Java Application Debugging

Fsoft Academy

Agenda

- What is debugging?
- Breakpoints
- Starting the Debugger
- Debug Perspective
- Controlling the program execution
- Evaluating variables in the debugger
- Changing variable assignments in the debugger

Overview

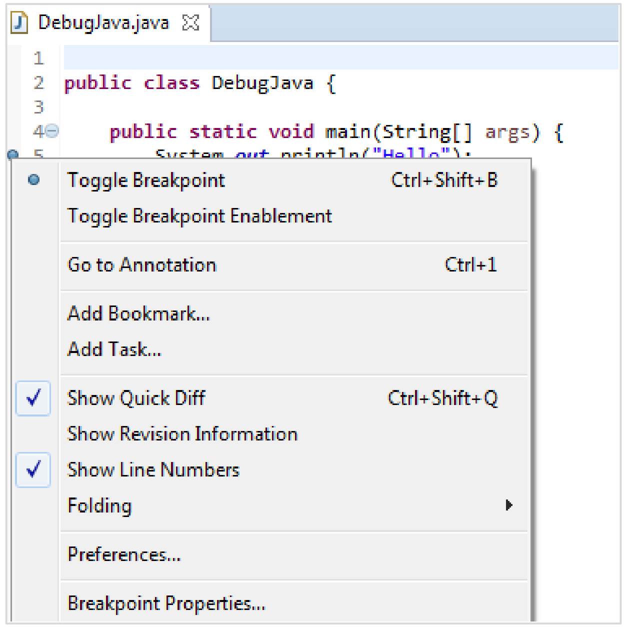
Debugging allows you to *run a program interactively while watching the source code* and the variables during the execution.

- A **breakpoint** in the source code specifies *where the execution of the program should stop during debugging*. Once the program is stopped you can investigate variables, change their content, etc.
- To stop the execution, if a field is read or modified, you can specify watch-points.
- **Breakpoints** and **watch-points** are sometimes called stop points.

Breakpoint

A **breakpoint** is a debugging tool used in software development *to pause the execution* of a program at a specific point, allowing you to inspect the program's state and behavior at that moment.

- Breakpoints are invaluable for finding and fixing issues in your code



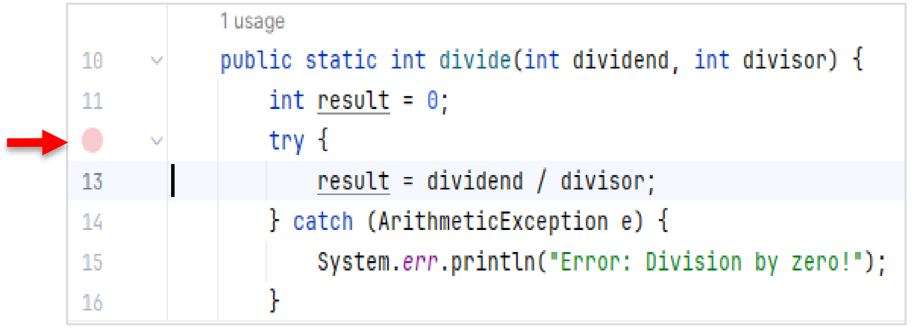
DebugJava.java

```
public class DebugJava {

    public static void main(String[] args) {
        System.out.println("Hello");
```

Toggle Breakpoint Ctrl+Shift+B
Toggle Breakpoint Enablement
Go to Annotation Ctrl+1
Add Bookmark...
Add Task...
Show Quick Diff Ctrl+Shift+Q
Show Revision Information
Show Line Numbers
Folding
Preferences...
Breakpoint Properties...

Breakpoint

- **To set a breakpoint:**
  - Open your Java source code file in IDEA (IntelliJ, Eclipse, ..).
  - In the left margin next to the line number where you want to set the breakpoint, click on the empty space. A red dot will appear, indicating the breakpoint.

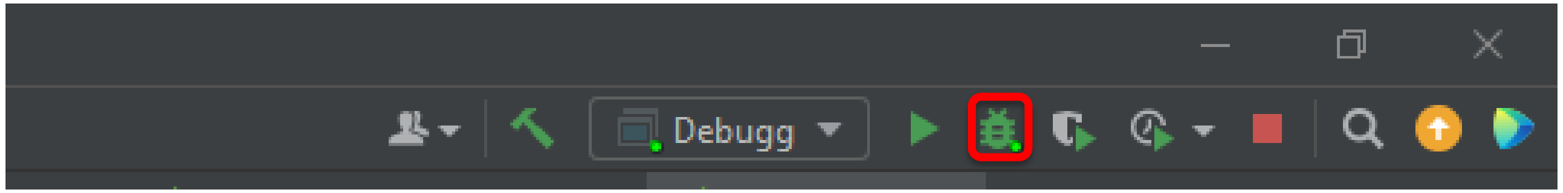


```
1 usage
public static int divide(int dividend, int divisor) {
    int result = 0;
    try {
        result = dividend / divisor;
    } catch (ArithmeticException e) {
        System.err.println("Error: Division by zero!");
    }
```

  - You can click again on the red dot to remove the breakpoint.

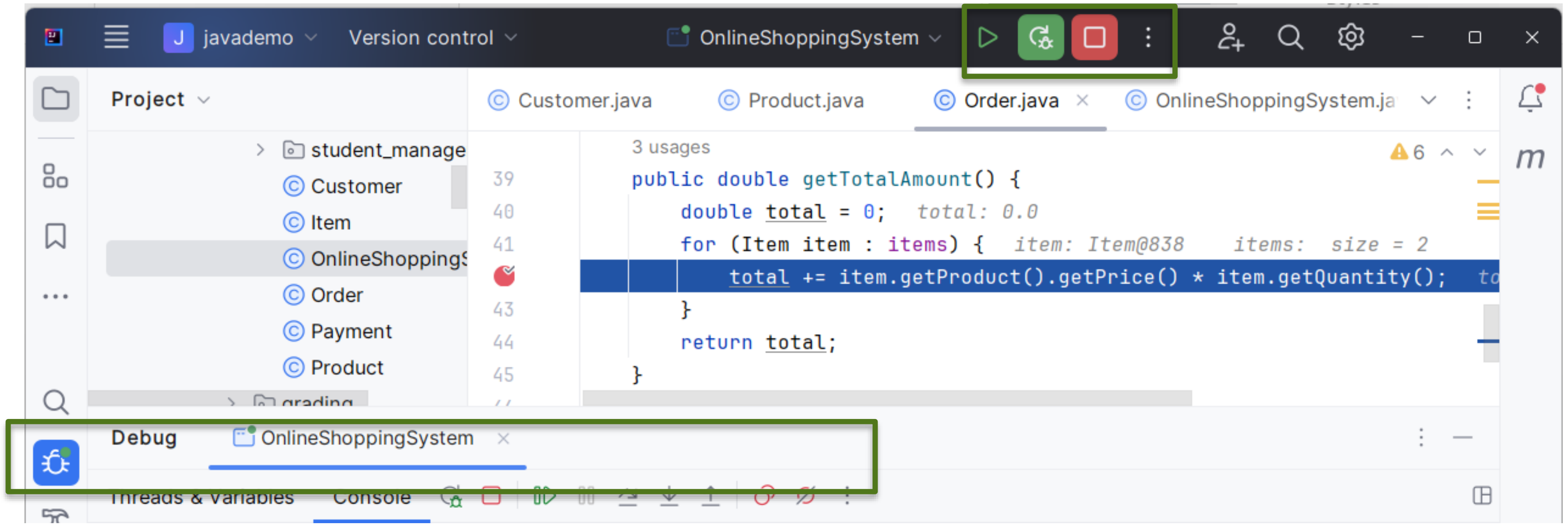
Starting the Debugger

- If you started an application once via the context menu, you can use the created launch configuration again via the **Debug** button in the IntelliJ toolbar.



Debug Perspective

- When a Java program is started in the debug mode, users are prompted to switch to the debug perspective.
  - **Debug view:** The Debug view in IntelliJ IDEA provides a comprehensive overview of your program's execution during debugging. It includes information about the call stack, breakpoints, variables, watches, and threads.



```
3 usages
public double getTotalAmount() {
    double total = 0;   total: 0.0
    for (Item item : items) {   item: Item@838    items:  size = 2
        total += item.getProduct().getPrice() * item.getQuantity();
    }
    return total;
}
```

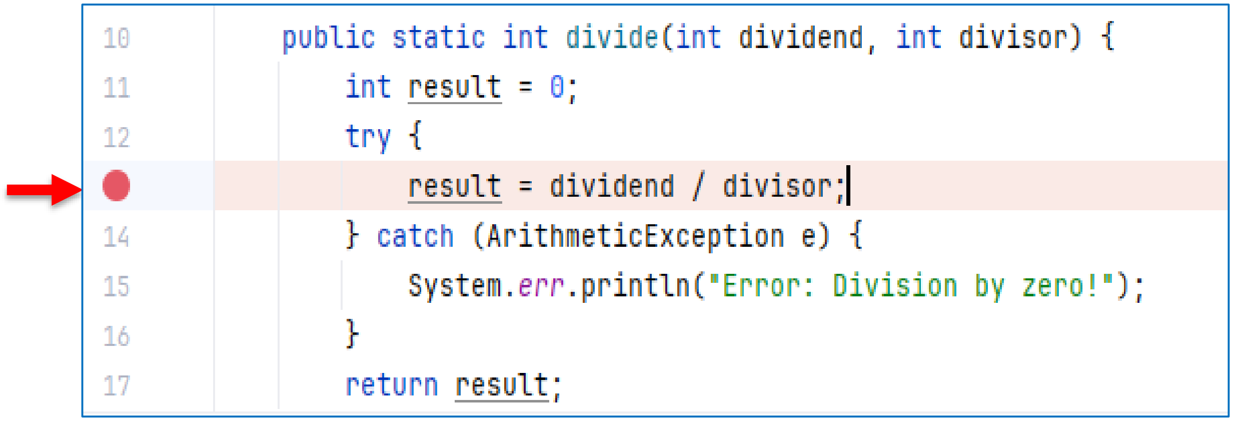
Example

- Use this class to practice debugging in IntelliJ IDEA and fix the issue.

```java
package org.example;

public class DebugExample {

    public static void main(String[] args) {
        int result = divide(10, 0);
        System.out.println("Result: " + result);
    }

    public static int divide(int dividend, int divisor) {
        int result = 0;
        try {
            result = dividend / divisor;
        } catch (ArithmeticException e) {
            System.err.println("Error: Division by zero!");
        }
        return result;
    }
}
```

Example

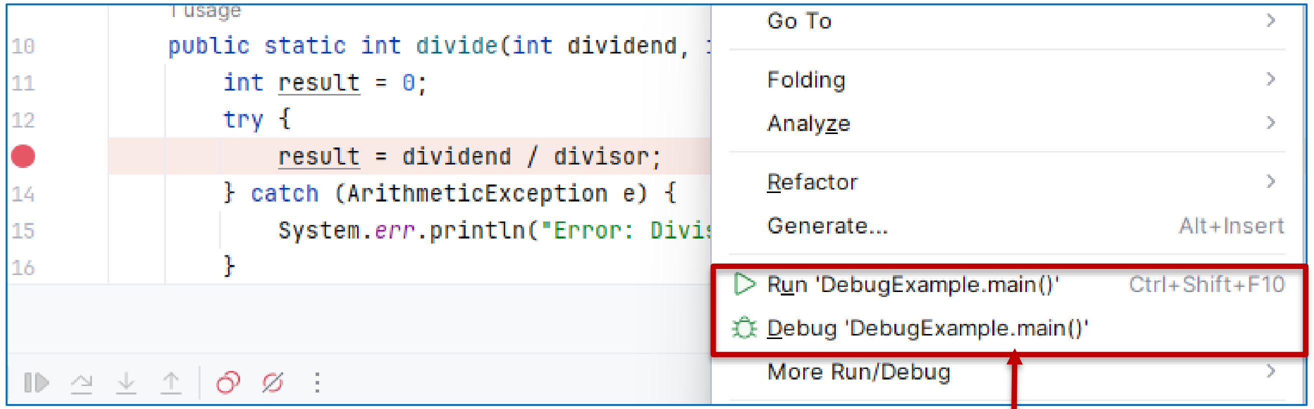
- **Step 1: Set Breakpoints**
  - Set a breakpoint on the line with: `result = dividend / divisor;`



```java
public static int divide(int dividend, int divisor) {
    int result = 0;
    try {
        result = dividend / divisor;
    } catch (ArithmeticException e) {
        System.err.println("Error: Division by zero!");
    }
    return result;
```

Example

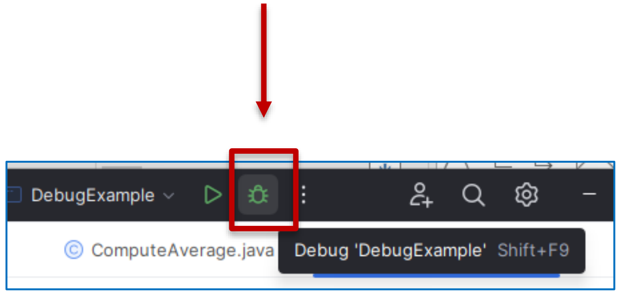
- **Step 2: Start Debugging**
  - Run the main method in debug mode by right-clicking on the main method and selecting "Debug 'main()'."



```
1 usage
public static int divide(int dividend, i
    int result = 0;
    try {
        result = dividend / divisor;
    } catch (ArithmeticException e) {
        System.err.println("Error: Divis
    }
```

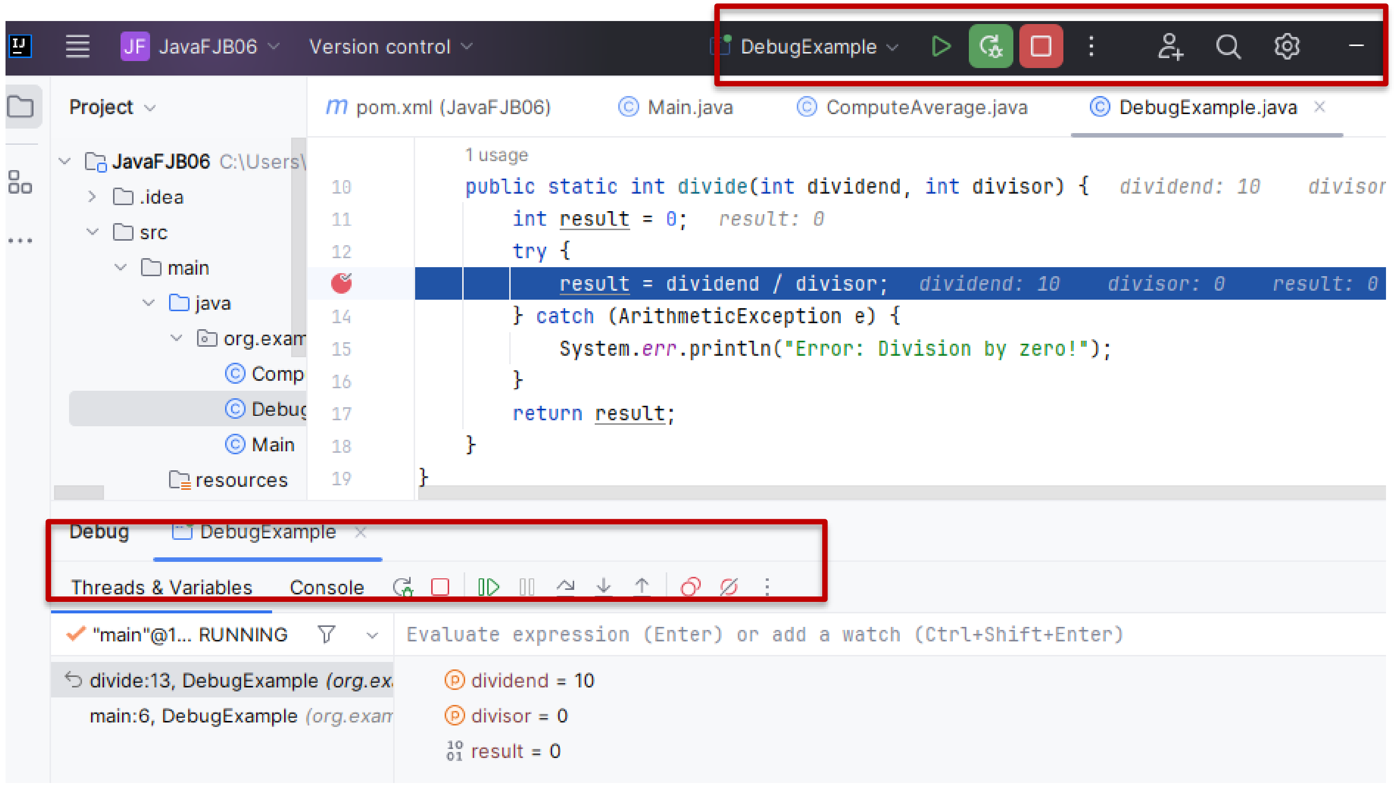
Go To
Folding
Analyze
Refactor
Generate... Alt+Insert
Run 'DebugExample.main()' Ctrl+Shift+F10
Debug 'DebugExample.main()'
More Run/Debug

Or



Example

- **Step 3: Debug Perspective**

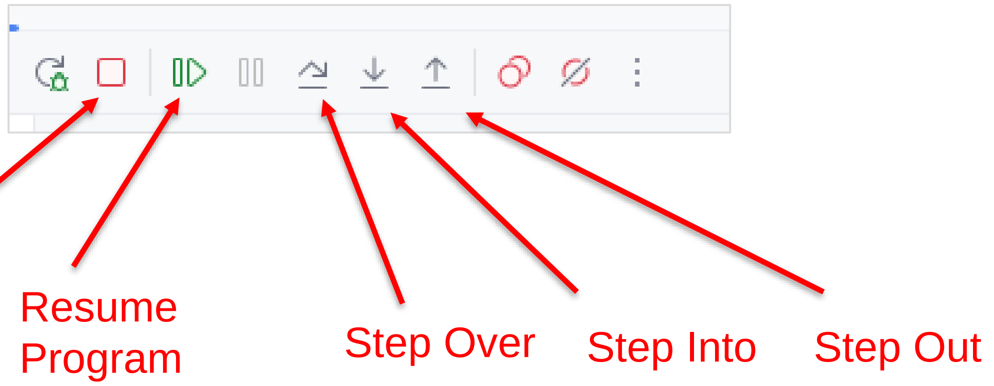


```
1 usage
public static int divide(int dividend, int divisor) {   dividend: 10    divisor
    int result = 0;   result: 0
    try {
        result = dividend / divisor;   dividend: 10    divisor: 0    result: 0
    } catch (ArithmeticException e) {
        System.err.println("Error: Division by zero!");
    }
    return result;
}
}
```

```
"main"@1... RUNNING
divide:13, DebugExample (org.ex
main:6, DebugExample (org.exam

Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
dividend = 10
divisor = 0
result = 0
```

Example

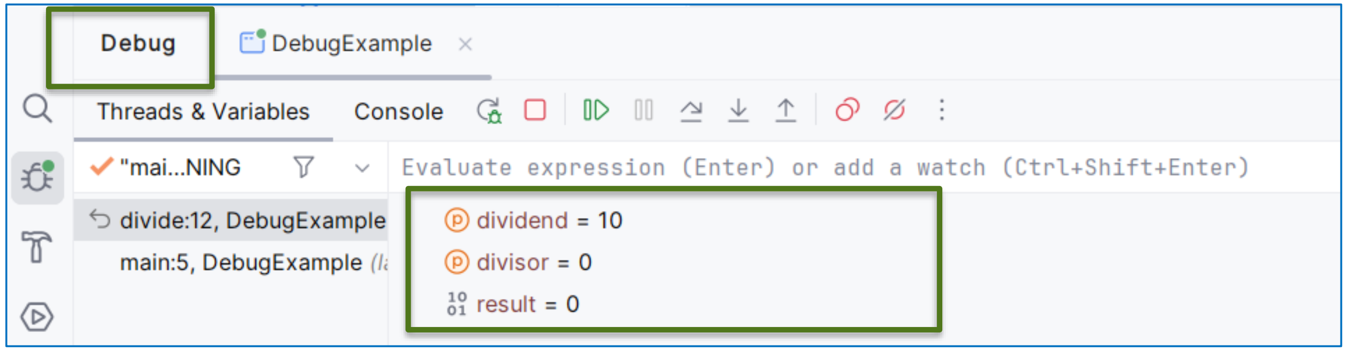
Step 4: Debugging Controls



- "**Resume Program**" button (a green arrow): to continue execution until the next breakpoint.
- "**Step Over**" button (a blue arrow): to execute the current line of code and stop at the next line.
- "**Step Into**" button (a blue arrow pointing down): to step into a method call (if any) on the current line.
- "**Step Out**" button (a blue arrow pointing up): to execute the rest of the current method and stop at the caller.
- "**Stop**" button (a red square): to terminate the debugging session.

Example

- **Step 5: Inspect Variables**
  - In the Debug Tool Window, you can view the values of variables by expanding the variables section.

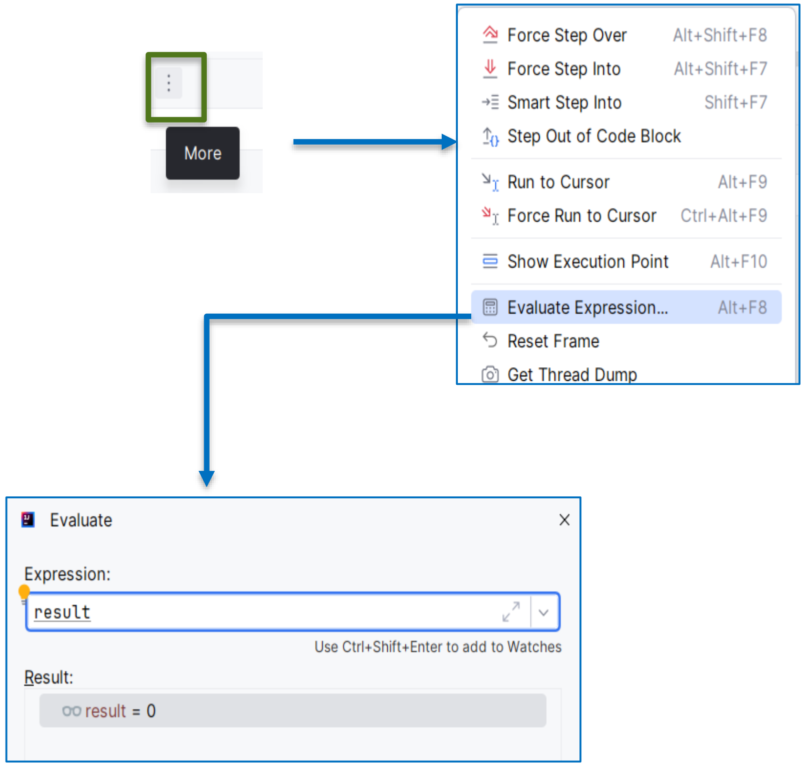


```
dividend = 10
divisor = 0
result = 0
```

  - You can also evaluate expressions by clicking on the "Evaluate Expression" button and entering expressions to be evaluated.

Example

- **Step 5: Inspect Variables (cont)**
  - You can also evaluate expressions by clicking on the "**Evaluate Expression**" button and entering expressions to be evaluated.



More
Force Step Over Alt+Shift+F8
Force Step Into Alt+Shift+F7
Smart Step Into Shift+F7
Step Out of Code Block
Run to Cursor Alt+F9
Force Run to Cursor Ctrl+Alt+F9
Show Execution Point Alt+F10
Evaluate Expression... Alt+F8
Reset Frame
Get Thread Dump
Evaluate

Expression:

```
result
```

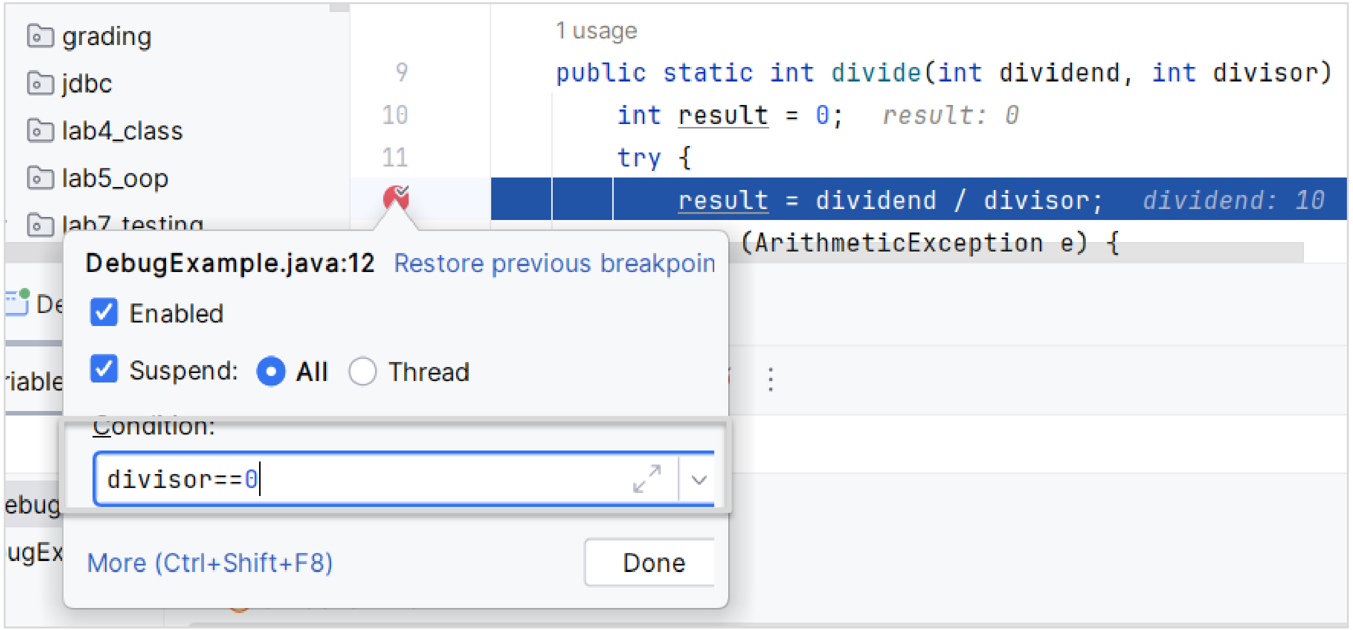
Use Ctrl+Shift+Enter to add to Watches

Result:

```
result = 0
```

Example

- **Step 6: Conditional Breakpoints (Optional)**
  - You can set conditions for your breakpoints. Right-click on a breakpoint and choose "**More.**"
  - Set a condition, and the program will pause only when the condition is met.



```
1 usage
public static int divide(int dividend, int divisor)
    int result = 0;   result: 0
    try {
        result = dividend / divisor;   dividend: 10
    (ArithmeticException e) {
```

DebugExample.java:12 Restore previous breakpoin

[✓] Enabled

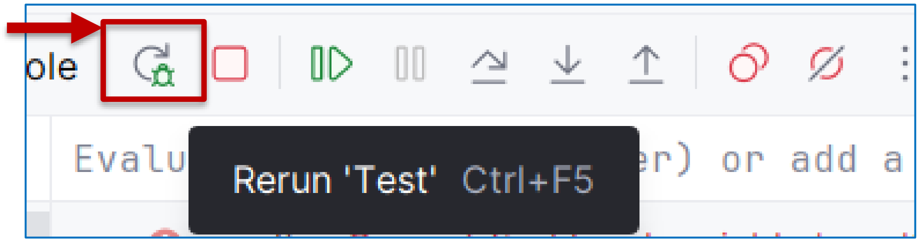
[✓] Suspend: [● All] ○ Thread

Condition:

```
divisor==0
```

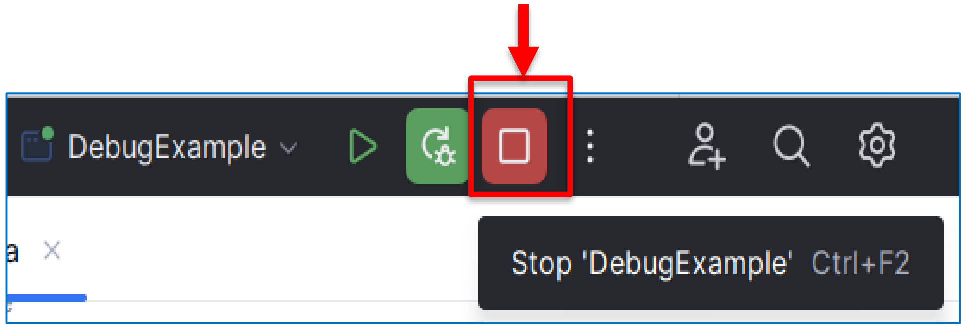
Example

- **Step 7: Fix and Continue Debugging**
  - If you identify an issue while debugging, make the necessary code changes in your source code.
  - You can then re-run the code in debug mode by clicking the "**Rerun**" button or using the shortcut **Shift + F9**.
  - Your changes will be applied in the new debugging session.

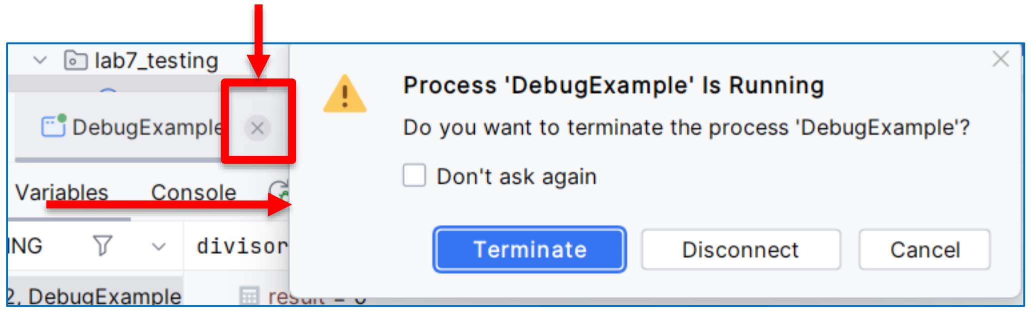


Example

- **Step 8: Stop Debugging**
  - When you're finished debugging, click the "Stop" button in the Debug Tool Window,



  - or simply close the Debug Tool Window to stop the debugging session.


Process 'DebugExample' Is Running

Do you want to terminate the process 'DebugExample'?

Don't ask again

Lesson Summary

- What is debugging?
- Breakpoints
- Starting the Debugger
- Debug Perspective
- Controlling the program execution
- Evaluating variables in the debugger
- Changing variable assignments in the debugger

THANK YOU!

Any questions?