CS F213 - Object Oriented Programming

J. Jennifer Ranjani

email: jennifer.ranjani@pilani.bits-pilani.ac.in

Chamber: 6121 P, NAB

Consultation: Appointment by e-mail

https://github.com/JenniferRanjani/Object-Oriented-Programming-with-Java

BITS Pilani

Pilani Campus

Multithreaded Programming

Multitasking

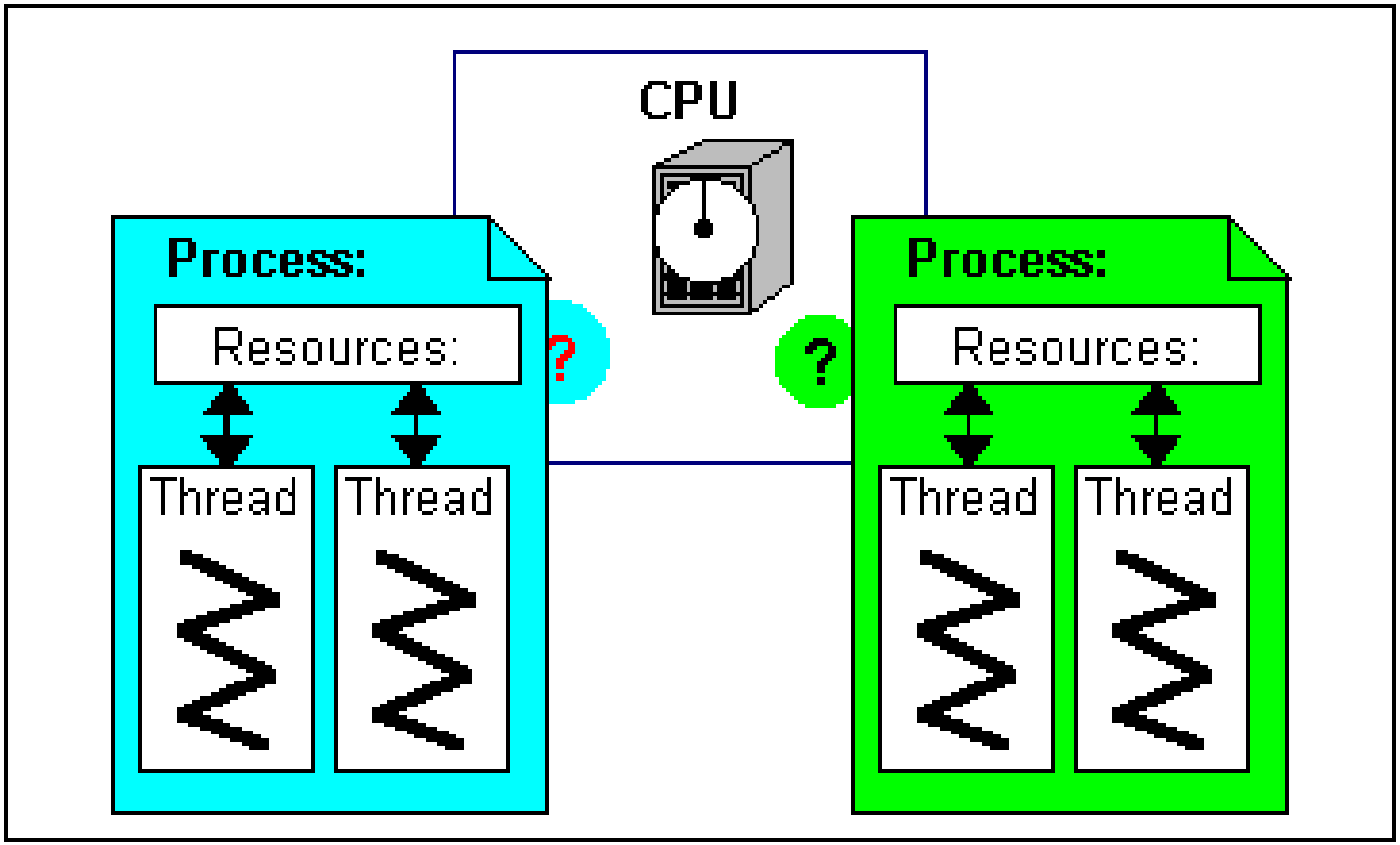
Process based

- Program in execution is called as a process
- Two or more programs running concurrently.
- Eg. Browsing and listening to music

Thread based

- Thread is a part of the program that has separate path of execution
- A single program that can perform two or more tasks simultaneously.
- Eg. Formatting using a text editor at the same time it is printing.

Process vs. Thread



Thread Model

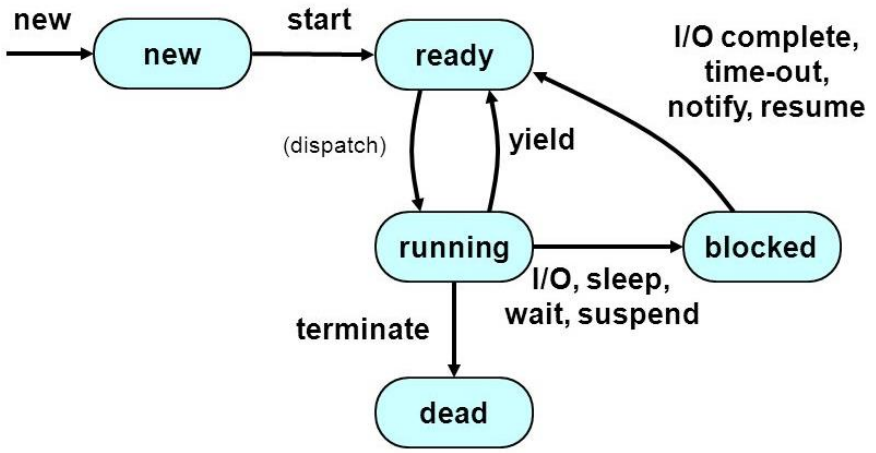
- All Java class libraries are designed with multithreading in mind.

- Single threaded systems use event loop with polling
  - Threads run a infinite loop
  - It polls a single event queue, let say, waiting for a network file to be read
  - The program wait until the event handler returns which wastes the CPU time.
  - When a thread blocks for a resource, entire program stops running

- Java Multithreading
  - Eliminates loop/polling mechanism
  - One thread can pause without stopping the other parts of the program
  - Eg. It allows animation loops to sleep for a second without causing the whole system to pause
  - One thread that is blocked pauses.

Thread States

- ***Ready to run (New):*** First time as soon as it gets CPU time.
- ***Running:*** Under execution.
- ***Suspended:*** Temporarily not active or under execution.
- ***Blocked:*** Waiting for resources.
- ***Resumed:*** Suspended thread resumed, and start from where it left off.
- ***Terminated:*** Halts the execution immediately and never resumes.



Thread Priorities

- Priorities determine how thread should be treated with respect to the others
- Priorities are integers that specify relative priority of one thread to another.
- Higher priority does not mean that the thread runs faster.
- When switching from one thread to the next, the priority is used for deciding which one to choose next – context switch

Rules determining Context Switch

innovate achieve lead

- A thread can voluntarily relinquish control
  - When explicitly yielding, sleeping or when blocked
  - The highest priority thread that is ready to run is given the CPU
  - Non-preemptive multitasking
- A thread can be preempted by a higher priority thread
  - When a lower priority thread that does not yield the processor is simple preempted by a higher priority thread no matter what it is doing
  - As soon as the higher priority thread want to run, it does
  - Preemptive Multitasking
- Some operating systems, time slice equal priority threads in round robin fashion. For others, thread should voluntarily yield otherwise it will not run.

Main Thread

- When Java program starts, the main thread starts running immediately
  - It is the thread from which other threads are spawned
  - Often, it is the last thread to finish execution
  - It is created automatically but it can also be controlled through a Thread object
  - **currentThread()** can be used to obtain a reference to the main thread

Creating a Thread

It can be created in two ways

- Implementing the **Runnable** Interface
  - By creating a class that implements the Runnable interface
  - Only a single method **run()** need to be implemented
  - Inside run(), define the code that the thread needs to perform
  - run() method can call other methods, use other classes and declare variables

- Extending the **Thread** class
  - By creating a new class that extends the **Thread** and then by creating the instance of the class
  - The extending class must override the **run()** method which is the entry point for the new thread

- Call to **start()** begins the executions of the new thread

Choosing an Approach

- It is best to implement Runnable, if we are not overriding any of the other methods by the Thread class .

- When you inherit Thread class if will not be allowed to extend any other class.

Using isAlive() & join()

innovate achieve lead

- Mostly we want the main thread to finish last
- It is accomplished by calling sleep() within main() with a long delay to ensure that all the child threads are terminated prior to the main thread
- Question: How will main know when the child terminates?
- isAlive() – determines whether a thread has finished; returns true is the thread is still running
- join() – this method waits until the thread on which it is called terminates.
  - Maximum amount of time we want a thread to wait can also be specified.

Thread Priority

- Priority is represented by a number between 1 and 10
- 3 Priority constants are defined in Thread class
  - public static int MIN_PRIORITY - 1
  - public static int NORM_PRIORITY - 5
  - public static int MAX_PRIORITY - 10
- Methods
  - final void setPriority(int level)
  - final int getPriority()

Difference between the sleep and yield methods

- Similarity
  - Both are static methods and operate on the current thread
  - Both get CPU back from the thread to thread scheduler
  - Both relinquish CPU from current thread, but does not release any lock held by the thread.
  - If the locks are to be released along with the CPU, then use wait() method.

- Difference
  - Sleep is more reliable because when yield is used there is a possibility of same thread getting the CPU
  - It is advisable to use Thread.sleep(1) instead of yield.

Synchronization

- When multiple threads try to access the same resource they often produce erroneous or unforeseen results.
- Synchronization makes sure that only one thread can access the resource at a given point of time.
- Only one block enters the synchronized block at a time
- Thread can own a monitor, when it acquires a lock it is said to have entered the monitor, all the other threads attempting to enter the monitor are suspended until the thread inside exits.

Synchronized statements

- Creating synchronized methods within classes provides easy and effective way of achieving synchronization

- If we want to synchronize objects of a class that was not designed for multithreaded access.
  - The class does not have synchronized methods
  - The class is provided by third party and we don't have access to the code

- Solution: Put call to the methods inside a synchronized block

Interthread Communication

innovate achieve lead

- Synchronized blocks unconditionally blocks all the other threads from asynchronous access
- More subtle level of control can be achieved through interprocess communication
- Multithreading is used to replace polling. Polling is implemented by a loop that is used to check a condition repeatedly. Once the condition is true, appropriate action is taken. This wastes CPU time.
  - Eg. Producer / Consumer problem

Methods for Inter process communication

- wait() – tells the calling thread to give up the monitor and go to sleep until some other thread enters the monitor and calls notify() or notifyAll()
- notify() – wakes up the thread that called wait() on the same object
- notifyAll() - wakes up all the threads that called wait() on the same object. And one will be granted access

Suspend, Resume, Stop

- suspend(), resume(), stop() – are used to pause, restart and terminate a thread
- But, they are deprecated because they can sometimes cause serious failures.
- wait() and notify() methods can be used to suspend and resume thread using a Boolean flag