# Core Java

## Multithreading in Java

# Lesson Objectives

- **To understand the following topics**
  - Multithreading
  - Main Thread
  - Creating Threads
  - Life Cycle of Thread
  - Scheduling and Priority
  - Concurrency Issues

# What is Multithreading?

- **Single threaded program – Single line of execution**
  - They have a beginning, sequence and an end at any given time
- **Multithreading allows you to write very efficient programs that make maximum use of the CPU, because idle time can be kept to a minimum**
- **Multithreaded application – Multiple line of execution**
  - Each have a beginning, sequence and an end of its own
  - They can run in parallel

# Advantages of Multithreading

- **Maintaining user interface responsiveness**
  - separate thread for each internet user
- **Maximum throughput**
- **Simple Multitasking**
- **Building multi-user applications**
- **Multi-processing**

# Main Thread

- **When a Java program starts up, main thread runs.**
  - Child threads are spawned from this main thread.
- **Last thread to finish execution.**
  - When the main thread stops, your program terminates
  - If the main thread finishes before a child thread the Java run-time system may hang.

# Thread Class

| Method | Meaning |
|---|---|
| getName / setName | Obtain / set a thread's name |
| getPriority | Obtains a thread's priority |
| isAlive | Determine if a thread is still running |
| join | Wait for a thread to terminate |
| resume | Resume the execution of a suspended thread |
| run | Entry point for a thread |
| sleep | Suspend a thread for a period of time |
| start | Start a thread by calling its run method |
| suspend | Suspend a thread |
| currentThread | Returns a reference to the currently executing thread object. |
| interrupt | Interrupts this thread |
| yield | Causes the currently executing thread object to temporarily pause and allow other threads to execute |

# **How to create a** Thread ?

- **Two ways to create a thread:**
  - Extend Thread class
  - Implement Runnable interface

# Extend Thread Class to Create Threads

- **Class should extend Thread class.**
- **Inherited class should:**
  - Override the *run()* method.
  - Invoke the *start()* method to start the thread.

```
class PingPong extends Thread{
String word;
public PingPong(String pp){
word=pp;
}
public void run(){  //overriding the run() method
 try{
        for(;;)
{
        System.out.println(word);
        Thread.sleep(1000);
}
}
catch(InterruptedException e)
{ System.out.println("sleep interrupted");  } }
```

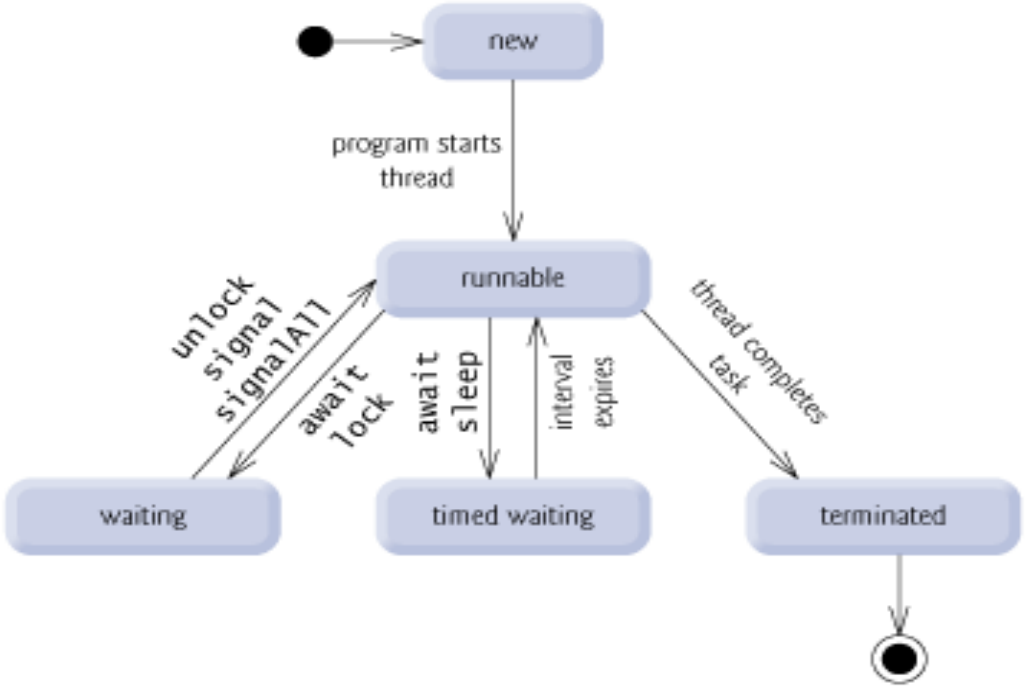
# Extend Thread Class to Create Threads

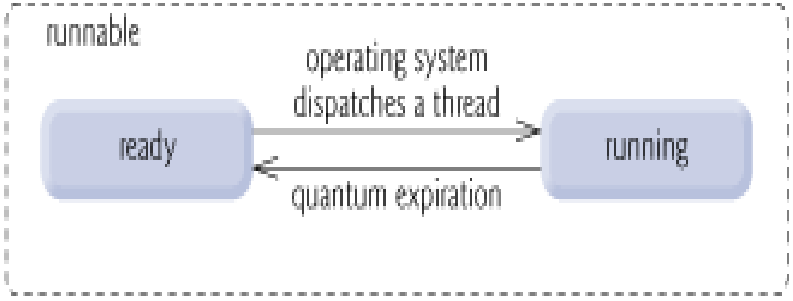
```
public static void main(String args[]){
    Thread t1 = new PingPong("ping");
    Thread t2 = new PingPong("pong");
    t1.start(); //invokes the run() method.
    t2.start();
        }
}
```

# Implement Runnable Interface

- **Easiest way to create a thread.**
  - Create a class that implements the runnable interface.
  - Class to implement only the run method that constitutes the new thread.
  - After you have created a class that implements Runnable, you will instantiate an object of type Thread within that class. Thread defines several constructors.
  - Thread ( Runnable *threadOb*, String *threadName* );
  - where *threadOb* is an instance of the class which implements the **Runnable** interface.
  - Once created, the new thread will not start running until you call its start() method, which is declared within the Thread. In turn, start() executes a call to run().

# Thread Lifecycle



# Priority of Threads

- **Threads have priorities ranging from 1 to 10.**
- **Constants that defines Thread priorities are:**
  - Thread.MIN_PRIORITY (1)
  - Thread.NORM_PRIORITY (5)
  - Thread.MAX_PRIORITY (10)
- **Default priority is 5.**
- **Do not rely on thread priorities when you design your multithreade application.**

# Multithreading Policies

- **In 32-bit Java implementations for Win '95 & Win NT, threads are time-sliced.**
  - Each thread is given a limited execution time on a processor.
  - When that time expires the thread is made to wait.
  - Other threads of *equal* priority get their chance to use their quantum in a round - robin fashion.
- **Thus, a running thread can be pre-empted by a thread of equal priority.**
- **In Solaris, a running thread can only be pre-empted by a thread of higher priority.**

# Thread Synchronization

- What happens when two threads have access to the same object and each calls a method that modifies the state of the object? Corrupted objects can result. Such a situation is called a race condition.

- For example, suppose you have a thread that writes data to a file while, at the same time, another thread is reading data from the same file. When your threads share information, you need to synchronize the threads to get the desired results.

- **Synchronization:**
  - Technique to ensure that a shared resource is used by only one thread at a time.
  - Avoids race condition.
  - Key to synchronization : concept of Object monitor. Monitor is an object that is used as a mutually exclusive lock.

```
public class counter {
private int count=0;
public int incr()  {
int n = count;
count =  n+1;
return n;
}
}
```

| Thread 1 | | Thread 2 | | Value of Count |
|---|---|---|---|---|
| cnt = counter.incr(); | | ---------- | | 0 |
| n = count; | [0] | ---------- | | 0 |
| ----- | | cnt = counter.incr() | | 0 |
| ----- | | n = count; | [0] | 0 |
| ---- | | count = n + 1 | [1] | 1 |
| | | return n; | [1] | 1 |
| count = n + 1 | [1] | -------- | | 1 |
| return n; | [0] | -------- | | 1 |

# Thread Synchronization

- **final void wait() :**
  - tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify().

- **Final void notify():**
  - wakes up the first thread that called wait()on the same object.

- **final void notifyall() :**
  - wakes up all the threads that called wait() on the same object. The highest priority thread will run first.

# Using Synchronized Blocks

- **Synchronization affects concurrency.**
- **Object Monitor**
- **If a method scope is more than needed, reduce it to just a *block:***

```
class SyncTest {

public void doMethod() {
System.out.println("not synchronized");

synchronized(this)
{
System.out.println("synchronized"); }
}
}
```