Technical University of Denmark

A Short Guide to

Virtual Threads in Loom

Authors:

Kasper Solhøj Jørgensen

Tobias van Deurs Lundsgaard

19th of June, 2022

DTU Compute
Department of Applied Mathematics and Computer Science

Getting Started with Loom

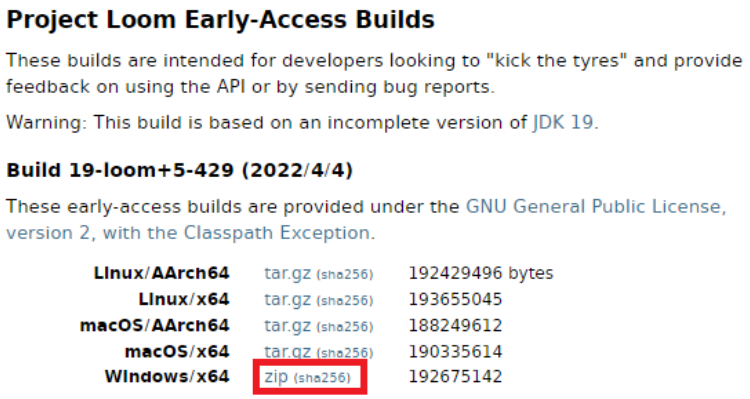
This note serves as a short guide on how to set up Project Loom on your Windows computer. It will also mention a few commands to get you started with using virtual threads for your applications. Note that this guide is made for devices running Windows 10 and 11.

Note: As of August 2022, Java 19 is now an official version, and you may install it directly. Loom is still in preview though, so you should use the **--enable-preview** option as described below.

Installation

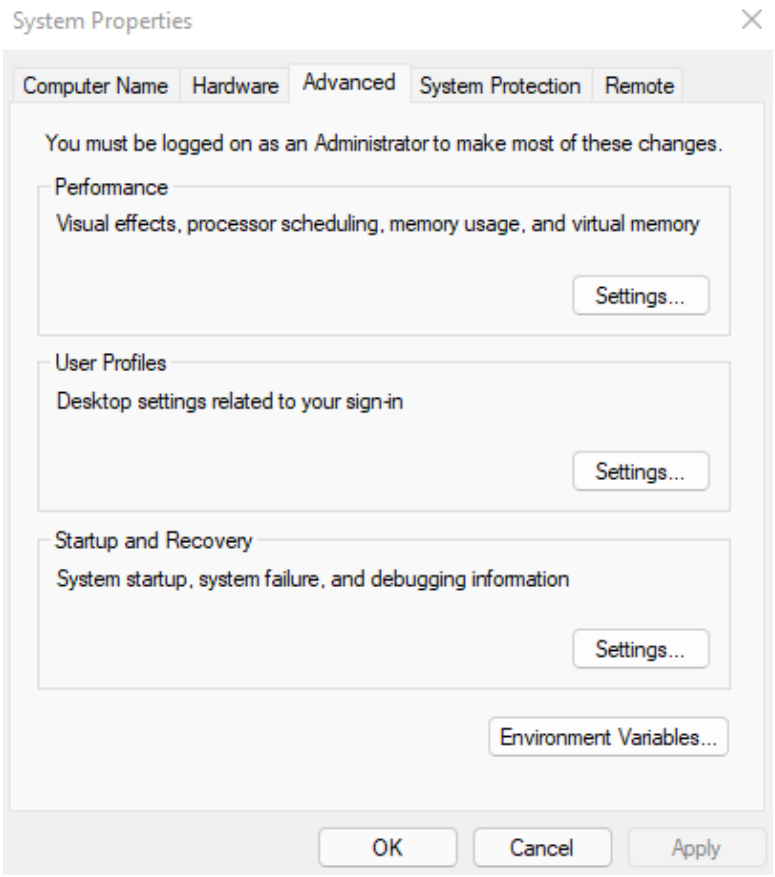
In this section we shall describe the installation process for getting Project Loom set up on your Windows machine.

Firstly you must download the zip file of JDK-19 with Project Loom. The file can be found here: **https://jdk.java.net/loom/**
You should see something like the image below. Click the highlighted download link.

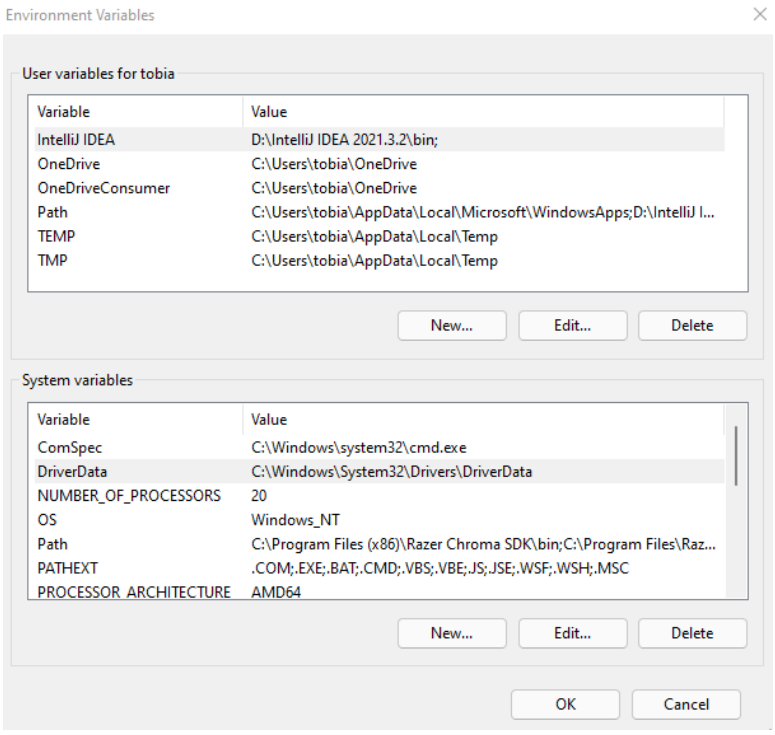


Once the zip file has been downloaded, extract it to your computer in a location you prefer. A good option would be the Java folder. My path is **C:\Program Files\Java**.

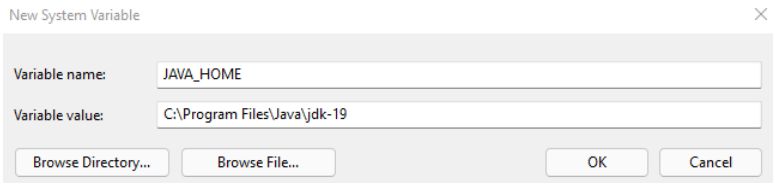
Now we need to change a few environment variables, to enable JDK-19 on your computer. Search for **Environment Variables** on your Windows search bar. This should get you to the following window.



Here we click on **Environment Variables** at the bottom of the window. Doing so should take you to the following window:

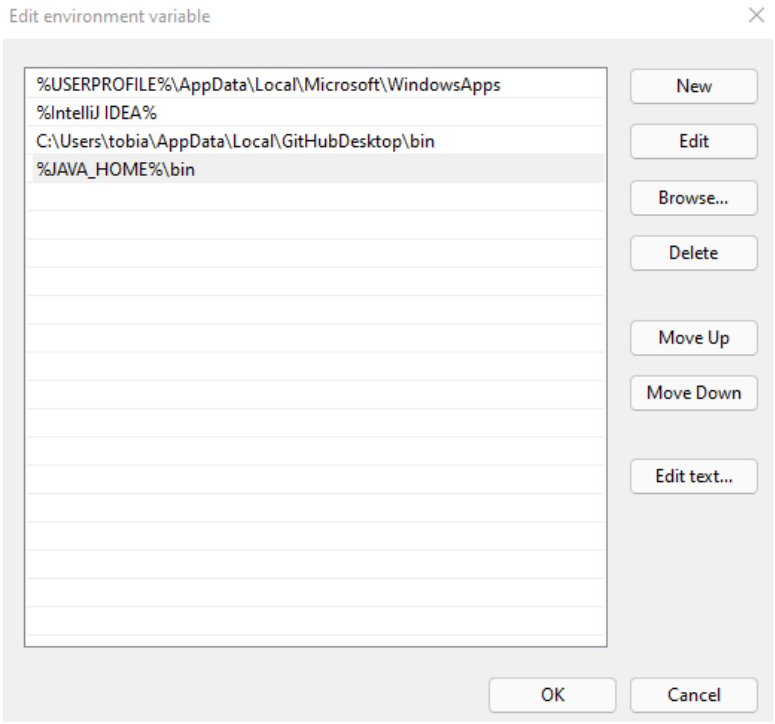


Now we need to add some new system variables. These can be found in the lower list of the image above. Click the "New" button and write the variable name **JAVA_HOME** and set variable value to the path in which you chose to extract JDK-19. It could look like the following:



Now should go back and add a line to the environment variables located above the system variables. Click "new" and add the line:
%JAVA_HOME%\bin

The environment variables should now look something like the following image. You might have more or less environment variables.



Once these steps have been completed we can use JDK-19 and our virtual threads.

We shall ensure that Project Loom works by running a test program. The program will be a Hello World using the following code. Simply copy the code into a file named HelloWorld.java:

```
    public class HelloWorld implements  Runnable {
        public static void main(String[] args) throws InterruptedException {
            Thread thread = Thread.startVirtualThread(new HelloWorld());
            thread.join();
        }

        @Override
        public void run() {
            System.out.println("Hello World");
        }
    }

```

To run the code, open a terminal and navigate to the folder in which you have stored the HelloWorld.java file. To compile the file, use the following command:
javac HelloWorld.java –enable-preview –release 19

Afterwards you can run the program with this command:
java –enable-preview HelloWorld

You should see the words "Hello World" being printed in your terminal. This is an indication that the program works and that you have successfully installed the JDK. You are now free to use virtual threads.

Coding With Virtual Threads

We shall now provide a brief explanation of some of the tools you have available to use when coding with virtual threads. The most important thing to remember is that virtual threads have been designed to be very similar to existing thread tools in Java.

Creating and Starting Virtual Threads

Creating a virtual thread to run some code is simple. In the following code we shall see how to create and start a virtual thread. We shall use a class named RunTask which implements either callable or runnable.

```
    Thread thread = Thread.ofVirtual().start(new RunTask());

```

It is also possible to start a virtual thread with a different command which can be seen below. Which one you use is a simple matter of preference.

```
    Thread thread = Thread.startVirtualThread(new RunTask());

```

It is also possible to create an unstarted thread that you wish to start later.

```
    Thread thread = Thread.ofVirtual().unstarted(new RunTask());

```

When we would like to start the unstarted virtual thread we just use *.start()*

```
    thread.start();

```

Executor

It is also possible to make an *ExecutorService* for virtual threads. Those familiar with using *ExecutorService* thread pools in Java will see that the structure is very familiar.

```
    ExecutorService es = Executors.newVirtualThreadPerTaskExecutor();

```

As the name implies, for every submitted task this *ExecutorService* will create a virtual thread to handle it. Submitting a task is as simple as the following:

```
    es.submit(new RunTask());

```

You should now be able to use virtual threads to achieve concurrency in Java. Remember that the precautions to avoid issues like race conditions must still be upheld when coding with virtual threads.