Java Practical Session

IN5170-Autumn 2022

August 25, 2022

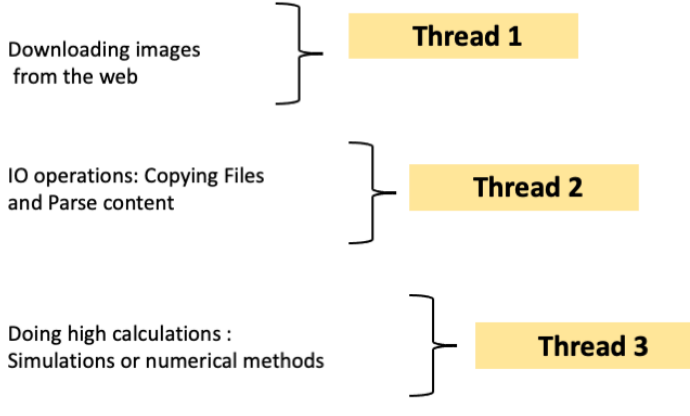
Multithreading

- By default programming languages are sequential
- Public static void main(String {} args{
  initalize array();
  downloadStocks ();
  initializeTimeSeriesModels();
  makepredictions();
  }
- In single-threaded application these methods will be called one by one
- Time consuming and might freeze the application

Multithreading

- Multithreading could be separating one or more tasks which are time-consuming and might be subject to interference by the execution of order tasks.
- **Example**: Stock market software that's able to download data from the web (Yahoo Finance). It may take several minutes to fetch the data, but we want the application to be not frozen.
- **Solution** : Create a distinct thread for the download operation and during this, the user can do whatever he wants in the application.

Multithreading



Multithreading

*„Multithreading is the ability of the **CPU** to execute multiple processes or threads concurrently"*

Both **processes** and **threads** are independent sequences of execution

PROCESS: a process is an instance of program execution

→ when you open a software or a web browser: these are distinct processes

→ the **OS** assigns distinct registers, stack memory and heap memory to every single process

in **Java** we can create processes with the help of **ProcessBuilder** class



Java concurrency and Threads

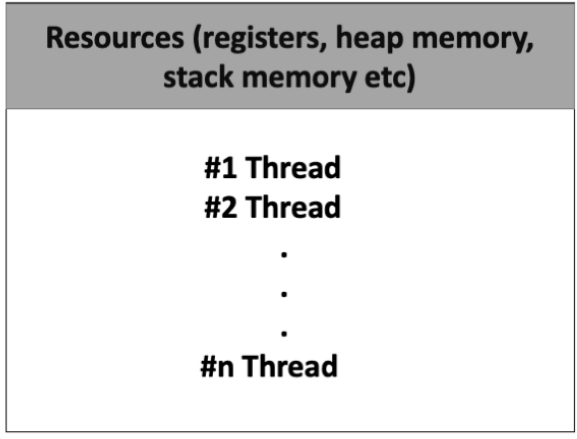
- A process is a unit of execution that has its own memory space.
- If one java application is running and we run another one each application has its own memory space in heap.
- A thread is a unit of execution within a process. So each process can have multiple threads and in java, every process or application has at least one thread and that's the main thread for UI applications, this is just called the JavaFx application thread.
- Every java process also has multiple system threads that handle everyday tasks like memory management and input/output.

Memory Management of Threads

- Every thread created by a process shares that processes memory and files.
- Now in addition to the process's memory or heap, each thread's got what's called a thread stack and the thread stack is a memory that only that particular thread can access.
- So every java application runs as a single process and each process can then have multiple threads within it. So every process has a heap and every thread has a thread stack.

Memory Management of Threads

- The local variables and method arguments, as well as method calls, are stored on the stack
- Objects are stored on the heap memory and live as long as it is referenced from somewhere form the application
- Every thread has its own stack memory but all threads share the heap memory


Resources (registers, heap memory, stack memory etc)
#1 Thread
#2 Thread
.
.
.
#n Thread

Thread Life Cycle Java

- **NEW** : Instantiation of thread, It is in this state until we start it –**start()** method.
- **RUNNABLE**: After starting the thread, it's in Runnable state executing the tasks
- **WAITING**: waiting for signals from other thread. – **wait()** and **sleep()** methods
- **DEAD**: After the thread finishes its task

Note: Thread class has more methods that can be performed on the threads for more information refer link

Threads

There are two ways to create a new thread of execution.

- One is to declare a class to be a subclass of Thread. This subclass should override the run method of class Thread. An instance of the subclass can then be allocated and started.
- The other way to create a thread is to declare a class that implements the Runnable interface. That class then implements the run method.

Interrupt and Join

An interrupt is an indication to a thread that it should stop what it is doing and do something else. It's up to the programmer to decide exactly how a thread responds to an interrupt, but it is very common for the thread to terminate

Many methods that throw InterruptedException, such as sleep, are designed to cancel their current operation and return immediately when an interrupt is received

The join method allows one thread to wait for the completion of another. If t is a Thread object whose thread is currently executing, **t.join();** causes the current thread to pause execution until t's thread terminates.

ThreadInterference : Counter Example

Interference happens when two operations, running in different threads, but acting on the same data, interleave. This means that the two operations consist of multiple steps, and the sequences of steps overlap

Suppose 2 threads, A and B are accessing the same counter object, A calls increment and B calls decrement. Either of these operations consists of at least 3 steps.

- Read C from memory
- Increment or decrement C
- Write back C to the memory.

To avoid this we use thread synchronization.

References


Oracle Thread Class
https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html


Oracle Java Documentation
https://docs.oracle.com/en/java/