CSE201: Monsoon 2020,
Advanced Programming

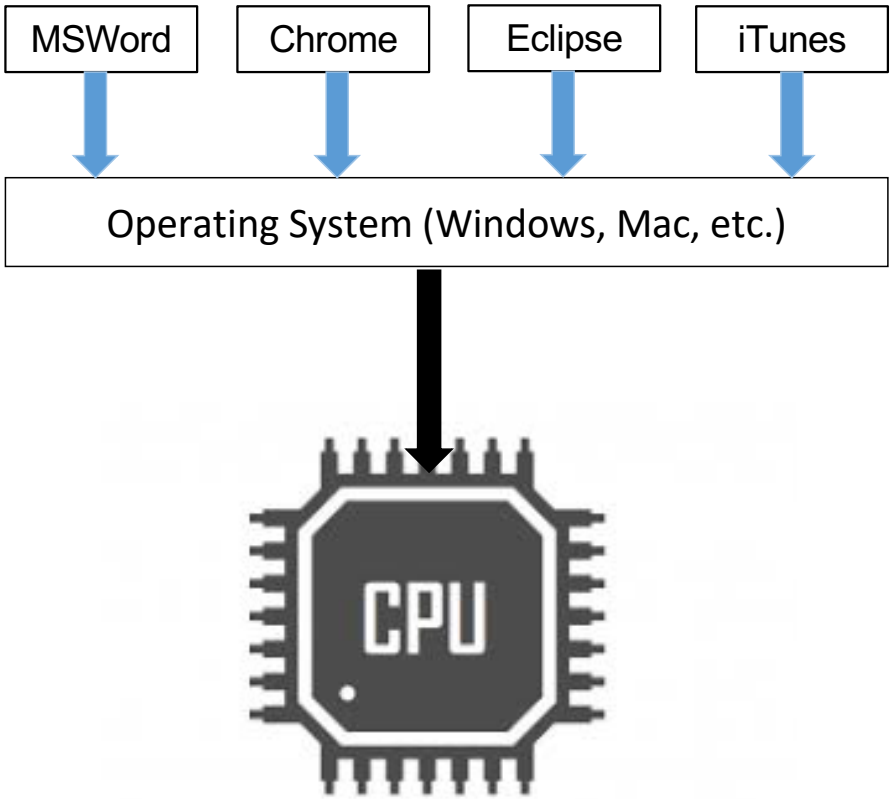
Lecture 17: Intro to Multithreading

Vivek Kumar
Computer Science and Engineering
IIIT Delhi
vivekk@iiitd.ac.in

Today's Lecture

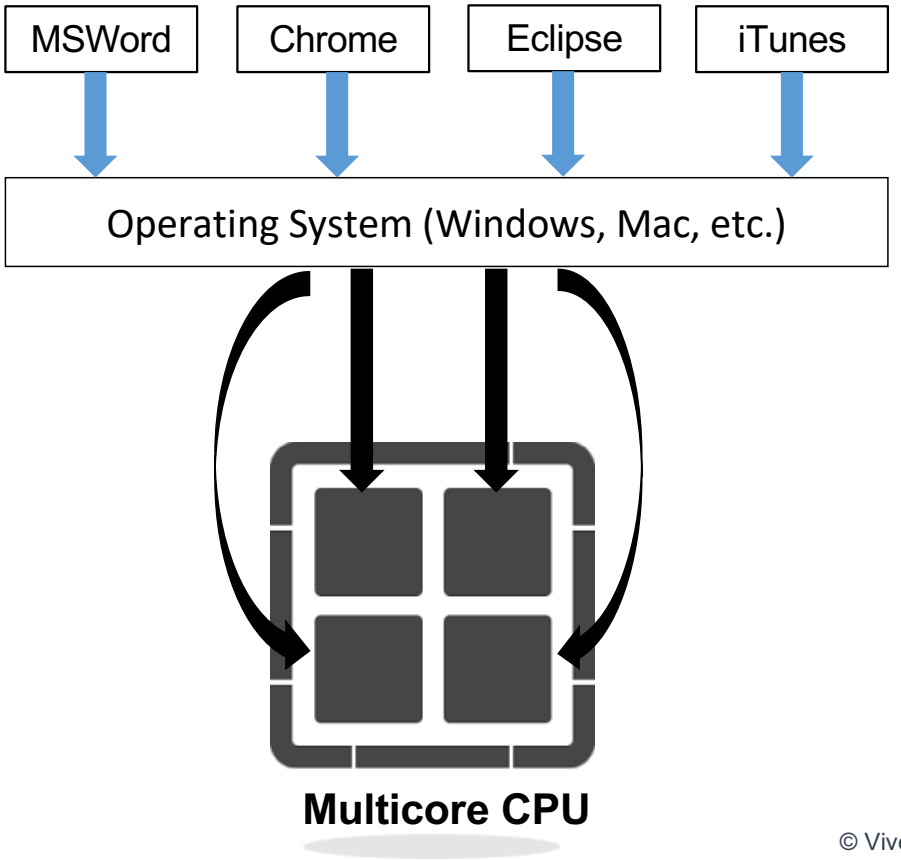
- Introduction to processes and threads

Your Laptop is Multitasking (1/2)



- Multitasking allows many more tasks to be run than there are CPUs
- In the case of a computer with a single CPU, only one task is said to be running at any point in time, meaning that the CPU is actively executing instructions for that task
- Multitasking solves the problem by scheduling which task may be the one running at any given time, and when another waiting task gets a turn

Your Laptop is Multitasking (2/2)

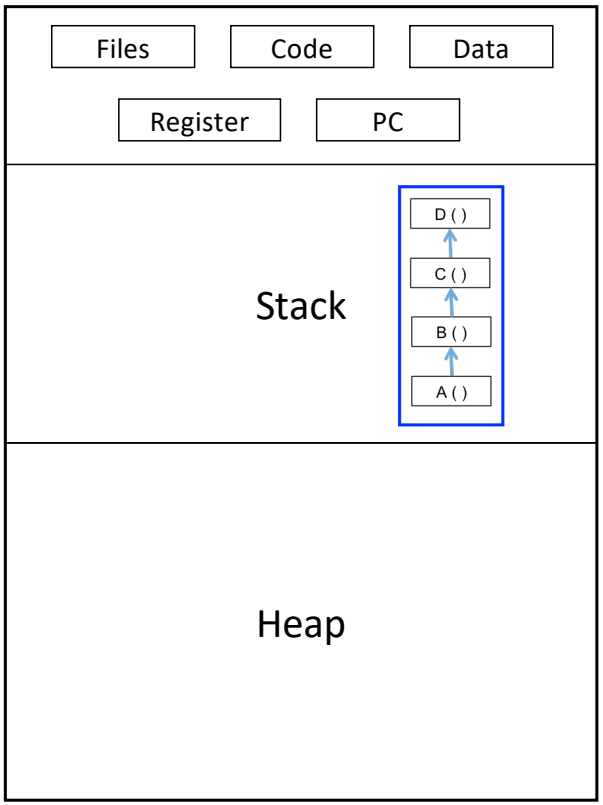


- In case of multicore processor each core will be running one task
- Here too OS will be multitasking by deciding which task runs on which core

Process

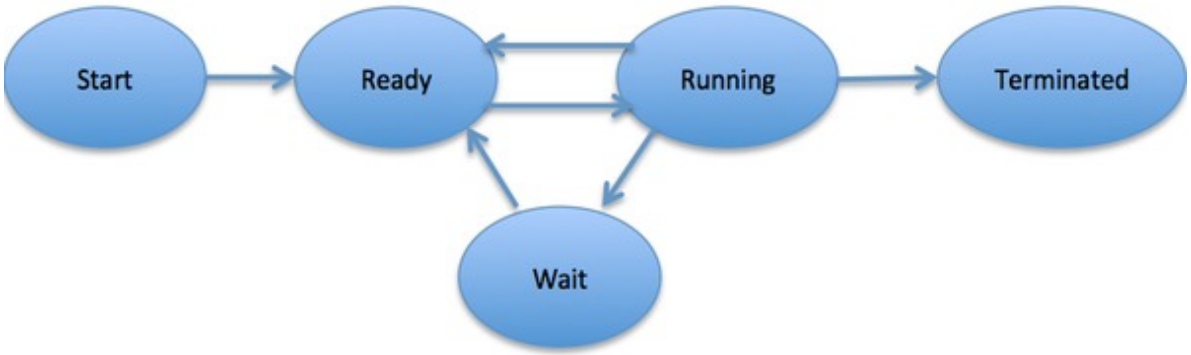
- Process is a program in execution
  - i.e., an icon of M.S. Word on your desktop isn't a process. It would become a process once you have launched it
- Each process has its own address/memory space
- Separate processes don't have access to each other's memory space
  - O.S. can help in creating a communication channel between processes if there be any need

Process Structure



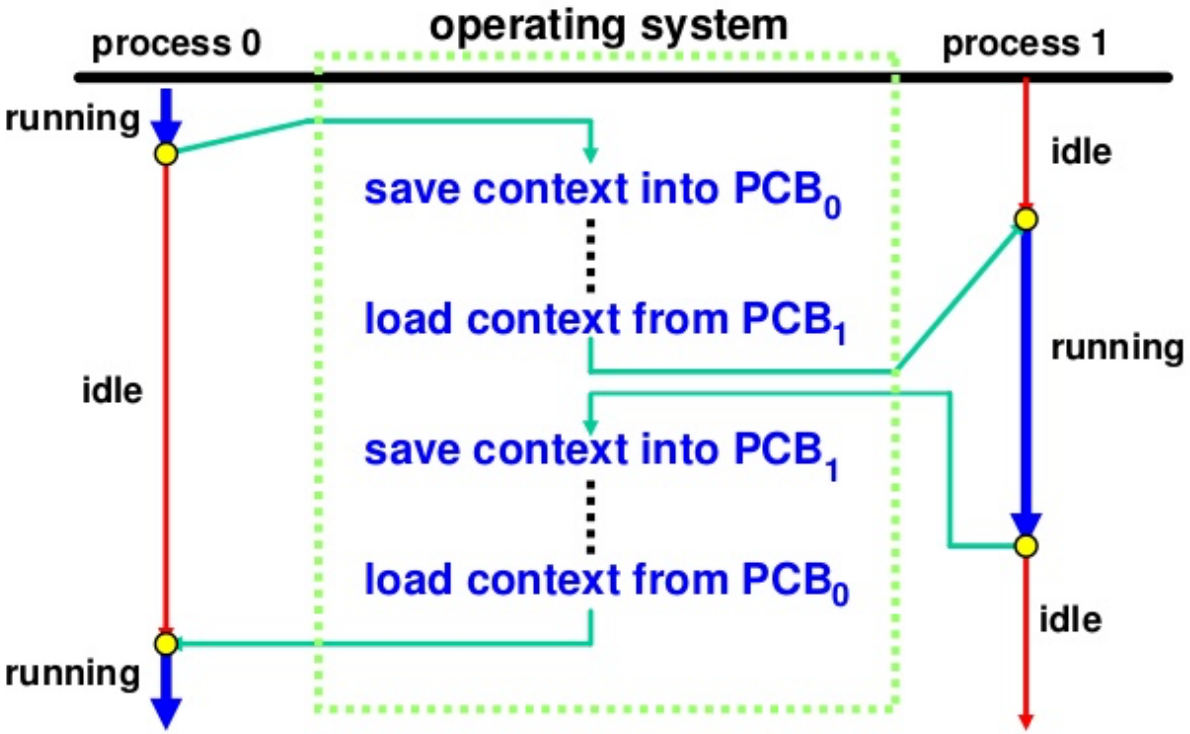
- Process contains:
  - Code
    - Program instructions
  - Data
    - Global variables in program
  - Program counter (PC)
    - Address of currently executing program instruction
  - Registers
    - CPU registers
  - Stack
    - Local variables
    - caller-callee relationship between function

Process Lifecycle



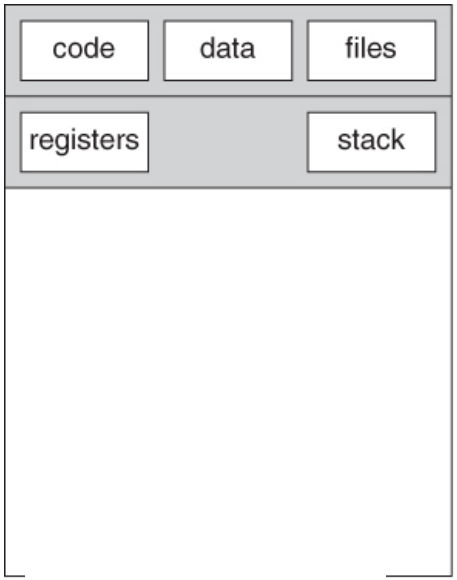
- Diagram on left shows process life-cycle
  - New – process being created
  - Ready – waiting for a free processor
  - Running – instructions are executing
  - Waiting – waiting for some event (I/O, etc.)
  - Terminated – execution is completed

Context Switch



- Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process
- A Process Control Block (PCB) is a data structure maintained by the Operating System to keep track of processes (e.g., state, id, CPU registers etc.)

Thread – A Lightweight Process

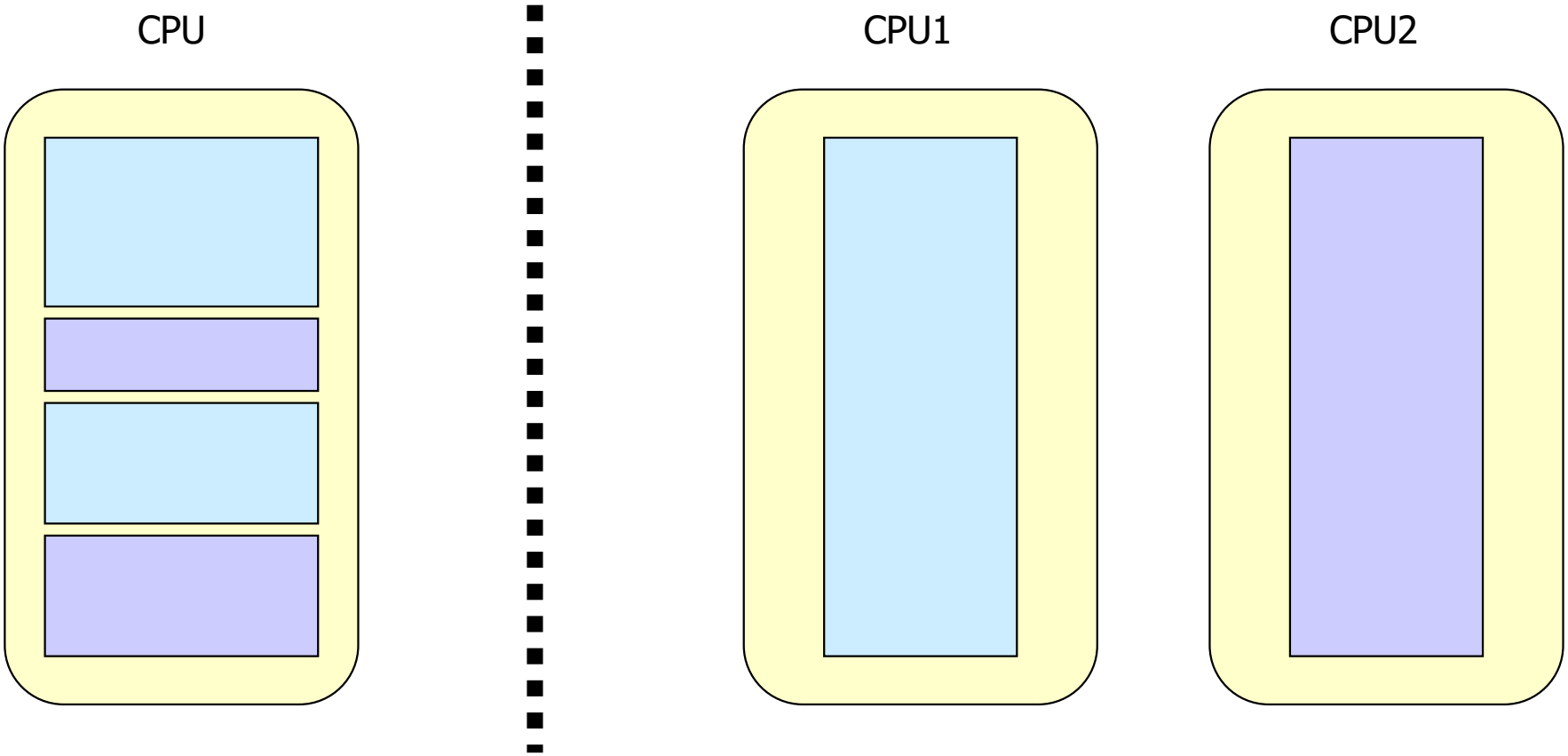


- Processes are heavyweight
  - Personal address space (allocated memory)
  - Communication across process always requires help from Operating System
- Threads are lightweight
  - Share resources inside the parent process (code, data and files)
    - Easy to communicate across sibling threads!
  - They have their own personal stack (local variables, caller-callee relationship between function)
    - Each thread is assigned a different job in the program
- A process can have one or more threads

Uses of Threads in Java

- Video game application (one process)
  - one thread for graphics
  - one thread for user interaction
  - one thread for networking with peers on internet
- Parallel programming
  - Speeding up the program execution by using multiple threads
  - A program to find the array sum can be made faster by subdividing the array among **`N`** threads (total indices at each thread = **`array.length/N`**). Each thread will find its local sum and later these local sums will be combined to find the total sum
- Producer-consumer type applications
  - News reporter (producer) sends report to new agency and one or more office clerks (consumers) will process each items and upload to Twitter, Facebook, Website, or telecast on TV
- Client-server type applications
  - Server could create a new thread for listening to a new client

Concurrency vs. Parallelism



Advantages of Multithreading

- Responsiveness
  - Even if part of program is blocked or performing lengthy operation, multithreading allows the program to continue
- Economical resource sharing
  - Threads share memory and resources of their parent process which allows multiple tasks to be performed simultaneously inside the process
- Utilization of multicores
  - Easily scale on modern multicore processors

Class Thread in Java

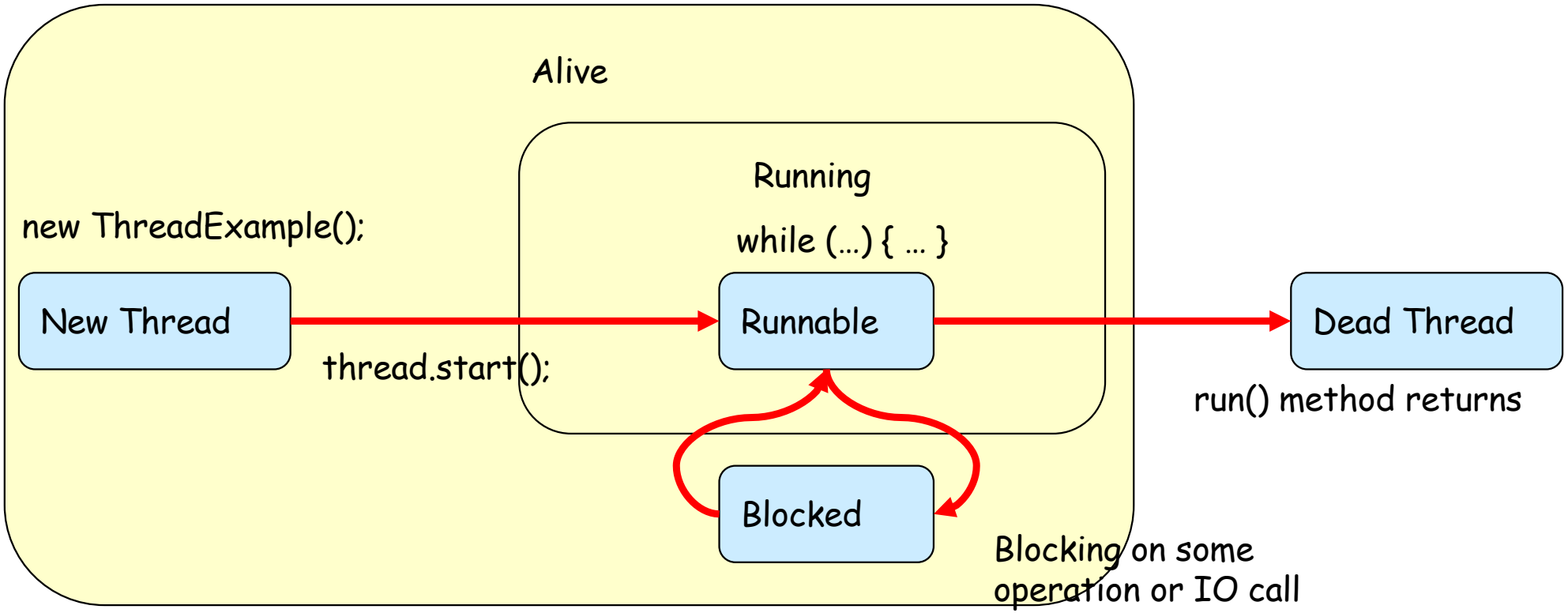
Constructors of Thread class

1. **Thread** ( )
2. **Thread** (*String str*)
3. **Thread** (*Runnable r*)
4. **Thread** (*Runnable r*, *String str*)

You can create new thread, either by extending Thread class or by implementing Runnable interface. Thread class also defines many methods for managing threads. Some of them are,

Method	Description
setName()	to give thread a name
getName()	return thread's name
getPriority()	return thread's priority
isAlive()	checks if thread is still running or not
join()	Wait for a thread to end
run()	Entry point for a thread
sleep()	suspend thread for a specified time
start()	start a thread by calling run() method

Thread State Diagram (Lifecycle)



Thread Priority

- Every thread has a priority (or seniority)
- The highest priority runnable thread is always selected by JVM for execution above lower priority threads
- The priority values range from 1 to 10, in increasing priority
- When a thread is created, it inherits the priority of the thread that created it
- The priority can be adjusted subsequently using the **setPriority()** method
- The priority of a thread may be obtained using **getPriority()**
- Pre-defined priority constants in Thread class:
  - MIN_PRIORITY=1
  - MAX_PRIORITY=10
  - NORM_PRIORITY=5

Java Application Thread

- When we execute an application:
  - The JVM creates a Thread object whose task is defined by the **main()** method
  - It starts the thread with NORM_PRIORITY
  - The thread executes the statements of the program one by one until the method returns and the thread dies

Daemon Threads

- Daemon threads are “background” threads, that provide services to other threads, e.g., the garbage collection thread
- JVM will not exit if non-Daemon threads are executing
- JVM will exit if only Daemon threads are executing
- Daemon threads die when the JVM exits

Next Lecture

- Programming with threads in Java
  - Threads in Action !
  - A.K.A., Parallel Programming!