
MULTITHREADING
©2015 Manipal Global Education


manipalglobal
education services
INTRODUCTION
MultiThreading
2
©2015 Manipal Global Education

manipalglobal
education services

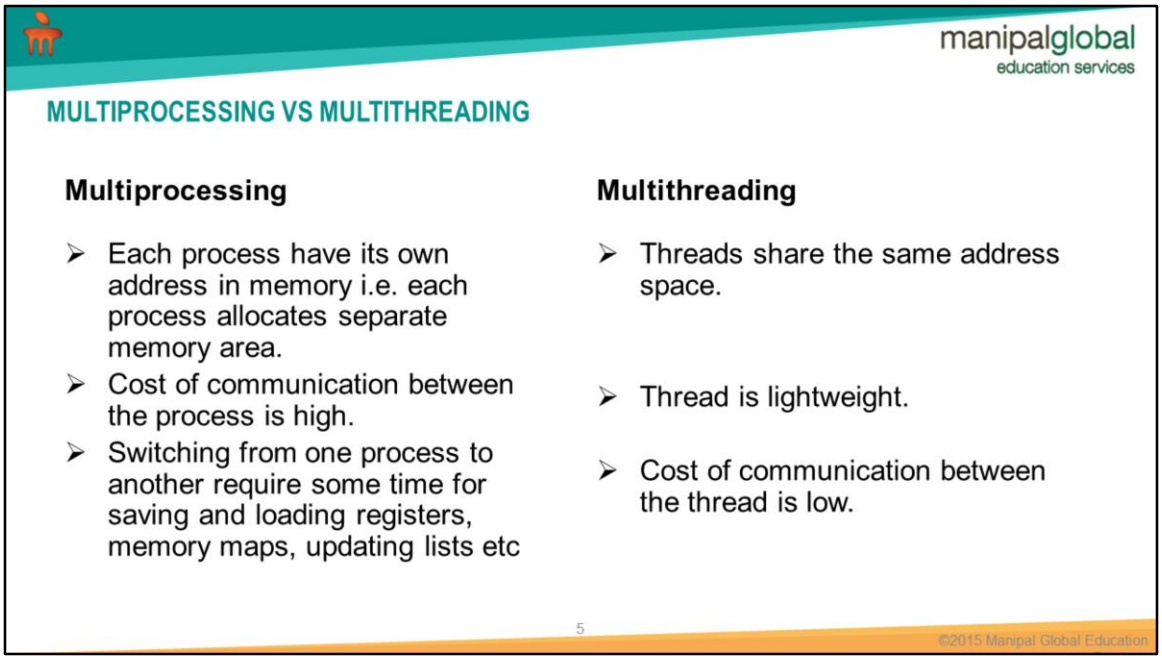
LEARNING OBJECTIVES

At the end of this lesson, you will be able to:

- Distinguish between Multi-Processing and Multi-Tasking
- Understand Lifecycle of Threads
- Understand the ways of creating Threads
- Understand synchronization
- Understand Inter-Thread Communication

BASIC CONCEPTS OF MULTITASKING

- Multitasking is the ability to execute more than one task at the same time on the same processor.
- Multitasking can be divided into two categories:
  - Process-based multitasking
  - Thread-based multitasking


manipalglobal
education services
MULTIPROCESSING VS MULTITHREADING
Multiprocessing
Each process have its own address in memory i.e. each process allocates separate memory area.
Cost of communication between the process is high.
Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc
Multithreading
Threads share the same address space.
Thread is lightweight.
Cost of communication between the thread is low.
5
©2015 Manipal Global Education

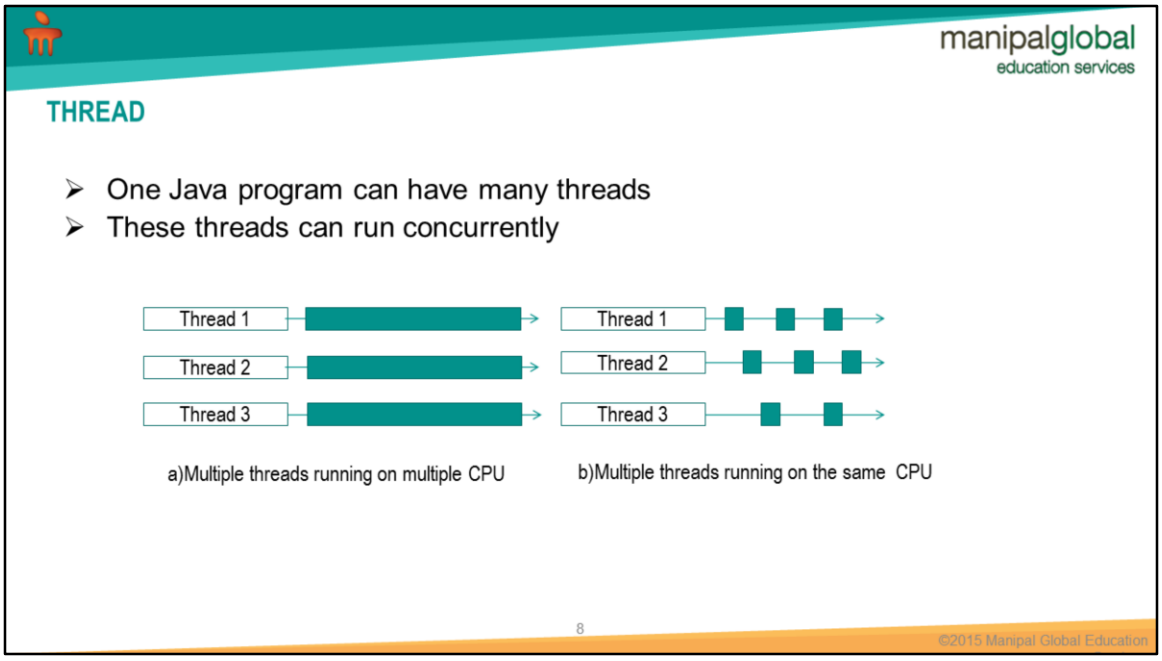
manipalglobal
education services

Disadvantage of multitasking

- **Race condition**
  - Two or more Threads access to same object
- **Deadlock condition**
  - Two or more Threads are blocked forever, waiting for each other
- **Lock starvation**
  - A Thread is unable to gain regular access to shared resources and is unable to make progress

WHAT IS A THREAD

- An independent execution path of a program
- "Thread" can be
  - An instance of class java.lang.Thread
  - A thread of execution
- An instance of Thread is an object
  - Like any other object in Java, it has variables and methods, and lives and dies on the heap
- A thread of execution is an individual process (a "lightweight" process) that has its own call stack


manipalglobal
education services
THREAD
One Java program can have many threads
These threads can run concurrently
Thread 1
Thread 2
Thread 3
Thread 1
Thread 2
Thread 3
a)Multiple threads running on multiple CPU
b)Multiple threads running on the same CPU
8
©2015 Manipal Global Education

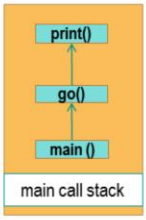
BENEFITS OF THREADS VERSUS PROCESSES

- Threads have some advantages of (multi) processes, They take less time to:
  - Create a new Thread rather than a process
  - Terminate a Thread rather than a process
  - Switch between two Threads within the same process
  - Communication overheads

main Thread

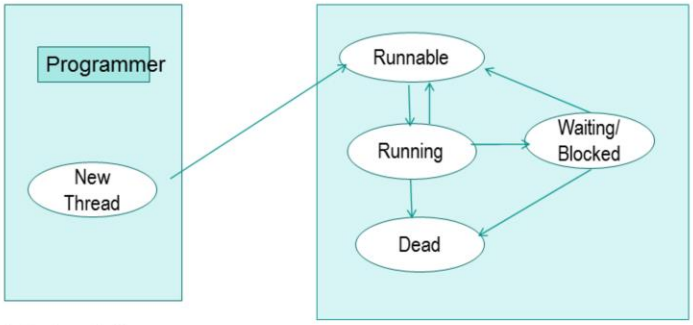
- The main method runs in one thread called the main thread

```
public class Test {
       void print() { }
       void go() {
              print();
       }
public static void main(String[] args){
       Test tx = new Test();
       tx.go();
}}
```



- In the main call stack the main() is the first method on the stack i.e. method at the bottom of stack

Life cycle of a thread



- Thread Scheduler
  - Part of the JVM that decides which thread should run at any given moment
  - Moves threads from one state to another
  - The order in which runnable threads are chosen to run is not guaranteed

manipalglobal
education services

Different states of a thread

A Thread can be only in one of the five states :

- New: Thread object created but not started
- Runnable: Enters runnable state after Thread object is made thread of execution
- Running: When the thread scheduler selects the thread from the runnable pool for execution
- Blocked or Waiting: Enter this state when waiting for resources held by other thread, sleeping, blocked etc
- Terminated or Dead: When the thread completely execution and finishes the job assigned to it
  - Once a thread is dead, it cannot be brought back to execution

manipalglobal
education services

run() method of Runnable Interface

- Thread Class implements Runnable Interface
- Runnable interface has one method run()

- run() Method
  - The job to be done by a thread is coded in the run() method
  - When a Thread starts running, it executes the code in run() method

```
public void run()
{
      //  the code goes here
}
```

- A thread in Java begins as an instance of Thread class(New State)

manipalglobal
education services

Creating a thread

Two ways to create thread in java

- **Extend the Thread class , override the run() method**

```
public class TestThread extends Thread {
 public void run() {
     System.out.println("run method defines the work to be done");
  }}
```

- **A Class implements Runnable interface ,overrides the run() method and instance of this class is passed as argument When creating Thread**

```
public class ThreadJob implements Runnable {
         public void run() {
           System.out.println("run method defines the work to be done");
       }}
```

manipalglobal
education services

Creating a thread

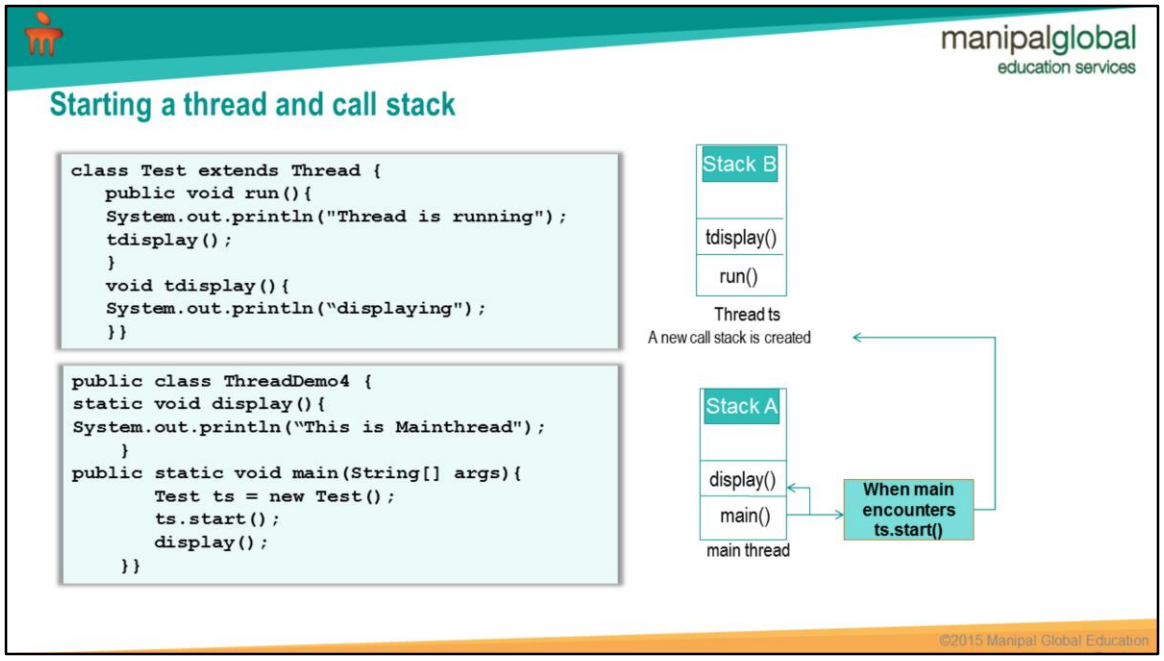
- Regardless of whether run() method is in a Thread subclass or a Runnable implementation class, a Thread object is needed to do the work

```
public class ThreadDemo{
               public static void main(String[] args){
               //Extending a thread class
              TestThread  thread= new TestThread();

              //Implementing the Runnable interface
               ThreadJob tJob = new ThreadJob();
              Thread thread = new Thread(tJob);
               }
       }
```

Starting a thread

- A Thread is started by calling the method **start()** on thread instance
  thread.start();
  where thread is an instance of Thread class
- After the Thread object is started, a new call stack will be launched
- After start() method is called:
  - A new thread of execution starts (with a new call stack)
  - The thread moves from the new state to the runnable state
  - When the thread gets a chance to execute, its target run() method will run


manipalglobal
education services
Starting a thread and call stack
class Test extends Thread {
public void run(){
System.out.println("Thread is running");
tdisplay();
}
void tdisplay(){
System.out.println("displaying");
}}
public class ThreadDemo4 {
static void display(){
System.out.println("This is Mainthread");
}
public static void main(String[] args){
Test ts = new Test();
ts.start();
display();
}}
Stack B
tdisplay()
run()
Thread ts
A new call stack is created
Stack A
display()
main()
main thread
When main encounters ts.start()
©2015 Manipal Global Education

Thread Methods

- currentThread()
  - Static Method
  - Returns a reference to the currently executing thread object
- setName(String name)
  - Sets/Changes the name of thread
- getName()
  - Returns thread's name
- isAlive()
  - Tests if thread is alive

Thread Methods

- sleep()
  - Static method of Thread Class
  - Used to "slow down a thread" by forcing it to go into sleep mode for the specified amount of time
- join()
  - If called on a thread **t**, the current thread will go to the blocked state until **t** completes

manipalglobal
education services

Thread priority

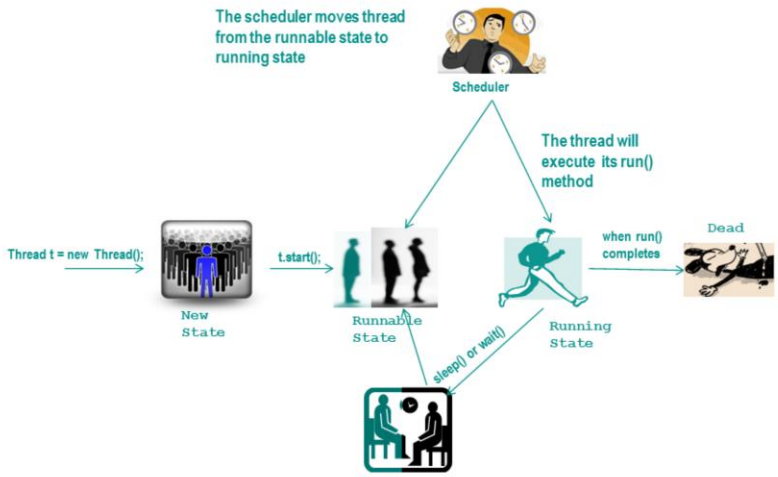
- Each thread has an assigned priority
- Thread priorities are the integers in the range of 1 to 10 that specify the priority of one Thread with respect to the priority of another Thread.
- A threads priority is set to the priority of the thread that creates it
- The two methods used to set and get the priority of a thread
  - setPriority()
  - getPriority()
- Each Java Thread is given a numeric priority between MIN_PRIORITY and MAX_PRIORITY.
- The Java Run-time system selects the runnable Thread with the highest priority of execution when a number of Threads get ready to execute.
- A Thread with higher priority runs before Threads with low priority.

YIELDING

- A given Thread may, at any time, give up its right to execute by calling the yield method
- Threads can only yield the CPU to other Threads of the same priority– attempts to yield to a lower priority Thread are ignored

- yield() - Static Method
  - Causes the current thread pause temporarily and allow other waiting threads of the same or high priority to execute
  - use yield() to promote graceful turn-taking among equal-priority threads.
  - No Guarantee

OVERVIEW



DAEMON THREADS

- Low priority threads
- Service provider for other threads or objects running
- Runs intermittently in the background doing the operations like garbage collection for the java runtime system
- Any thread can be made a daemon thread in Java
- Used for background supporting tasks and are only needed while normal threads are executing
- The setDaemon() method is used to specify that a thread is daemon thread

manipalglobal
education services

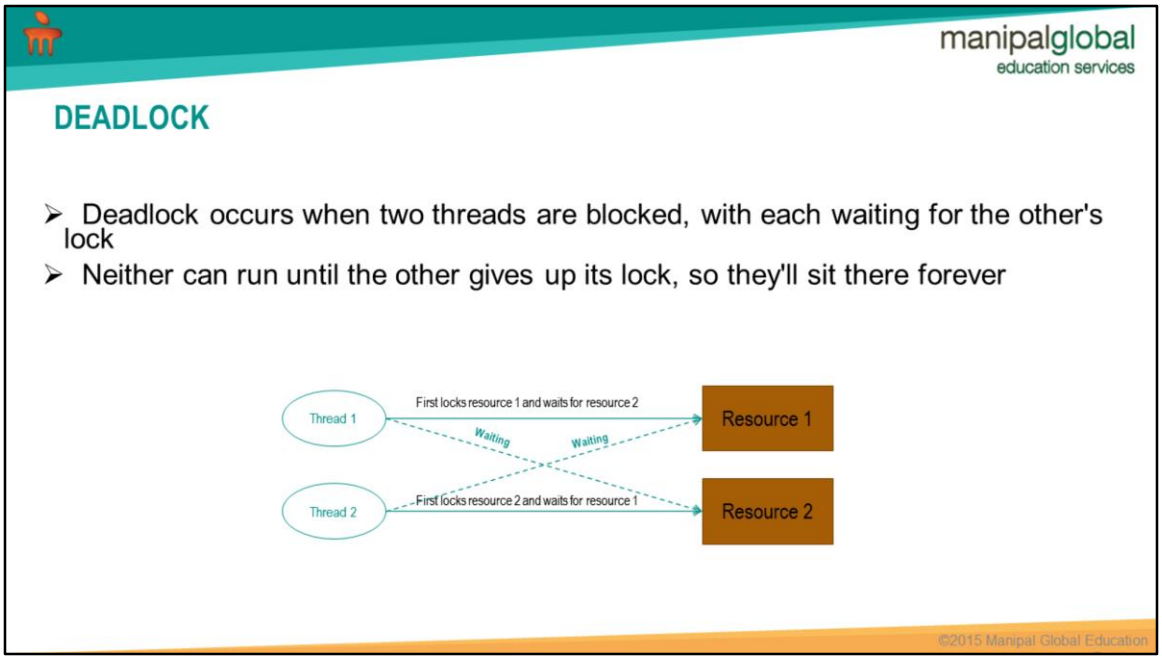
THREAD SYNCHRONIZATION

```
private synchronized void Withdraw(int amt) {
}
Is equivalent to
private void makeWithdrawal(int amt) {
                    synchronized(this){
}        //synchronize on the current instance
       }
```

- Only methods or blocks of code can be synchronized, not variables or classes
- Not all methods in a class need to be synchronized
- Two threads cannot execute a synchronized method of a class, for the same object
  - One thread will need to wait until the other finishes its method call
- If a thread goes to sleep, it holds any of the locks it has
- A thread can acquire more than one lock for different objects

THREAD SAFE CLASSES

- When a class has been carefully synchronized to protect its data, it is said to be "thread-safe“
- Many classes in the Java APIs already use synchronization internally in order to make the class "thread-safe“
- For example
  - StringBuffer and StringBuilder are nearly identical classes, except that all the methods in StringBuffer are synchronized, while those in StringBuilder are not


manipalglobal
education services
DEADLOCK
Deadlock occurs when two threads are blocked, with each waiting for the other's lock
Neither can run until the other gives up its lock, so they'll sit there forever
Thread 1
First locks resource 1 and waits for resource 2
Waiting
Waiting
Resource 1
Thread 2
First locks resource 2 and waits for resource 1
Resource 2
©2015 Manipal Global Education

manipalglobal
education services

INTER-THREAD COMMUNICATION

- Threads can interact with each other using wait(),notify() and notifyAll() methods
- These methods help threads communicate about the status of an event that the threads care about
- wait() and notify()
  - Lets one thread put itself into a "waiting room" until some other thread notifies it that there's a reason to come back out
- notifyAll()
  - Lets all the threads waiting for and event rush out of the waiting area back to runnable
- Note: wait(), notify(), and notifyAll() must be called from within a synchronized context

A thread can't invoke a wait or notify method on an object unless it owns

manipalglobal
education services

WHAT IS THREADS SCHEDULING?

- The execution of multiple Threads on a single CPU is called scheduling.
- Thread scheduling:
  - Determines which runnable Threads to run
  - Can be based on Thread priority
  - Is a part of OS or Java Virtual Machine (JVM)
- The Java runtime supports a very simple, deterministic scheduling algorithm known as fixed priority scheduling.

- Types of scheduling:
  - Pre-emptive
  - Non-Preemptive


manipalglobal
education services
SUMMARY
MultiThreading
29
©2015 Manipal Global Education