Day 1

Monday, August 15, 2022 12:52 PM

Threads and Concurency

Concurency: breaking up a task and executing each part independently in any order adverse affects

Threads: a subset of a process that can be executed concurently

Threads of the main process will run in the same memory space
- But they will be handled indepently

Every thread has its own call stack where it can store local variables

However they share the same heap space so they can access the same objects in memory

Multithreading: the idea of using multiple threads in an application to achieve parallelism

Each thread can perform is own task

In Java we will achieve this with the Thread class or the Runnable interface
- Typically you will recommended to use libraries to multithread for you

States of Threads: threads will follow a lifecycle with 6 states

1: New: new thread, not running yet
2: Runnable: either ready to run or running corrently
3: Blocked: waiting to acquire a lock/waiting for permission to use a resource
4: Waiting: waiting for some thread to finish logic
5: Timed Waiting: same as above but theres a time limit
6: Terminated: it is done executing

Thread Priorities: signify a special ordering a thread should execute

To set the priority you can use either static variables
in the Thread class, or specify your own

MIN_PRIORITY = 1
NORM_PRIORITY = 5 (default)
MAX_PRIORITY = 10

Thread Problems / Challenges

If you attempt multitreading yourself you could possibly
run into one of these issues

Deadlock: when two or more threads are blocked
trying to acess the same resource
- halts the program indefitely

Livelock: except instead of waiting on a lock
the states of the processes involved constantly keep
changing in regard to one another
- neither are able to move on halting the program

We combat these issues with Synchronization

Synchronization: the capability to control the access of multiple
threads to a single shared resource

This done with the synchronized keyword

This enforces only one thread can access the resource
at a time

We can tell other threads when the resource is opem with
wait() and notify()

The Producer Consumer Problem

There is a fixed sized buffer that is shared

Producer adds to the buffer
Consumer takes away from the buffer

The problem is, data should only be added if the buffer
is not full
Data should only be consumed if the buffer is not
empty

Data should only be consumed if the buffer is not empty