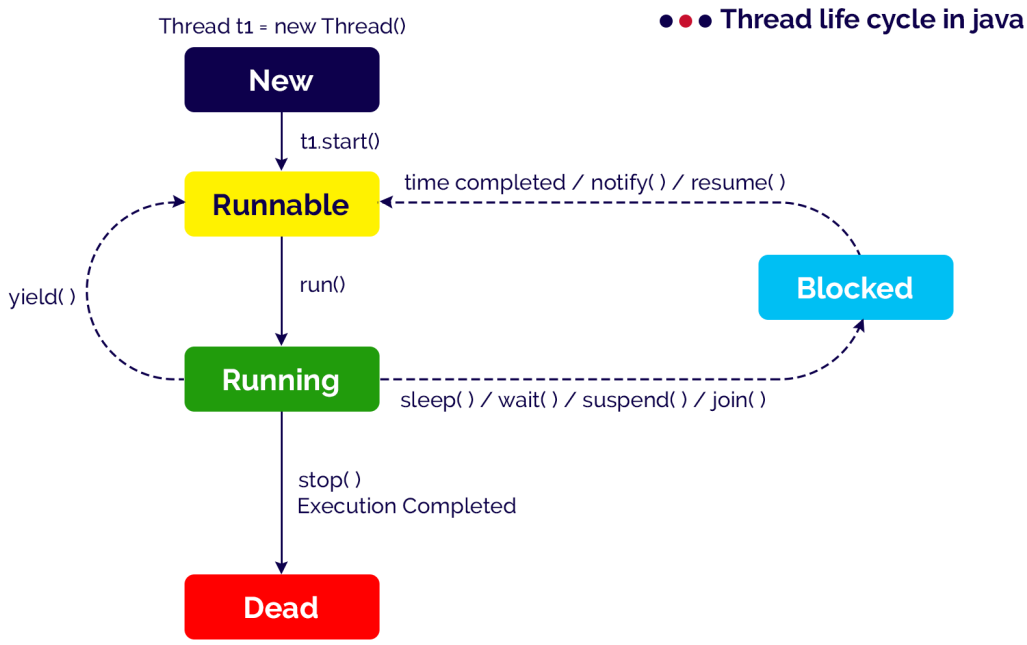
Unit-4 Multithreading Programming

A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources especially when your computer has multiple CPUs. Each of the threads can run in parallel.

Multi-threading enables you to write in a way where multiple activities can proceed concurrently in the same program.

Q-1 Explain thread life cycle with diagram.



Thread is a:

- Feature through which we can perform multiple activities within a single process.
- Lightweight process.
- Series of executed statements.
- Nested sequence of method calls.

New

When a thread object is created using new, then the thread is said to be in the New state. This state is also known as Born state.

```
Thread t1 = new Thread();
```

Runnable / Ready

After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task `t1.start( );`

Running

When a thread calls run( ) method, then the thread is said to be Running. The run( ) method of a thread called automatically by the start( ) method.

Blocked / Waiting

A thread in the Running state may move into the blocked state due to various reasons like sleep( ) method called, wait( ) method called, suspend( ) method called, and join( ) method called, etc.

When a thread is in the blocked or waiting state, it may move to Runnable state due to reasons like sleep time completed, waiting time completed, notify( ) or notifyAll( ) method called, resume( ) method called, etc.

Dead / Terminated

A thread in the Running state may move into the dead state due to either its execution completed or the stop( ) method called. The dead state is also known as the terminated state.

Q-2 Explain the ways to create thread in java program.

Method -1 Create thread in java By extending Thread class

Step-1: Create a class that extends Thread class.

Step-2: Override the run( ) method with the code that is to be executed by the thread. The run( )

method must be public while overriding.

Step-3: Create the object of the newly created class in the main( ) method.

Step-4: Call the start( ) method on the object created in the above step.

Program 1:

```
public class DemoThread {

	public static void main(String[] args) {
		MyThread t1=new MyThread();
		t1.start();
	}
}
class MyThread extends Thread
```

```
{
	public void run()
	{
		for(int i=0;i<5;i++)
			System.out.println(" i = " + i);
	}
}
```

Program 2:

```
public class DemoThread {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		MyThread t1 = new MyThread("Thread1 ",1);
		t1.start();
		MyThread t2 = new MyThread("Thread2 ",6);
		t2.start();
	}
}

class MyThread extends Thread
{
	String tname;
	int no;
	MyThread(String tname,int no)
	{
		this.tname=tname;
		this.no=no;
	}
	public void run()
	{
		int n=5;
		for(int i=0;i<n;i++)
			System.out.println(tname + "i = " +(i+no));
	}
}
```

Method -2 Create thread in java by implementing runnable interface

Step-1: Create a class that implements Runnable interface.

Step-2: Override the run( ) method with the code that is to be executed by the thread. The run( ) method must be public while overriding.

Step-3: Create the object of the newly created class in the main( ) method.

Step-4: Create the Thread class object by passing above created object as parameter to the Thread class constructor.

Step-5: Call the start( ) method on the Thread class object created in the above step.

Program 1:

```
public class DemoThreasdRunnable {

	public static void main(String[] args) {
		MyThread1 t1=new MyThread1();
		Thread th1=new Thread(t1);
		th1.start();
	}
}
class MyThread1 implements Runnable
{
	public void run()
	{
		for(int i=0;i<5;i++)
			System.out.println(" i = " + i);
	}
}
```

Program 2:

```
public class DemoThread {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		MyThread t1 = new MyThread("Thread1 ",1);
		Thread th1=new Thread(t1);
		th1.start();
	}
}

class MyThread implements Runnable
{
	String tname;
	int no;
	MyThread(String tname,int no)
	{
		this.tname=tname;
		this.no=no;
	}
	public void run()
	{
		int n=5;
		for(int i=0;i<n;i++)
			System.out.println(tname + "i = " +(i+no));
	}
}
```

Q-2 Explain synchronized threads.

There is the asynchronous behavior of the programs. If one thread is writing some data and another thread which is reading data at the same time, might create inconsistency in the application.

When there is a need to access the shared resources by two or more threads, then synchronization approach is utilized.

Java has provided synchronized methods to implement synchronized behavior.

In this approach, once the thread reaches inside the synchronized block, then no other thread can call that method on the same object. All threads have to wait till that thread finishes the synchronized block and comes out of that.

In this way, the synchronization helps in a multithreaded application. One thread has to wait till other thread finishes its execution only then the other threads are allowed for execution.

```
public class DemoTable {

	public static void main(String[] args) {
		table obj=new table();
		Mythread1 t1=new Mythread1(obj,5);
		t1.start();
		Mythread1 t2=new Mythread1(obj,4);
		t2.start();
	}
}
class Mythread1 extends Thread
{
	table t;
	int n;
	public Mythread1(table t,int n) {
		this.t=t;
		this.n=n;
	}
	public void run() {
		t.printTable(n);
	}
}

class table
{
	public void printTable(int n)
	{
		for(int i=1; i<=5;i++)
			System.out.println(n +" * " + i + " = " +n*i);
	}
}
```

Another Program :

```
public class DemoBankCustomer
{
	public static void main(String[] args)
	{

		Bank b=new Bank(5000);
```

```
		Thread t1=new Thread(b,"Ram");
		Thread t2=new Thread(b,"Shyam");
		Thread t3=new Thread(b,"Bharat");
		t1.start();t2.start();t3.start();
	}
}

class Bank implements Runnable
{
	int bal;

	Bank(int bal)
	{
		this.bal = bal;
	}
	synchronized public void run()
	{
		System.out.println(Thread.currentThread().getName() +" Available Bal = "+bal);
		bal-=1000;
	}
}
```

Q-3 Explain interthread communication.

Inter-thread communication or **Co-operation** is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of **Object class**:

- wait() -Causes the current thread to wait until another thread invokes the notify() or notifyAll().
- notify() -Wakes up a single thread that is waiting on this object's monitor.
- notifyAll() -Wakes up all the threads that called wait( ) on the same object.

```
public class DemoBank {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		HDFCBank b = new HDFCBank();
		T1 t1 = new T1(b);
		T2 t2 = new T2(b);
	}
}

class T1 extends Thread
{
	HDFCBank b;

	T1(HDFCBank b) {
		this.b = b;
```

```
		this.start();
	}

	public void run() {
		b.withdraw(8000);
	}
}

class T2 extends Thread {
	HDFCBank b;

	T2(HDFCBank b) {
		this.b = b;
		this.start();
	}

	public void run() {
		b.deposit(10000);
	}
}

class HDFCBank {
	int bal = 5000;

	synchronized void withdraw(int amt) {
		if (amt > bal)
		{
			System.out.println("AvlBal = " + bal + " Less balance... Wait a while");
			try {
				wait();
			}
			catch (Exception e) {}
		}
		bal = bal - amt;
		System.out.println("Money withdraw = " +  amt +" AvlBal = " + bal);
	}

	synchronized void deposit(int amt)
	{
		bal = bal + amt;
		System.out.println("Money Deposited..." + amt +" AvlBal = " + bal);
		notifyAll();
	}
}
```

Q-4 Explain thread priority

Object-oriented works within a multithreading environment in which thread scheduler assigns the processor to a thread based on the priority of thread. Whenever we create a thread in Java, it always has some priority assigned to it. Priority can either be given by JVM while creating the thread or it can be given by the programmer explicitly.

Priorities are represented by a number between 1 and 10.
The default priority is set to 5 as excepted.
Minimum priority is set to 1.

Maximum priority is set to 10.

How to get and set priority of a thread in java.

public final int getPriority(): java.lang.Thread.getPriority() method returns priority of given thread.
public final void setPriority(int newPriority): java.lang.Thread.setPriority() method changes the priority of thread to the value newPriority. This method throws IllegalArgumentException if value of parameter newPriority goes beyond minimum(1) and maximum(10) limit.

```
class ThreadDemo extends Thread
{
       public static void main(String[] args)
    {
        ThreadDemo t1 = new ThreadDemo();

        System.out.println("t1 Default thread priority : "+ t1.getPriority());

        t1.setPriority(2);

        System.out.println("t1 thread priority : " + t1.getPriority());

        t1.start();
    }
    public void run()
    {
        System.out.println("Inside run method");
    }
}
```