Lab

YourKit

This lab has two parts. The purpose of part one is to give you some experience using YourKit on a trivial example. In part two, you will use the knowledge gained in part one to identify and fix a performance bug in a more complex program.

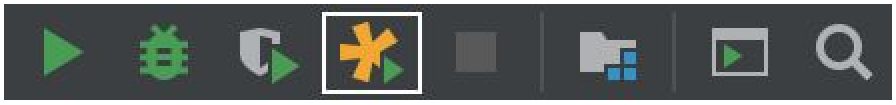
Part 1

For this part of the lab you will use YourKit to find out how many times a function runs and how long it takes to run with different inputs in a given program.

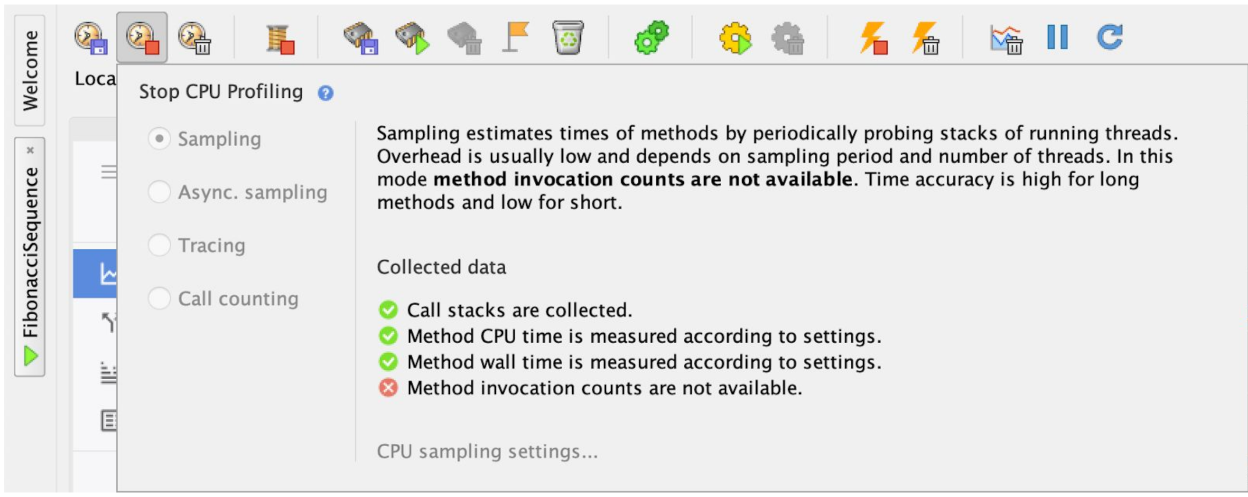
1. Create a new Java project in IntelliJ Idea
2. Add the FibonacciSequence.java file to your project
    a. The purpose of this file is to calculate the nth number in the Fibonacci Sequence. Quickly read about the Fibonacci Sequence if you are unfamiliar with it. https://www.mathsisfun.com/numbers/fibonacci-sequence.html
3. Create a table like the one below. You will be filling out this table to turn in at the end of part 1. The last row is an example of how to fill out the table.

CPU analysis of the FibonacciSequence Class			
Number in Sequence (n)	Time Spent in Fibonacci Function (MS)	Fibonacci Function Count	Fibonacci Number
32			
33			
34			
35			
40 (example)	114,559	323,216,348	165580141

4. Read the descriptions for each of the columns below to understand what numbers should be in the table
    a. Number in Sequence: the input the user gives to the console
    b. Time Spent in Fibonacci Function (MS): the time in milliseconds spent in the Fibonacci function (You will use YourKit to find this)
    c. Fibonacci Function Count: the number of times the Fibonacci function was called (You will use YourKit to find this)
    d. Fibonacci Number: The number outputted by the console
5. Click on the YourKit icon in IntelliJ (near the upper right corner, to the right of the run configurations drop-down), or right-click on the FibonacciSequence class and select "Profile…" to start it up.



    a. Go to YourKit and hover over the second button (the one highlighted in the image below) and press the button to stop profiling.



Stop CPU Profiling

(●) Sampling
( ) Async. sampling
( ) Tracing
( ) Call counting

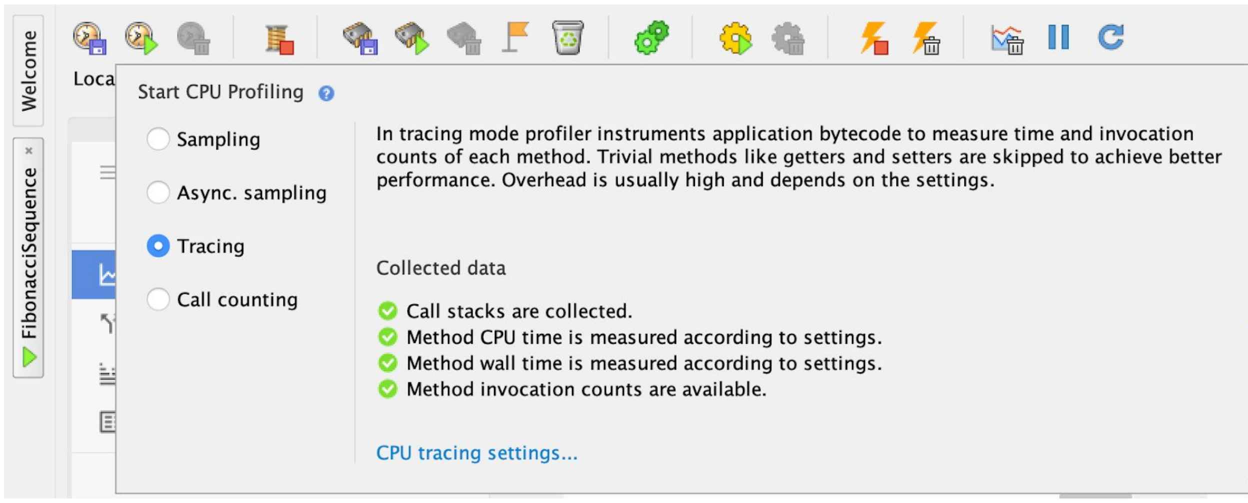
Sampling estimates times of methods by periodically probing stacks of running threads. Overhead is usually low and depends on sampling period and number of threads. In this mode **method invocation counts are not available**. Time accuracy is high for long methods and low for short.

Collected data

✓ Call stacks are collected.
✓ Method CPU time is measured according to settings.
✓ Method wall time is measured according to settings.
✗ Method invocation counts are not available.

CPU sampling settings...

    b. The bottom of the icon will change to a green diamond. Hover over it again and when the popup appears, select tracing.



Start CPU Profiling

Sampling

Async. sampling

[✓ Tracing]

Call counting

In tracing mode profiler instruments application bytecode to measure time and invocation counts of each method. Trivial methods like getters and setters are skipped to achieve better performance. Overhead is usually high and depends on the settings.

Collected data

✓ Call stacks are collected.
✓ Method CPU time is measured according to settings.
✓ Method wall time is measured according to settings.
✓ Method invocation counts are available.

CPU tracing settings...

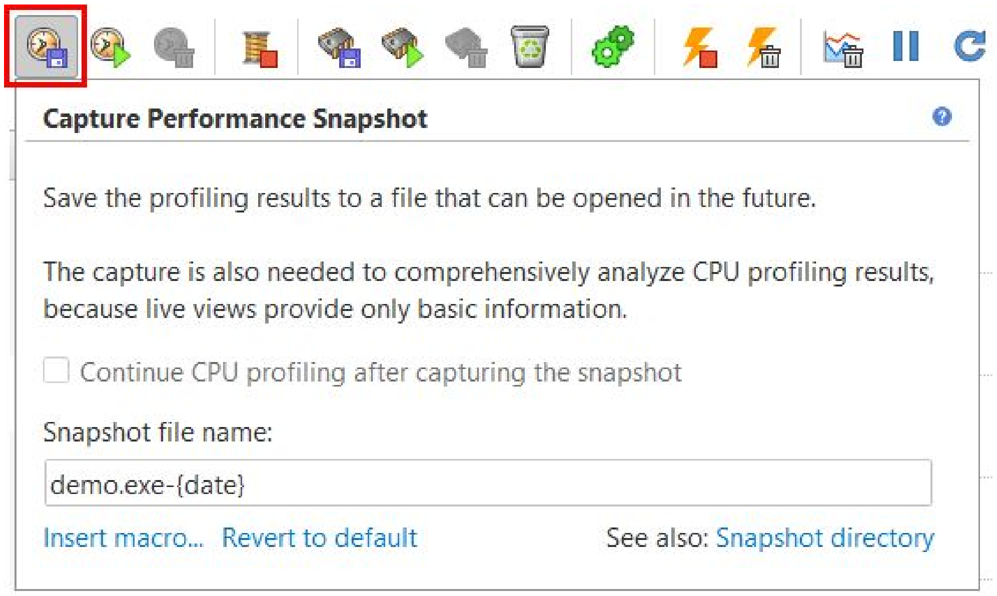
c. Push the same button again to start profiling (this time in tracing mode).

d. Click on “Call tree - All threads merged”. It should not be necessary to have this option selected, but we have found that for some users, calls are not logged if this is not selected.

e. Go back to Intellij, type the ‘n’ number you are being prompted to enter in the console, and press enter.

f. Wait for the program to complete. For large ‘n’ (in the 30’s or higher) it can take several minutes for the program to finish.

g. Go back to YourKit and click the save button in the top left corner.



Capture Performance Snapshot

Save the profiling results to a file that can be opened in the future.

The capture is also needed to comprehensively analyze CPU profiling results, because live views provide only basic information.

☐ Continue CPU profiling after capturing the snapshot

Snapshot file name:

demo.exe-{date}

Insert macro... Revert to default See also: Snapshot directory

h. Once saved, open and select “Call tree - By thread”. You should see two panes with method run information. If the panes are empty, see the note below. Click on the main method in the top pane and look at the “Callees List” in the bottom pane. Find the time (Own Time) and count there for the fibonacci(int) method and enter that in your table.

Note: There are two reasons we know of that could cause the panes to be empty. 1) you didn't have "Call tree - By Thread" selected before entering the input number in Intellij to continue the program, and 2) the program ran too fast. YourKit does not include methods that run in less than a millisecond. Try again, making sure you carefully follow all steps above. If you still don't see results, indicate for the appropriate row in your table that no results were displayed and continue to the next step.

i. Repeat the above steps, specifying a different input value until all rows in your table are filled in.

6. Save your table as a PDF and turn it in to Canvas. Make sure to use commas for large numbers (example do **2,000,000** and **not 2000000**).

Part 2

Now that you know how to profile a Java program with YourKit, you will use it to find a performance bug in a more complex program. The program is the Weather station program you used in the second mocking tutorial with a few modifications.

1. Create a new Intellij project.
2. Download this code, unzip it and copy all of the code into your project's source code.
3. Profile the WeatherStation class (following the same steps you used for part 1) to find and correct the performance bug.

   Hint: The bug is in the method in the Callees List that has the largest "Own Time"
4. Once you have corrected the bug, take a screenshot of the corrected method that contained the bug and submit it to Canvas.