Intro to Java (Page 4)

JUMP June 2019

Draft: 07/09/2019

Table of Contents

Multithreaded Programming

1. Fundamentals 101
2. Thread Class and the Runnable Interface
3. Creating a Thread
4. Extending Thread
5. Multiple Threads
6. When does a Thread end?
7. Thread Priorities
8. Synchronization
9. Synchronized Methods
10. “synchronization” statement
11. Thread Communication
12. Suspending, Resuming and Stopping Threads

Multithreaded Programming

1. Fundamentals 101
2. Thread Class and the Runnable Interface
3. Creating a Thread
4. Multiple Threads
5. When does a Thread end?
6. Thread Priorities
7. Synchronization
8. Sync Methods
9. “synchronization” block statement
10. Thread Communication
11. Suspending, Resuming and Stopping Threads

Multithreaded Programming - Fundamentals 101

- A multithread program contains two or more parts than can execute concurrently.
- Each part is a thread.
- Each thread defines a separate path of execution.
- Multithreading is a specialized form of multitasking
- 2 distinctive types of multitasking
  - Process-based
  - Thread-based

Multithreaded Programming - Fundamentals 101

1. Process-based: this is the entire program executing and as such process-based multitasking is the ability for the same program to execute more than once concurrently and independently.
2. Thread-based: the smallest unit of code that can be dispatched by the scheduler to be executed. This implies that a single program can execute more than one task at any given time.
3. Java controls thread-based programming, but not process-based programming since this is under the responsibility of the OS and the architecture of the machine.

Multithreaded Programming - Fundamentals 101

1. The main advantage of multithreading is to utilize idle time present in most programs
2. In a single-core system, concurrently executing threads share the CPU, with each thread receiving a slice of CPU time.
3. This means that in a single-core system, 2 or more threads are not actually running at the same time, but idle CPU time is utilized
4. In a multicore/multiprocessor systems, it is actually possible that 2 or more threads will execute simultaneously. This can increase the performance of certain operations in your app.

Idle time is when a CPU isn't executing anything

Multithreaded Programming - Fundamentals 101

1. A thread can be in one of several states
    1. Ready to Run: as soon as it gets CPU time
    2. Suspended: temporarily halt its execution
    3. Resumed: from being suspended
    4. Blocked: when waiting for a resource
    5. Terminated: the end and it cannot be resumed

- **Synchronization** is a special type of feature which allows the execution of threads to be coordinated in a certain well-defined ways.
- Java has a complete subsystem devoted to synchronization.

Multithreaded Programming - Thread Class and the Runnable Interface

- The entire multithread system is built upon the “Thread” class and its companion interface “Runnable”.
- Both are packaged in “java.lang”.
- “Thread” encapsulate a thread of execution
- To create a new thread, you will either extend “Thread” or implement the “Runnable” interface.

Multithreaded Programming - Thread Class and the Runnable Interface

- Methods most commonly used from the “Thread” class.

Method	Definition
final String getName()	Get a thread’s name
final int getPriority()	Returns a thread’s priority
final boolean isAlive()	Returns whether a thread is still running
final void join()	Waits for a thread to terminate
void run()	Entry point for the thread
static void sleep (long milliseconds)	Suspends a thread for a specified period of time in milliseconds
void start()	Starts a thread by calling its “**run()**” method

- All processes have to have at least one thread of execution which is typically called the “main thread”, because it is the one that is executed when your program begins, from the main thread we can spawn others.

Multithreaded Programming – Creating a Thread

- Create a thread by instantiating an object of type “Thread”.
- The “Thread” class encapsulates an object that is “runnable”.
- This can be done in 2 ways
  - Implement the “Runnable” interface, or
  - Extend the “Thread” class

See live code in STS “ThreadCreation”

Multithreaded Programming – Creating a Thread – Improving the code…

- Thread could being execution as soon as it is created
- No need to store the name of the thread since we can pass it in the constructor

See live code in STS “ThreadCreationImproved”

Multithreaded Programming – Extending Thread

1. Extending Thread is “the” other way to instantiate thread objects.
2. Must override the run() method when extending
3. Must call start() to begin the execution
4. Can override all other methods, but typically never required.

See live code in STS “ExtendingThread”

Multithreaded Programming - Multiple Threads

1. You can spawn as many threads as you need.

See live code in STS “MultipleThreads”

Multithreaded Programming - When does a Thread end?

- We can use the isAlive() method to determine when a thread is going to expire.

See live code in STS “AreThreadsAlive”

Multithreaded Programming - Thread Priorities

- Each thread is associated with a priority setting.
- The priority of the thread determines in part how much CPU time it can receive relative to other threads.
- Low priority receive little attention
- High priority receive much attention
- Other factors can influence a thread's ability to execute, for example, if one is waiting for a resource to be made available, it will be blocked until it is ready.
- Operating systems also affect the way threads are scheduled.

Multithreaded Programming - Thread Priorities

- Child thread are always started to a priority equal to its parent.
- You can change a thread's priority using the "setPriority()" method which is a member of the "Thread" class.
- Values can be 1 to 10, max being 10.
- Normal Priority is 5
- Constants
  - Thread.HIGH_PRIORITY = 10
  - Thread.LOW_PRIORITY = 1
  - Thread.NORM_PRIORITY = 5
- "getPriority()" returns the current setting of a thread.

See live code in STS "ThreadPriority"

Multithreaded Programming - Synchronization

- With multiple threads, there may be a need to coordinate the activities of two or more.
- This is called “synchronization”.
- Access to a shared resource is often a reason because it may be available to only one single thread at a time.
- Writing to a specific file for example could only be done by a single thread and until it has finished with the file “close()” then other threads cannot access said file

Multithreaded Programming - Synchronization

- The key to synchronization is to monitor which control access to an object.
- Monitors work by implementing the concept of “lock”.
- A locked object by a thread becomes unavailable to other threads.
- When a thread exits, the object is unlocked and become available.

Multithreaded Programming - Synchronization

- All objects in Java have a monitor, this is built-in into the language
- All objects can be synchronized.
- Synchronization is supported with the keyword “**synchronization**”
- Since this feature was built-in to the language, it is often nearly transparent to implement.
- There are 2 ways to synchronize your code:
  - Using Synchronized Methods
  - Using the “synchronized” Statement

Multithreaded Programming - Synchronized Methods

- A method can be defined using the synchronized keyword.
- When the method is called, the calling thread enters the object's monitor which then locks the object.
- No other thread can unlock this object
- When the thread returns, the monitor will unlock the object
- Synchronization is achieved with virtually no programming effort on your part.

See live code in STS "SynchronizedMethods"

Multithreaded Programming - “synchronization” block statement

There are cases where you want to synchronize a method which doesn’t have the “synchronized” modifier.

Could be you are using a third party package and you do not have access to the source code.

So, in this case, we must create a “synchronized” block.

See live code in STS “SynchronizedBlockStatement”

Multithreaded Programming - Thread Communication

- Sometimes we need control over our threads.
- And in order to do so, there may be situations where threads need to be able to communicate.
- Thread communication is built-in into all Java objects.
- Java supports interthread communication
- Methods wait(), notify() and notifyAll() are all implemented in the “Object” class and are the means that provide interthread communication.

Multithreaded Programming - Thread Communication

- “wait()”: when a thread is temporarily blocked from running, it calls the wait() method, this cause the thread to go to sleep and the monitor for that object to be released, allowing another thread to use the object.
- A sleeping threat is awakened when some other thread enter the same monitor and calls “notify()” or “notifyAll()”.
- “wait()”, “notify()” and “notifyAll()” must be used in a “synchronization” block.

See live code in STS “ThreadCommunication”

More Info: https://www.geeksforgeeks.org/inter-thread-communication-java/

Multithreaded Programming - Suspending, Resuming and Stopping Threads

Before Java 2

- Originally before Java 2 we could control execution of threads using
  - “resume()”
  - “suspend()”
  - “stop()”
- Problem is that “suspend()” could cause deadlock since there could be concurrency issue at play, the “communication” with other threads wasn’t implemented as efficiently

Flag variables in programming are also called state variables, their purpose is to be used to determine the state of operations, often the variable is shared across methods, might be a “parent” resource.

Since Java 2

- Design the “run()” method to periodically check a flag which determines if it should suspend, resume or stop it’s own execution
- Often we create 2 flag variables, 1 for suspend/resume and 1 for stop.
- For suspend/resume as long as the flag is set to “running” the “run()” method will continue to let the thread execute
- If that variable is set to “suspend”, the thread is “paused”
- For the stop flag, if it’s set to “stop” then the thread must terminate

See live code in STS “ThreadControl”