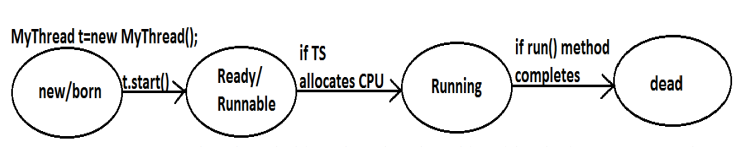
**Life Cycle of Thread**

There are different types of thread state are as follows as-

1. New or Born State-

The thread is in new state if you create an instance of Thread class but before the invocation of start () method.

1. Runnable state-

The thread is in runnable state after invocation of start () method, but the thread scheduler has not selected it to be the running thread.

1. Running state-

The thread is in running state if the thread scheduler has selected it.

1. Dead state-

A thread is in terminated or dead state when its run () method exits.

1. Sleep state-

If running thread calls sleep method then it will enter into sleep state. If sleeping thread got interrupted or time expire then it will enter into ready state.

1. Waiting state-

If running thread calls wait method then it will enter into waiting state. If waiting state got notification then it will enter into another wating state.

1. Suspended state-

If running state called suspend method then thread will enter into suspended state.

1. Resume state-

If we call thread from resume () method then it will enter into ready state.

**Synchronization in Java-**

We can apply synchronization on method and block only. We cannot apply it on variables and class.

Synchronization means multiple threads is accessing the one resource at the same time called as. The main purpose of this is we need to ensure that resource will be used by only one thread at a time. The process by which this is achieved is called synchronization.

Why?

**package** com.synchronizaitons;

**public** **class** Account {

**private** **int** balance=5000;

**public** **int** getBalance() {

**return** balance;

}

**public** **int** withdraw(**int** amount) {

balance= balance-amount;

**return** balance;

}

}

**package** com.synchronizaitons;

**public** **class** AccountDetails **implements** Runnable {

Account account = **new** Account();

@Override

**public** **void** run() {

**for** (**int** x = 0; x < 5; x++) {

makeWithdrawal(500);

**if** (account.getBalance() <= 0) {

System.***out***.println("Account is overdrawn...");

}

}

}

**private** **void** makeWithdrawal(**int** amt) {

**if** (account.getBalance() >= amt) {

System.***out***.println(Thread.*currentThread*().getName() +

"is going to withdraw => " + amt);

}

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

System.***out***.println(e.getMessage());

}

**int** bal = account.withdraw(amt);

System.***out***.println(Thread.*currentThread*().getName() +

"completed the withdrawal, Avl bal => " + bal);

}

}

**package** com.synchronizaitons;

**public** **class** MainTest {

**public** **static** **void** main(String[] args) {

AccountDetails accountDetails= **new** AccountDetails();

Thread thread1=**new** Thread(accountDetails);

Thread thread2= **new** Thread(accountDetails);

thread1.setName("Rohan");

thread2.setName("soham");

thread1.start();

thread2.start();

}

}

In this example, there are two thread which are executed randomly but I want to execute one by one thread at a time then go for synchronization.

Note- Just make the makeWithdrawal method as synchronized, so you will get the output like as

**Output- using synchronization**

soham is going to withdraw amt => 500

soham completed the withdrawal , Avl bal = > 4500

soham is going to withdraw amt => 500

soham completed the withdrawal , Avl bal = > 4000

soham is going to withdraw amt => 500

soham completed the withdrawal, Avl bal = > 3500

soham is going to withdraw amt => 500

soham completed the withdrawal , Avl bal = > 3000

soham is going to withdraw amt => 500

soham completed the withdrawal , Avl bal = > 2500

Rohan is going to withdraw amt => 500

Rohan completed the withdrawal , Avl bal = > 2000

Rohan is going to withdraw amt => 500

Rohan completed the withdrawal , Avl bal = > 1500

Rohan is going to withdraw amt => 500

Rohan completed the withdrawal , Avl bal = > 1000

Rohan is going to withdraw amt => 500

Rohan completed the withdrawal , Avl bal = > 500

Rohan is going to withdraw amt => 500

Rohan completed the withdrawal , Avl bal = > 0

Account is overdrawn...

**Output- without synchronization**

Rohan >>is going to withdraw=>

soham>>is going to withdraw=>

soham>>complete the withdrawal=>4500

Rohan >>complete the withdrawal=>4000

soham>>is going to withdraw=>

Rohan >>is going to withdraw=>

Rohan >>complete the withdrawal=>3500

Rohan >>is going to withdraw=>

soham>>complete the withdrawal=>3000

soham>>is going to withdraw=>

Rohan >>complete the withdrawal=>2000

soham>>complete the withdrawal=>2500

soham>>is going to withdraw=>

Rohan >>is going to withdraw=>

Rohan >>complete the withdrawal=>1500

soham>>complete the withdrawal=>1000

Rohan >>is going to withdraw=>

soham>>is going to withdraw=>

soham>>complete the withdrawal=>0

Account is overdrawn...

Rohan >>complete the withdrawal=>500

Account is overdrawn...

**Synchronized method-**

If you declare any method as synchronized, it is known as synchronized method.

Synchronized method is used to lock an object for any shared resource.

When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

**Example**- synchronized void test (){

// write code here

}

**Synchronized Block-**

Synchronized block can be used to perform synchronization on any specific resource of the method.

Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.

If you put all the codes of the method in the synchronized block, it will work same as the synchronized method.

Note-

* Synchronized block is used to lock an object for any shared resource.
* Scope of synchronized block is smaller than the method.

**Syntax-**

synchronized (object reference) {

  //code block

}