Multithreading

Authored by : Sangeeta Joshi

Presented by: Sangeeta Joshi

Agenda

- Thread
- Thread Scheduling
- Thread Lifecycle
- Synchronization
- DeadLock

Concurrency

- Concurrency is the ability to run *several parts of a program* or *several programs* in parallel.

- Concurrency can highly improve the throw-put of a program if certain tasks can be performed asynchronously or in parallel.

- Concurrency can be achieved through

  Multiprocessing

  or

  Multithreading

- Java provides support for Multithreading

Multiprocessing Vs Multithreading

Multiprocessing : No. of processes(programs) run simultaneously

Multithreading : It is a technique that allows a program or a process to execute many tasks concurrently (at the same time and parallel).

Which of the two is light weight ? …Why?

Thread

Thread :

- Entire task (a java process) can be divided into subtasks and those subtasks can be executed concurrently. One Thread corresponds to one subtask
- A thread is an entity within a process
- Example : MS Word : In this single process, no. of activities are carried out simultaneously like saving of file, reading input, spell-checking

- Thread is a class in java.lang package
- You can create your own thread type in two ways

  1. extend from Thread class

  or

  2. implement Runnable interface

- Constructors of Thread class:

  Thread()

  Thread (String name)

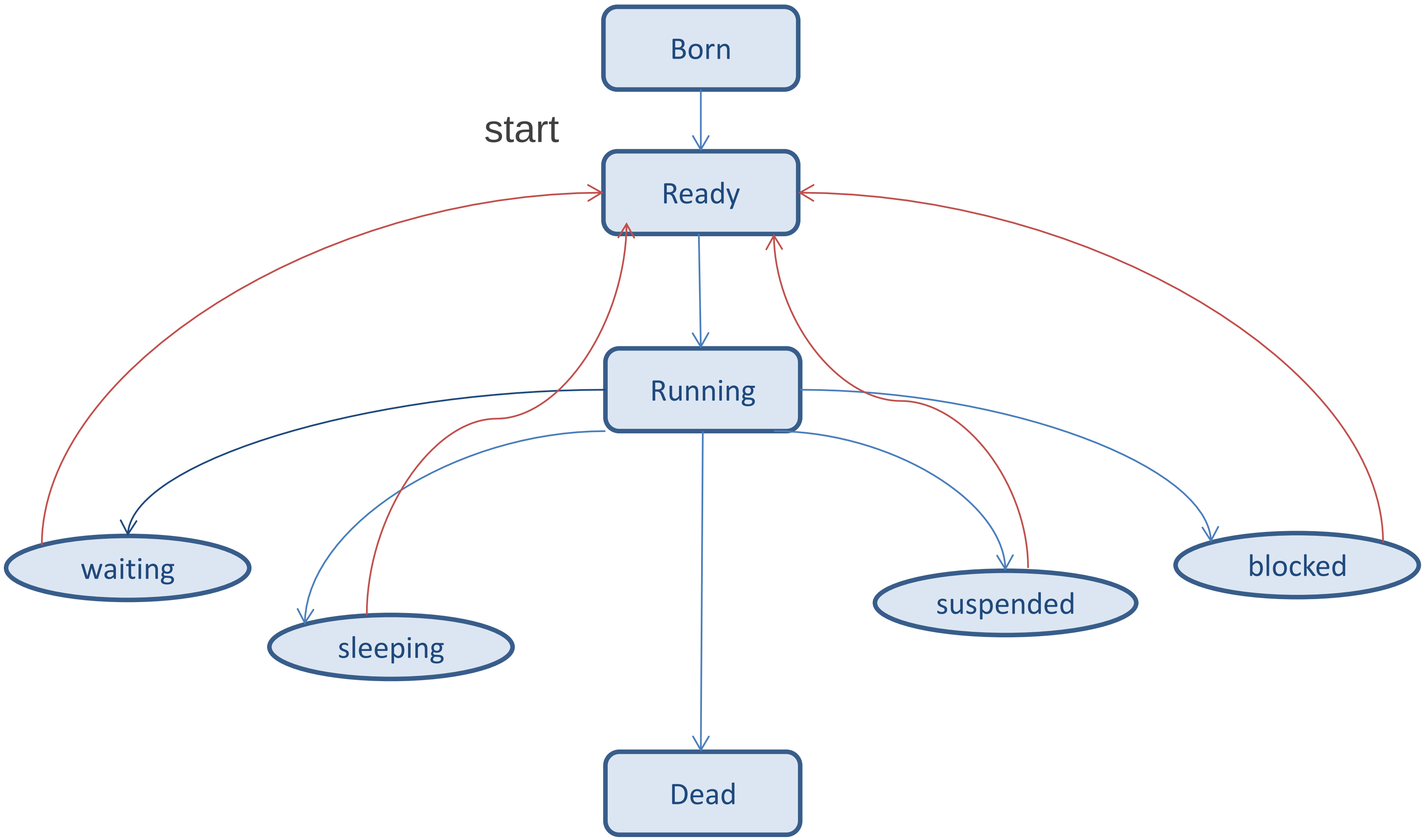
  Thread (Runnable r, String name)

  Thread (ThreadGroup tg)

Scheduling

- Scheduling is decided by Operating System.
- JVM runs on top of Operating System
- Java depends on Operating System for Thread Scheduling
- Hence multithreaded programs may show inconsistent behavior across different platforms

Lifecycle of a thread



Thread class methods

- The Thread class defines a number of methods useful for thread management.
- These include static methods, which provide information about, or affect the status of, the thread invoking the method.
- The other methods are invoked from other threads involved in managing the thread and Thread object.

Thread class methods continued…

- Thread.sleep : causes the current thread to suspend execution for a specified period.
  This is an efficient means of making processor time available to the other threads of an application

- t.join : This blocks the current thread from becoming runnable until after the thread referenced by t is no longer alive.
  The join method allows one thread to wait for the completion of another

- Thread.yield : makes the currently running thread head back to runnable to allow other threads of the same priority to get their turn
  the intention is to promote graceful turn-taking among equal-priority threads.

Thread Priorities

- Threads always run with some priority, usually represented as a number between 1 and 10

- Thread priority can be set using method setPriority(x)

- Thread class has the three following constants (static final variables)

  Thread.MIN_PRIORITY (1)

  Thread.NORM_PRIORITY (5)

  Thread.MAX_PRIORITY (10)

Synchronization

Multithreading may introduce some kind of errors :

thread interference

and

memory consistency errors

The tool needed to prevent these errors is ***synchronization***

Synchronization

thread interference :

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.

memory consistency errors:

Memory consistency errors occur when different threads have inconsistent views of what should be the same data

Synchronization

- Synchronization is built around an internal entity known as the *intrinsic lock* or *monitor lock*
- Every object has an intrinsic lock associated with it.
- By convention, a thread that needs exclusive and consistent access to an object's fields has to *acquire* the object's intrinsic lock before accessing them, and then *release* the intrinsic lock when it's done with them.
- A thread is said to *own* the intrinsic lock between the time it has acquired the lock and released the lock.
- As long as a thread owns an intrinsic lock, no other thread can acquire the same lock. The other thread will block when it attempts to acquire the lock.

Synchronization

- ***Locks In Synchronized Methods***:

  When a thread invokes a synchronized method, it automatically acquires the intrinsic lock for that method's object and releases it when the method returns.

  The lock release occurs even if the return was caused by an uncaught exception.

- ***Synchronized Statements:***

  Another way to create synchronized code is with *synchronized statements*.

  Unlike synchronized methods, synchronized statements must specify the object that provides the intrinsic lock:

Assignments

Any Questions?