School of Computer Science and IT
JAIN (DEEMED-TO-BE UNIVERSITY)
Department of Bachelor of Computer Applications

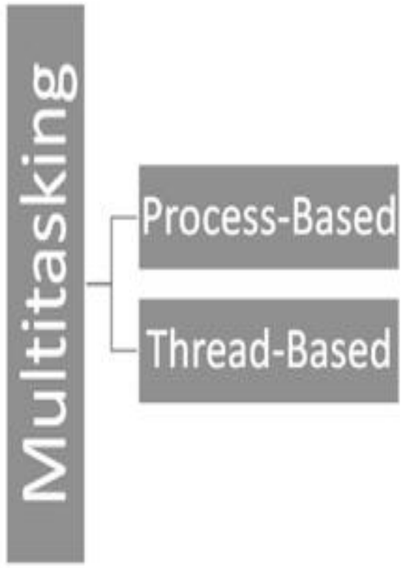
Module 4 Multithreaded Programming

Chapter 1 Introduction to Threads

Threads

Introduction to Threads

- A multithreaded program includes multiple parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution. Thus, multithreading is a specialized form of multitasking.



- In process based multitasking, a program is the smallest unit of code that can be dispatched by the scheduler.

Benefits of Multithreading

- Enables the programmers to do multiple tasks at the same time.
- Increase the speed of program execution by dividing the program into threads and execute them in parallel.
- Actions can be simultaneous across multiple application.
- It improves the performance and concurrency.

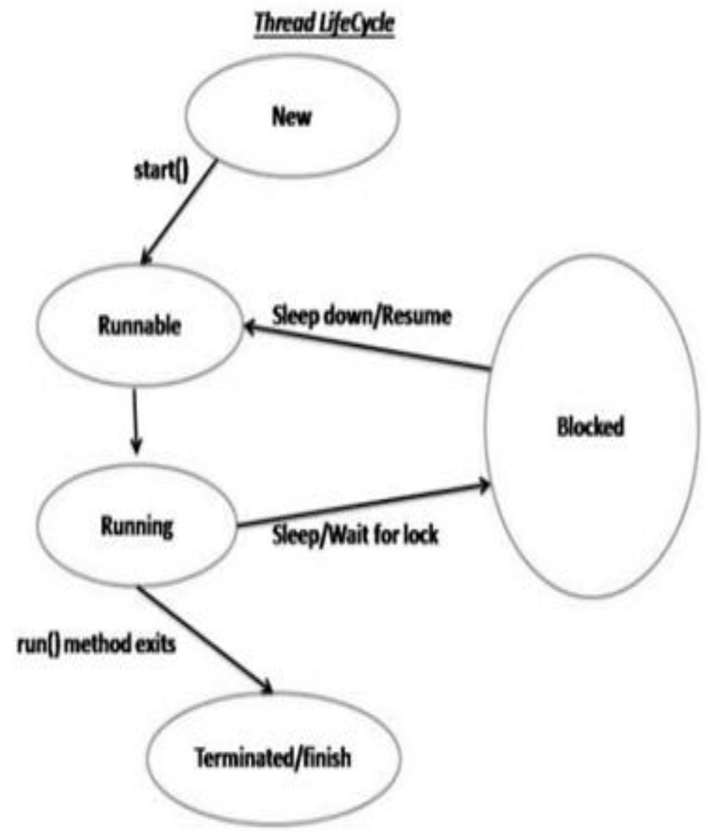
Java Thread Model

Java Thread Model

Multithreading is the main concept in Java programming. It has advanced features when compared to the single-thread model. Java's run-time system has been designed keeping a multi-thread model in mind. Let's see the life cycle of a thread.

a)Lifecycle of a Thread

Threads can be one in five states. They are:



First State-New Thread yet to Start

When coder creates a new Thread object using the new operator, thread state is New Thread. At this point, a thread is not alive and it's a state internal to Java programming.

Second State-Ready to Run the State

When we call to start() function on Thread object, it states is changed to Runnable and the control is given to Thread scheduler to terminate its execution. Whether to run this thread instantly or keep it in runnable thread pool before moving it depends on the OS implementation of thread scheduler.

Third State-Running State

- When a thread is performing, it states is changed to Running. Thread scheduler picks one of the thread from the runnable thread pool and changes it's reported to Running and CPU starts performing this thread.
- A thread can change state to Runnable, Dead or Blocked from running state depends on time slicing, thread completion of run() method or waiting for some resources.

Fourth State-Waiting/Blocked

- A thread can be waiting for the different thread to finish using thread join or it can be waiting for some resources to available.
- for example, producer consumer problem or waiter notifier implementation or IO resources, suddenly it's state is changed to Waiting.
- Once the thread wait state is over, it's state is changed to Runnable and it's moved back to the runnable thread pool.

Fifth State-Dead State

- Once the thread finished executing, it's state is changed to Dead and it's considered to be not alive.

According to Sun Microsystems, there are only four states. That is, new, runnable, non-runnable and terminated.

b)Thread Priorities

- In Java, each thread has its priority to implement itself. It helps the operating system to determine the order in which it has to be scheduled.
- When a thread moves from one running state to next, then it will decide its priority (direction of its movement depends on its priority). This process of navigating itself according to its priority is known as context switching.
- The priorities of a thread range between MIN_PRIORITY and MAX_PRIORITY. These priorities help to increase the running time of thread or execution time also.

c) Synchronization

As multithreading introduces an asynchronous behaviour to your programs, there must be a way for you to drive synchronicity when you require it.

Example:

If you need two threads to communicate and share a complex data structure, such as a linked list, you need some way to assure that they don't differ with each other. That is, you must prevent one thread from writing data while another thread is in the middle of reading it. For this purpose, Java executes an elegant twist on an age-old model of interprocess synchronization: the monitor. The monitor is a control mechanism first defined. You can think of a screen as a tiny box that can hold only one thread. Once a thread enters a monitor, all other threads must wait till that thread exits the monitor. In this way, a screen can be used to protect a shared asset from being managed by more than one thread at a time.

d) Thread Class

Java's multithreaded model is based on Thread class because it acts as a main class in the multithreaded system. They are used to design and execute the multithreading features of Java using Runnable interface.

Multithreaded features are utilised using Constructors of Threadclasses. Some of them are:

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

Methods of Thread Class

Thread class defines methods for managing threads. Some of the methods are as follows:

Method	Description
setName( )	Naming a thread
getName( )	Return thread’s name
getPriority( )	Return’s thread’s priority
isAlive( )	Checks whether the thread is currently running or not
join( )	Wait for a thread to end
run( )	Entry point of a thread
sleep( )	Suspend the thread
start( )	Start a thread by calling run( ) method

e) Runnable Interface

- Thread class can create a thread by extending its object. But only one class can be extended; it won't allow multiple inheritances to take place. So to create many inheritances, a programmer can create a thread by implementing **Runnable interface.**
- Using Runnable interface, multiple inheritances implemented at the same time.
- While implementing Runnable interface, a user need to provide an implementation of run()method.

Steps to Implement Runnable Interface

Let's look at the steps to be followed to implement Runnable Interface:

Step 1:

Create a new thread; your program will unless extending Thread or execute the Runnable interface.

Step 2:

To start executing run( ) method, call start( ) method on Thread class.

Step 3:

Implementing Runnable interface defines only an entry point for threads in the object, it does not create a Thread object. Rather it allows the user to pass the object to the Thread constructor (Runnable Implementation).

Example Program

Here is an example program of main class starts the "first thread" and "second thread":

```
public class MyThread{
public static void main(String
args[]){
FirstThread   f = new
FirstThread();       // first
thread//
SecondThread   s = new
SecondThread();  //second
thread//
 Thread thread1 = new Thread (f);
thread1.start();
 Thread thread2 = new Thread(s);
thread2.start(); }}
```

Output:

```
Message of f: 1
Message of s: 1
Message of f: 2
Message of s: 2
```

The Main Thread

Main Thread

One thread begins running immediately when a Java program starts up. It is usually called the main thread of your program because it is the one that is executed when your program begins. The main thread is relevant for two reasons:

1. It is the thread from which other "child" threads will be spawned.
2. Often, it must be the last thread to finish execution because it performs several shutdown actions.

Programmers can refer the main thread using CurretThread( ) method. General form of the main thread is
Static Thread currentThread( ); // used to return authority to the thread in which it is called.

Example-Program

```
// Controlling the main Thread.
class CurrentThreadDemo
{
public static void main(String args[]) {
Thread x = Thread.currentThread();
System.out.println("Current thread: " +
x);
// change the name of the thread
x.setName("My Thread");
System.out.println("After name change:
" + x);
try {
for(int i= 3; i> 0; i--) {
System.out.println(i);
Thread.sleep(500);
}
} catch (InterruptedException e)
{
System.out.println("The main
thread has interrupted");
}
}
}
```

Example-Explanation

- In this preceding program, a source of the current thread is obtained by calling currentThread( ), and this reference is stored in the local variable x. Next, the program displays information about the thread.
- The program then calls setName( ) to change the internal name of the thread. Information about the thread is then re-open. Next, a loop counts down from five, pausing one second between each line.
- The argument to sleep( ) method specifies the delay period in milliseconds.Notice the try/catch block around this loop. Here is the generated output by this program:
- Current thread: Thread[main,3,main]
- After name change: Thread[My Thread,3, main]

Creating Threads

The thread is processed in two ways. They can be achieved by

- Implementing Runnable Interface
- Extending the Thread class

Implementing Runnable

Implementing Runnable Interface is the simplest way of creating a class that implements the Runnable interface. Java gives an opportunity to apply multiple interfaces at a time. To implement Runnable a single method **run ( )** is used.

By implementing the Runnable interface, it creates a Thread object within a class that defines several constructor and pass Runnable implementation class object to its constructor.

Runnable Interface:

- Create an object of the high class.
- Assign a thread object for the class thread.
- Call the method "start" on the assign thread object.

Extending Thread

Extending thread is another way of creating a thread. It creates an instance of a new class that extends Thread. To prolong a thread it has to override run ( ) method. To start performing thread object, start () method should be called.

Creating Multiple Thread

Multiple threads are created for the optimal use of memory resources and speed up the execution time by dividing a task into multiple threads.

isAlive() and Join() Methods

All threads are interdependent with each other. One thread should know when another thread is ending. In Java, there are two methods to check whether one thread has finished its execution or not. The methods are:

- isAlive ( ) method
- Join ( ) method

isAlive() method tells the current status of a thread by returning a boolean value as True if the specified thread which is called is still running. Otherwise, it returns False.

Syntax:

final Boolean isAlive()

Join() Method

Join ( ) method waits till thread on which it is called is terminated. It is the most commonly used method when compared to isAlive( ). It informs the other thread to wait until the specified thread completes its execution.

Thread Priorities

- All threads have priorities. They are indicated by a number between 1 and 10. The thread scheduler schedules the thread according to their priority. The process of scheduling the threads to its priority is known as **Preemptive scheduling.** They are dependent on Java Virtual Machine, which chooses the schedule. The thread priorities are:

  - ❑ MIN_PRIORITY
  - ❑ MAX_PRIORITY
  - ❑ NORM_PRIORITY

- As discussed in Thread methods, set Priority( ) and get Priority( ) methods are called by passing an integer with their priorities which range between two constants MIN_PRIORITY and MAX_PRIORITY defined on threads.
- Default Priority of a Thread is 5(NORM_PRIORITY).
- The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

Problem with Priorities

1. Priorities depend on the system's OS and virtual memory. The thread method Thread.setPriority( ) does not prioritisation while handling multiple events.

2. No priority is beyond the possible range. For example, thread handling an audio data pre-empted by some random user thread.

3. The availability of priorities differs from system to system.

4. Manually manipulating a thread priority will interfere with the OS which in turn reduces the speed of the system.

5. It is hard to predict the current status of a thread because the current running application doesn't know what threads are running in other processes.

Example- Thread Priority

```
classTestMorePriority extends Thread{
public void run( )
{
System.out.println("first running thread name is:"+Thread.currentThread().getName());
System.out.println("another running thread priority is:"+Thread.currentThread().getPriority()); }
public static void main(String[] args){
TestMorePriority p1=new TestMorePriority();
TestMorePriority p2=new TestMorePriority();
p1.setPriority(Thread.MIN_PRIORITY);
p2.setPriority(Thread.MAX_PRIORITY);
p1.start();
p2.start();
```

Example-Output

The output of this above program, shown as follows when running under Windows, indicates that the threads did a context switch, even though neither voluntarily yielded the CPU nor blocked for I/O. The higher-priority thread got the majority of the CPU time.

Output:

```
first running thread name is:

another running thread priority is:

first running thread name is:

another running thread priority is:
```

Of course, the accurate output produced by this program depends on the speed of your CPU and the number of other tasks running in the system. When this same program is run under a non-preemptive system, many results will be obtained.

Thread Synchronization

- When more than one thread tries to access a shared resource, the programmer needs to ensure that only one thread will use the resource at a time. The process of allocating the resource for a single thread at a time is known as **Thread synchronisation.**
- Synchronised keyword is used to create a block of code called as a critical section. Every object with a critical section of code gets a lock associated with the object.

Syntax:

```
Synchronised (object)
{
 //Statement which is to be
synchronized
   }
```

Inter thread Communication

Interthread communication is a process of making all synchronised threads to communicate with each other. It is a mechanism by which, a thread delays in its critical section and another thread is made to enter the same critical section to execute a task. It can be done through Object methods ( ) such as:

- **wait( )** method notify the called thread to give up the monitor and go to sleep until some other thread calls notify( ).
- **notify( )** method wakes up the thread which already called wait( ) on the same object.
- **notifyAll( )** method wakes up all the thread that had called wait( ) method.

Suspending, resuming and stopping threads

Suspending a Thread

Suspending a Thread is a process of postponing a thread in suspended state and making it resume later. It can be achieved using suspend( ) method.

Syntax- public void suspend()

Resuming a Thread

Resuming a Thread is a process of making a thread resume to its original running state after suspending.

Syntax-public void resume( )

Stopping a Thread

Stopping a Thread is a process of stopping a thread completely. It can be done using stop() method.

Syntax-public void stop()

Summary

- ✓ Java is a multithreaded programming language. It contains two or more paths that can simultaneously run and handle the different task at the same time.
- ✓ A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution.
- ✓ According to Sun Microsystems, there are only four states. That is, new, runnable, non-runnable and terminated.
- ✓ Java's multithreaded model is based on Thread class because it acts as middle class in the multithreaded system.
- ✓ The thread scheduler schedules the thread according to their priority. The process of scheduling the threads to its priority is known as Preemptive scheduling.
- ✓ Interthread communication is a process of making all synchronised threads to communicate with each other.