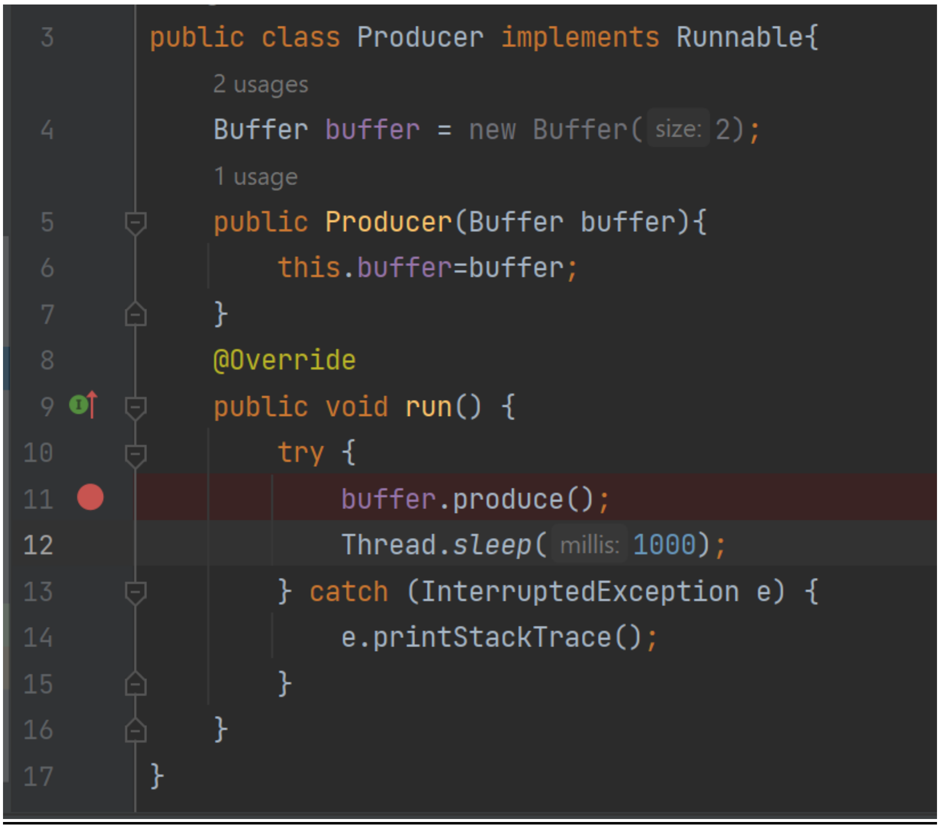
Multithreading debugging

When writing multithreaded apps, we must be extra **careful** as we may introduce **bugs** that will then be very hard to catch and fix. Concurrency-related bugs are trickier than those in a single-threaded application because of their random nature. An app may run flawlessly a thousand times and then fail unexpectedly for no obvious reason.

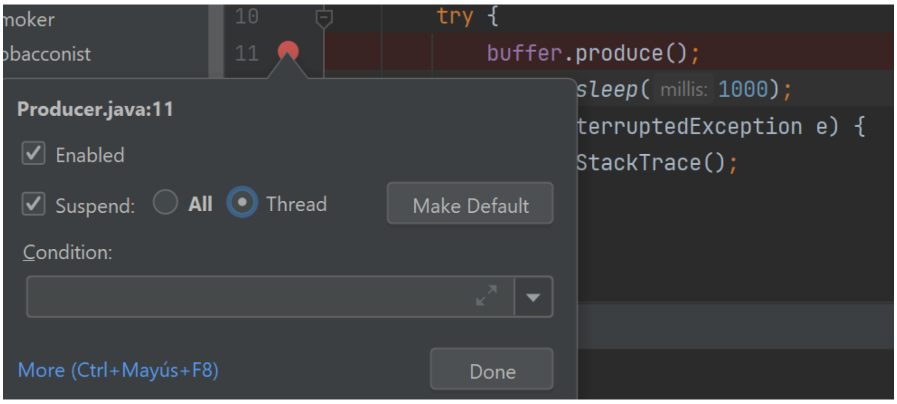
Using the **IntelliJ IDEA debugger**, you can test the multithreaded design of your application and reproduce concurrency-related bugs by controlling individual threads rather than the entire application.

To set a line breakpoint, click the gutter at the executable line of code where you want to set the breakpoint. Alternatively, place the caret at the line and press Ctrl+F8.



```
public class Producer implements Runnable{
    2 usages
    Buffer buffer = new Buffer( size: 2);
    1 usage
    public Producer(Buffer buffer){
        this.buffer=buffer;
    }
    @Override
    public void run() {
        try {
            buffer.produce();
            Thread.sleep( millis: 1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Configure the **breakpoint** to only **suspend** the **thread** in which it was **hit**. To do this, right click the **breakpoint**, then click **Thread**.



```
moker
bacconist
        try {
            buffer.produce();
        sleep( millis: 1000);
        terruptedException e) {
        StackTrace();
```

Producer.java:11

☑ Enabled

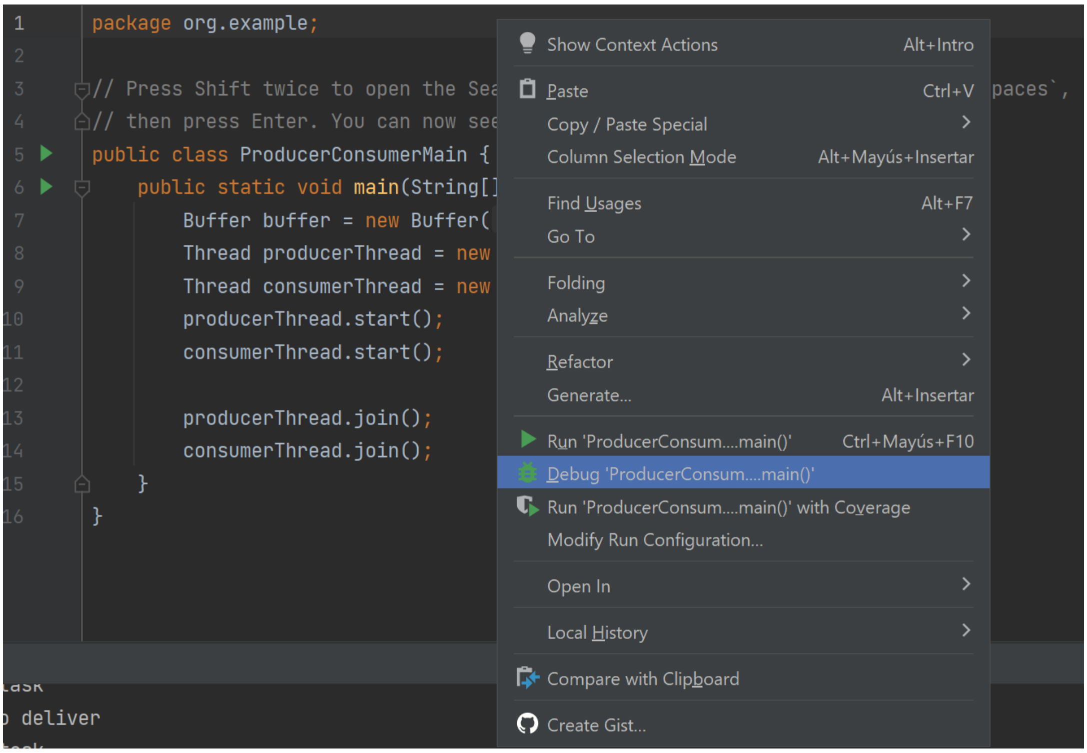
☑ Suspend: ○ All ● Thread Make Default

Condition:

More (Ctrl+Mayús+F8) Done

If you often suspend individual threads, you can click Make Default to make every new breakpoint, only suspend the thread where it was hit.

Start the **debug session** by clicking the **Run button** near the main method and selecting **Debug**.



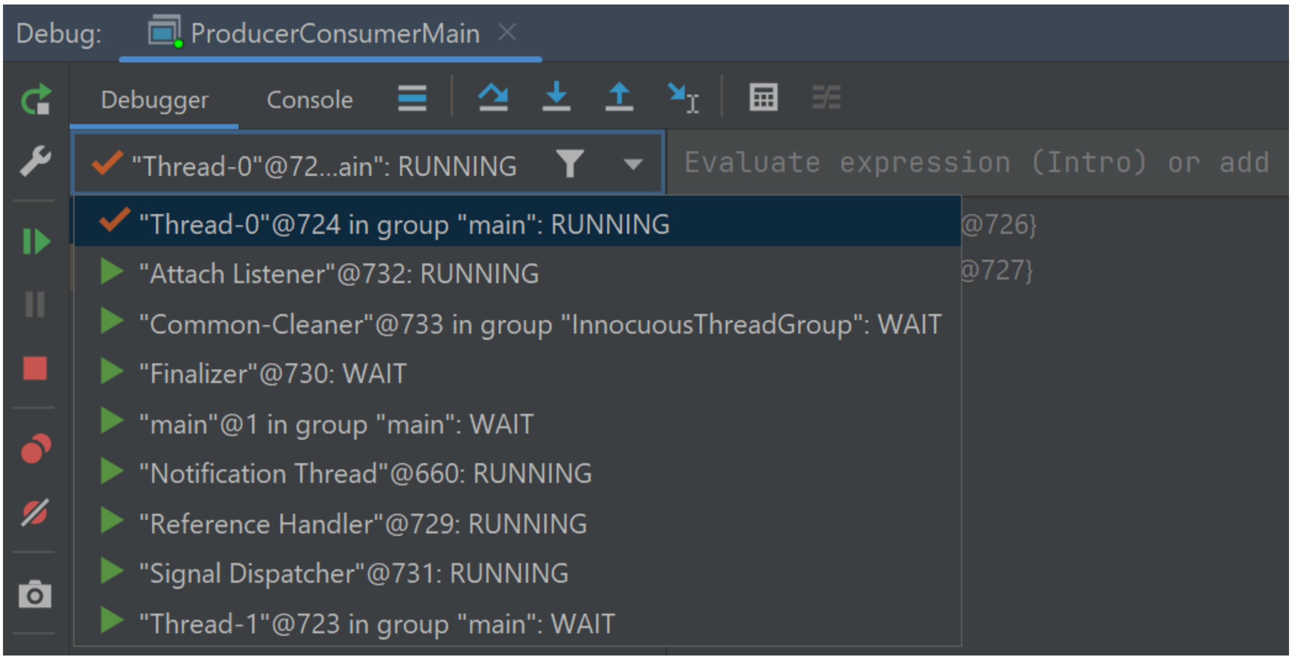
```
package org.example;

// Press Shift twice to open the Sea                    paces`,
// then press Enter. You can now see
public class ProducerConsumerMain {
    public static void main(String[]
        Buffer buffer = new Buffer(
        Thread producerThread = new
        Thread consumerThread = new
        producerThread.start();
        consumerThread.start();

        producerThread.join();
        consumerThread.join();
    }
}
```

Show Context Actions Alt+Intro
Paste Ctrl+V
Copy / Paste Special
Column Selection Mode Alt+Mayús+Insertar
Find Usages Alt+F7
Go To
Folding
Analyze
Refactor
Generate... Alt+Insertar
Run 'ProducerConsum....main()' Ctrl+Mayús+F10
Debug 'ProducerConsum....main()'
Run 'ProducerConsum....main()' with Coverage
Modify Run Configuration...
Open In
Local History
Compare with Clipboard
Create Gist...

When the program run, all threads are individually suspended in the method with the breakpoint. Now you can switch between the threads (in the Frames or Threads tab) and control the execution of each thread.



"Thread-0"@72...ain": RUNNING

✓ "Thread-0"@724 in group "main": RUNNING
"Attach Listener"@732: RUNNING
"Common-Cleaner"@733 in group "InnocuousThreadGroup": WAIT
"Finalizer"@730: WAIT
"main"@1 in group "main": WAIT
"Notification Thread"@660: RUNNING
"Reference Handler"@729: RUNNING
"Signal Dispatcher"@731: RUNNING
"Thread-1"@723 in group "main": WAIT

You can **resume** a thread by pressing **F9** or clicking the **Resume button** in the left part of the Debug tool window.

After you resume all kid threads, the debugger automatically switches back to the main thread.

Resume the main thread to let it execute the remaining statements and then terminate.

Review the program output in the Console tab.