Introduction to Threads

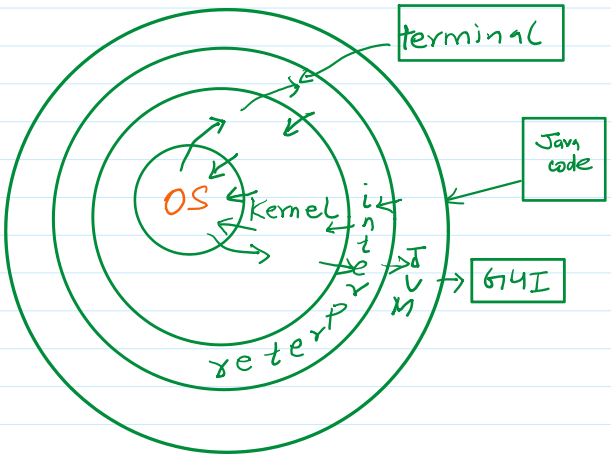
11 January 2026 17:00

The concept of thread is not a new one. For sometimes many operating system have had libraries that provide the C programmer with a mechanism to create threads. Other languages, such as Ada have support for threads embedded into the language, much as support for threads is built into the Java language. Nonetheless, the topic of thread is usually considered a peripheral programming topic. One that is only needed in special programming cases.

With Java, things are different. It is impossible to write any but the simplest Java program without introduction the topic of threads. And the popularity of Java ensures that many developer who might never have considered learning about threading possibilities in languages like C or C++ need to become fluent in threaded Programming.

Java Terms

Java: First, in terms of Java itself. As we know, Java started out as a programming language, and many people today think of Java as being simply a programming language. But Java is. Much more than just a programming language. It's also an API specification and a virtual machine specification. So when we say Java. We mean the entire Java platform. A programming language, an API and a virtual machine specification that taken together. Define an entire programming and runtime environment. Often we say Java. It's clear from the context that we're talking specifically about the programming language or parts of the Java API, or the virtual machine. The point to remember is that threading features we discuss. Derive their properties from all the components of the Java platform, taken as a whole. While it is possible to take the Java programming language directly. Compile it into assembly code and run it outside of the virtual machine. Such an executable may not necessarily behave the same as the programs.



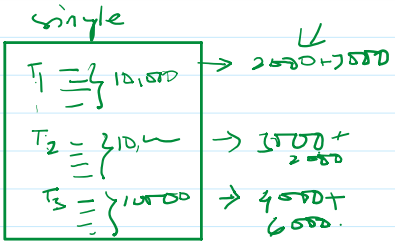
Virtual machine, interpreters and browsers

The Java Virtual Machine is another term for the Java interpreter, which is the code that ultimately runs Java programs by interpreting the intermediate byte code format of the Java programming language. The Java interpreter actually comes in three popular forms. The interpreter for developer (called Java.) That runs the program via a command line or a File Manager. The interpreter for end users (called JRE) That is a subset of the developer environment and from the basis of (among other things) the Java plug in. And the interpreter that is built into many popular web browsers:

- **Java Web Start: (not-in use)**
- Modern JDK version no longer include the Java browser plugin, making applet support obsolete in most environments.
- For secure and updated Java application deployments, transitioning to Java Web Start or standalone Java applications is recommended.
- We can use Spring framework instead of Java Web Start.

Programs, applications
This leads us to the term that we will use for things written in the Java language. Generally we call such entities programs, but there are two types of programs a typical Java programmer might write. Programs that can be run directly by the Java interpreter. And programs designed to be run by Java enabled browsers. The term **application** is used for stand-alone Java programs.

What exactly thread?
The term thread is a shorthand for a thread of control. And a thread of control is, at its simplest, a section of code executed independently of the other threads of control within the single program.

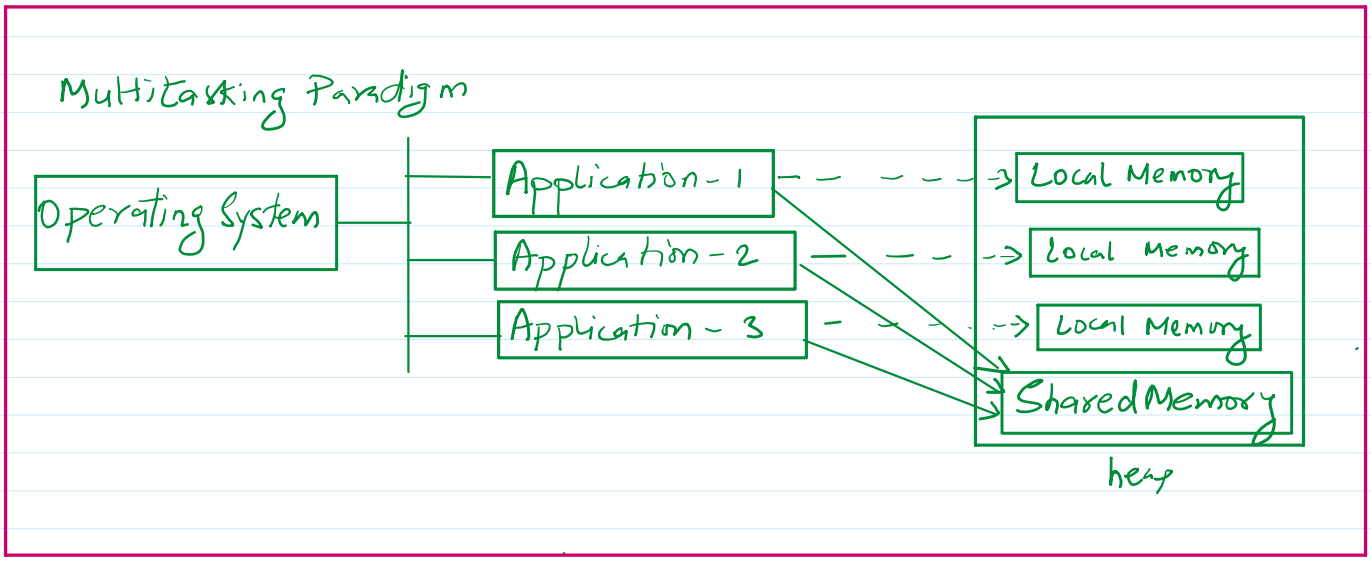


Thread of Control
Thread of control sounds like complicated technical term, but it's really a simple concept. It is a path taken by a program during execution. This determines what code will be executed. Does the if block get executed or does the else block? How many times does the while loop execute? If we were executing a task from A "to do" list as much as computer executes an application, what setup we perform and the order in which we perform them? Is our path of execution the result of our thread of control.

Having multiple threads of control is like executing task from two lists. We are still doing the task on each To Do List in the correct order, but when we get bored with the task on one of the list, we switch list with the intention of returning at some future time to the 1st list at the exact point where we left off.

Overview of Multitasking
We all are familiar with the use of multitasking operating system to run multiple programs simultaneously. Each of these programs has at least one thread within it, so at the same level we are already comfortable with the notion of a thread in a single process. The single threaded process has the following properties, which as it turns out are shared by all threads in a program with multiple threads as well:
- The process begins execution at a well-known point. In programming languages like C and C++ (Not to mention Java itself) the thread begins the execution at the first statement of the function or method call main().
- Execution of statement follows in completely ordered pre-defined sequence for a given set of inputs. An individual process is a single minded in this regard. It simply executes the next statement in the program.
- While executing the process has access to certain data. In Java there are three types of data a process can access: Local variables are accessed from threads stack, instance variables are accessed through object references and static variables are accessed through class or object references.

Now consider what happens when you sit at your computer and start two single threaded programs, a text editor, say, and a File Manager. You now have two processes running on your computer. Each process has a single thread with the properties just outlined. Each process does not necessarily know about the other process. Although depending on the operating system running on your computer, there are several ways in which the processes can send each other various messages. A common behavior is that you can drag a file icon from the File Manager into the text editor in order to edit the file. Each process thus runs independently of the other. Although they can cooperate if they so choose.



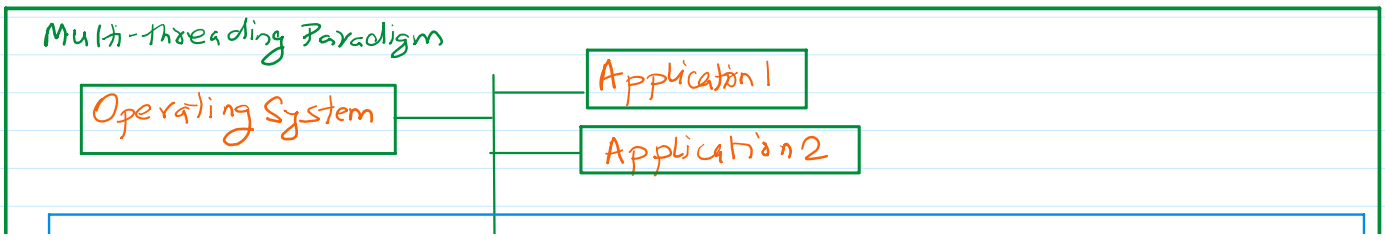
Process in a multitasking environment.

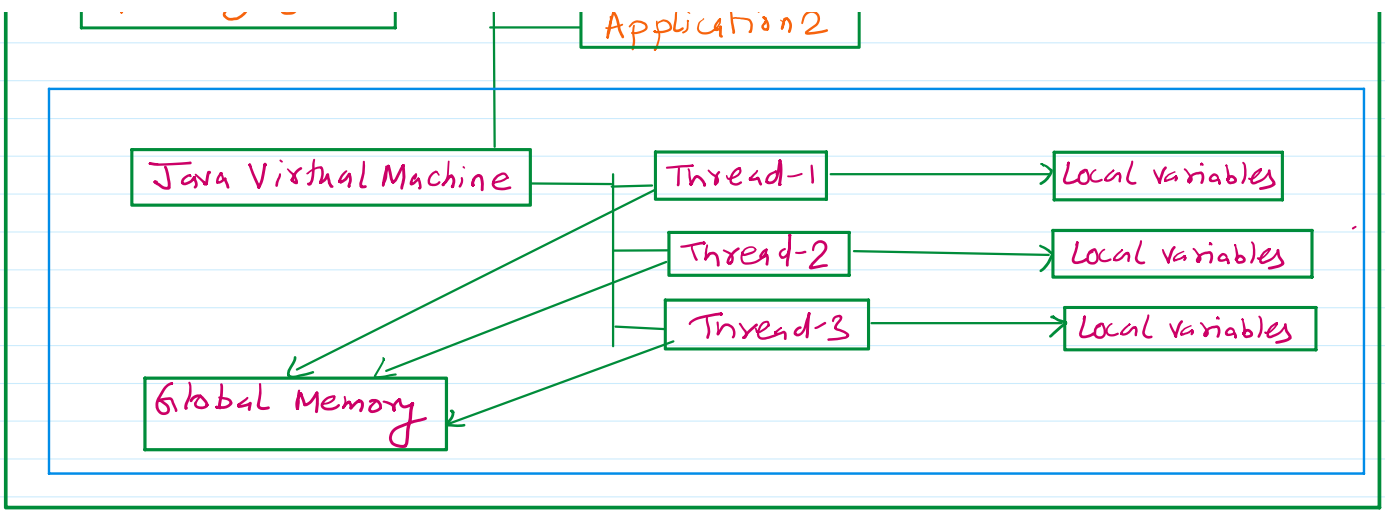
From the point of view of a person using the computer, these processes often appear to execute simultaneously, although many variables can affect that appearance. These variables depend on the operating system, for example. A given operating system may not support multitasking at all, so that no two programs appear to execute simultaneously, or the user may have decided That a particular process is more important than the other processes and hence should always run. Shutting out the other processes from running and again affecting the appearance of simultaneity.

Finally, the data contained within these two processes is by default separated. Each has its own stack for local variables and has its own data area for objects and other data elements. Under many operating system, the programmer can make arrangements so that the data object reside in the memory that can be shared between the processes, allowing both processes to access them.

Overview of Multitasking

All of this leads us to a common analogy. We can think of a thread just as we think of a process, and we can consider a program with multiple threads running within a single instance of a Java Virtual Machine, just as we consider multiple processes within an operating system.




Application2
Java Virtual Machine
Thread-1
Thread-2
Thread-3
Local variables
Local variables
Local variables
Global Memory