//Program on Thread.sleep() method :

```java
package com.ravi.sleep;

class MyTest extends Thread
{
    @Override
    public void run()  //m1  m2
    {
        for(int i=1; i<=5; i++)
        {
            System.out.println("i value is :"+i);  //11  22  33   44    55
            try
            {
                Thread.sleep(1000);
            }
            catch(InterruptedException e)
            {
                e.printStackTrace();
            }
        }

    }
}
public class SleepDemo2
{
    public static void main(String[] args)
    {
        MyTest m1 = new MyTest();
        MyTest m2 = new MyTest();

        m1.start();
        m2.start();
    }
}
```

public final long threadId() :
----
* It is a new method which introduced from JDK 19V, Actually before threadId() method we had getId() method to get the id of the thread but this method is not final so this method is deprecated from JDK 19V and instead of getId() now we have have threadId() method which is a final method.

* Both the methods are used to provide the id of the current thread.

```java
package com.ravi.sleep;

class MyThread1 extends Thread
{
    @Override
    public void run()
    {
        System.out.println("Child Thread is running");

        System.out.println("Child thread id is :"+Thread.currentThread().threadId());
    }
}

public class SleepDemo3
{
    public static void main(String[] args) throws InterruptedException
    {
        System.out.println("Main thread is Started");
        System.out.println("Main thread id is :"+Thread.currentThread().threadId());

        MyThread1 mt = new MyThread1();
        mt.start();

        mt.sleep(2000);
        //Thread.currentThread().sleep(3000);

        System.out.println("Main thread is Ended");

    }

}
```

Note : We are calling sleep() method on the current thread i.e main thread so main thread will go into sleeping state.

Old Thread life cycle :
----
* As we know a thread is well known for independent execution, During its life cycle It passes through different phases/state which are as follows :

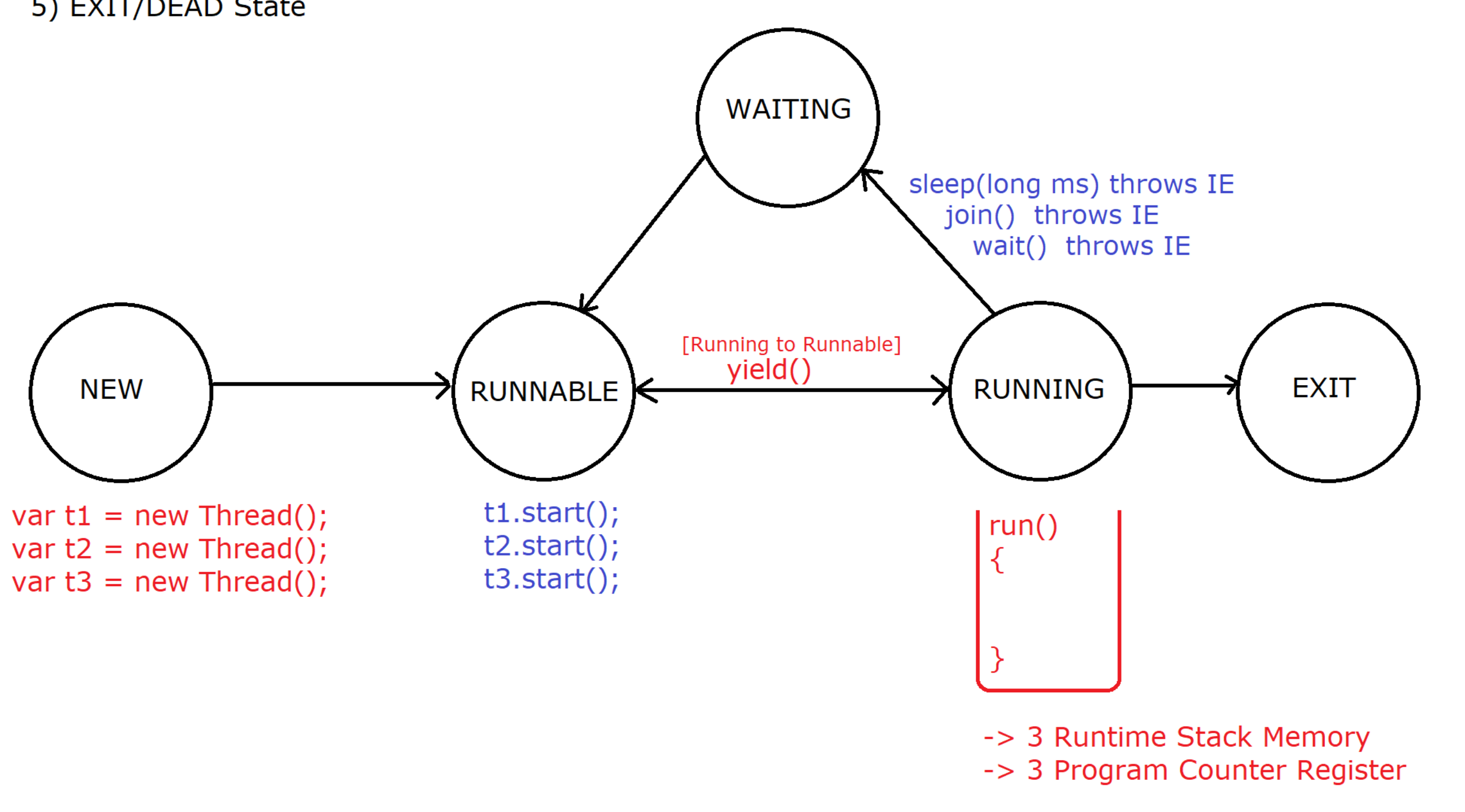
1) NEW State (Born State)

2) RUNNABLE State (Ready to run)

3) RUNNING State

4) WAITING State

5) EXIT/DEAD State



New State :-

Whenever we create a thread instance(Thread Object) a thread comes to new state OR born state. New state does not mean that the Thread has started yet only the object or instance of Thread has been created.

Runnable state :-

Whenever we call start() method on thread object, A thread moves to Runnable state i.e Ready to run state. Here Thread schedular is responsible to select/pick a particular Thread from Runnable state and sending that particular thread to Running state for execution.

Running state :-

If a thread is in Running state that means the thread is executing its own run() method in a separate runtime stack Memory.

From Running state a thread can move to waiting state either by an order of thread schedular OR user has written some method(wait(), join() or sleep()) to put the thread into temporarly waiting state.

From Running state the Thread may also move to Runnable state directly, if user has written Thread.yield() method explicitly.

Waiting state :-

A thread is in waiting state means it is waiting for it's time period to complete OR in some cases it is also waiting for lock (monitor) OR another thread to complete. Once the time period will be completed then it will re-enter inside the Runnable state to complete its remaining task.

Dead or Exit :

Once a thread has successfully completed its run method in the corresponding stack then the thread will move to EXIT state. Please remember once a thread is dead we can't restart a thread in java.

IQ :- If we write Thread.sleep(1000) then exactly after 1 sec the Thread will re-start?

Ans :- No, We can't say that the Thread will directly move from waiting state to Running state.

The Thread will definetly wait for 1 sec in the waiting state and then again it will re-enter into Runnable state which is control by Thread Schedular so we can't say that the Thread will re-start just after 1 sec.

Anonymous inner class by using Thread class with reference variable :

```java
package com.ravi.anonymous;

public class AnonymousThreadWithReference
{
    public static void main(String[] args)
    {
        //Anonymous inner class (with reference)
        Thread thread = new Thread()
        {
            @Override
            public void run()
            {
                String name = Thread.currentThread().getName();
                long id = Thread.currentThread().threadId();

                System.out.println("Id of the Thread is :"+id);
                System.out.println("Name of the Thread is :"+name);
            }
        };
        thread.start();

    }

}
```

Anonymous inner class by using Thread class without reference variable :

```java
package com.ravi.anonymous;

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

        //Anonymous inner class (without reference)
        new Thread()
        {
            @Override
            public void run()
            {
                String name = Thread.currentThread().getName();
                long id = Thread.currentThread().threadId();

                System.out.println("Id of the Thread is :"+id);
                System.out.println("Name of the Thread is :"+name);
            }
        }.start();


        //Anonymous inner class (without reference)
        new Thread()
        {
            @Override
            public void run()
            {
                String name = Thread.currentThread().getName();
                long id = Thread.currentThread().threadId();

                System.out.println("Id of the Thread is :"+id);
                System.out.println("Name of the Thread is :"+name);
            }
        }.start();

    }
}
```

---

public void join() throws InterruptedException :