` section.

You can call other methods in your expression and code fragment. However, if you are in expression mode, all breakpoints will be ignored. In `Code fragment`, the debugger can stop to other breakpoints, but in that state, you can't reevaluate another expression.

The debugger also allows you to see the value of expressions in your code, by selecting them and pressing `Ctrl+W` to make a tooltip appear, or `Ctrl+Alt+8` to make the full evaluation window appear.

4 Watches

You can create a **watch** to evaluate some variables or expressions multiple times throughout your debugging sessions.

- + button in the watches panel or right-clicking a variable in the local stack frame: add a watch.

At every new step in your program, the watched expressions will be reevaluated and displayed in the watches panel (which can be toggled with the icon with googles and a green plus in the variable panel). Watched expressions are attached to your project, and will survive from a debugging session to another. However, this is extremely slow and you should use it only in desperate situations.

5 Breakpoint types

Besides the line breakpoints, there are other kinds of breakpoints:

- **Method breakpoints**: breaks when entering and exiting the method the breakpoint is put on. Put a breakpoint on the prototype declaration line of your method or in `Run > Toggle Method Breakpoint`;
- **Exception breakpoints**: breaks when an exception of a specified type is thrown. `Ctrl+Shift+F8` to open the breakpoint dialog, press the + button, select `Java Exception Breakpoint` and choose the target exception class;
- **Field watchpoints**: breaks every time the field targeted is read or written. To create a field watchpoint, put a breakpoint on the field declaration.

6 Breakpoint configurations

By pressing `Ctrl+Shift+F8`, or right clicking a breakpoint, you have the ability to tune some useful parameters affecting the behaviour of a breakpoint. The following options can be especially helpful:

- `Enabled`: allows you to disable temporarily a breakpoint;
- `Suspend`: whether or not the breakpoint should suspend the execution of the program if hit. You can also specify if it should suspend all the VM (`All`), or only the thread that ran into it (`Thread`). This is especially helpful when debugging multithreaded applications;
- `Condition`: you can specify an expression in this field. If it evaluates to `true`, the action of the breakpoint will be executed, else it will be skipped;
- `Evaluate and log`: evaluates the given expression, and logs its result;
- `Pass count`: effectively executes the breakpoint action when it has been hit exactly the specified number of times.

For a method breakpoint, it is possible to specify that we want to break only at enter or exit of the method. Similarly, it is possible for a field watchpoint to specify if we want to break only when the field is read or written.

Don't be afraid, I just wanna help you.