[12:11 PM] Madabattula Rani

**Multithreading**

**Thread**

A Thread is a lightweight process that exists within a process and shares its resources. Multiple threads can execute within a single program. This allows for concurrency.

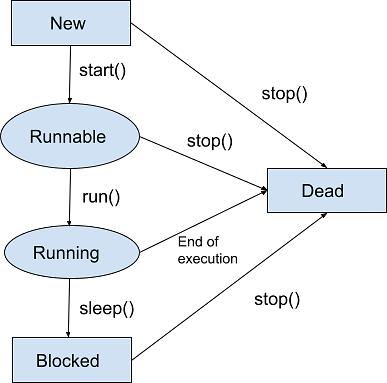
Lifecycle of a Thread in Java

The Life Cycle of a Thread in Java refers to the state transformations of a thread that begins with its birth and ends with its death. When a thread instance is generated and executed by calling the start() method of the Thread class, the thread enters the runnable state. When the sleep() or wait() methods of the Thread class are called, the thread enters a non-runnable mode.

Thread returns from non-runnable state to runnable state and starts statement execution. The thread dies when it exits the run() process. In [Java](https://www.simplilearn.com/tutorials/java-tutorial/what-is-java), these thread state transformations are referred to as the Thread life cycle.

There are basically 4 stages in the lifecycle of a thread, as given below:

1. New
2. Runnable
3. Running
4. Blocked (Non-runnable state)
5. Dead



* New State

As we use the Thread class to construct a thread entity, the thread is born and is defined as being in the New state. That is, when a thread is created, it enters a new state, but the start() method on the instance has not yet been invoked.

* Runnable State

A thread in the runnable state is prepared to execute the[code.](https://www.simplilearn.com/free-and-low-cost-online-resources-for-practicing-code-article) When a new thread's start() function is called, it enters a runnable state.

In the runnable environment, the thread is ready for execution and is awaiting the processor's availability (CPU time). That is, the thread has entered the queue (line) of threads waiting for execution.

* Running State

Running implies that the processor (CPU) has assigned a time slot to the thread for execution. When a thread from the runnable state is chosen for execution by the thread scheduler, it joins the running state.

In the running state, the processor allots time to the thread for execution and runs its run procedure. This is the state in which the thread directly executes its operations. Only from the runnable state will a thread enter the running state.

* Blocked State

When the thread is alive, i.e., the thread class object persists, but it cannot be selected for execution by the scheduler. It is now inactive.

* Dead State

When a thread's run() function ends the execution of sentences, it automatically dies or enters the dead state. That is, when a thread exits the run() process, it is terminated or killed. When the stop() function is invoked, a thread will also go dead.

**Creating Thread**

**-------------- Extending Thread Class**

package creating;  
  
public class Demo1 extends Thread  
{  
  
    public void run()  
    {  
        for(int i=1;i<=5;i++)  
        {  
            System.*out*.println(Thread.*currentThread*()+"--------"+i+ " ");  
            try {  
                Thread.*sleep*(2000);  
            } catch (InterruptedException e)  
            {  
  
               System.*out*.println("Thread Interrupted");  
            }  
  
        }  
    }  
    public static void main(String args[])  
    {  
        Demo1 t1 = new Demo1();  
        Demo1 t2 = new Demo1();  
        t1.start();  
        t2.start();  
  
    }}

**-------------- Implementing Runnable Interface**

package creating;  
  
public class Demo2  implements Runnable {  
  
  
    public void run() {  
        for (int i = 1; i <= 5; i++) {  
            System.out.println(Thread.currentThread().getName() + "--------" + i + " ");  
            try {  
                Thread.sleep(2000);  
            } catch (InterruptedException e) {  
  
                System.out.println("Thread Interrupted");  
            }  
  
        }  
    }  
  
    public static void main(String args[]) throws InterruptedException {  
  
  
        Demo2 t1 = new Demo2();  
        Thread t11 = new Thread(t1,"Thread 1");  
        Demo2 t2 = new Demo2();  
        Thread t12 = new Thread(t1,"Thread 2");  
        t11.start();  
        t11.join();  
        t12.start();  
  
  
  
  
  
    }  
}

**Constructors of this class are as follows:**

Thread() Allocates a new Thread object.

Thread(Runnable target)    Allocates a new Thread object.

Thread(Runnable target, String name)    Allocates a new Thread object.

Thread(String name)   Allocates a new Thread object.

Thread(ThreadGroup group, Runnable target)         Allocates a new Thread object.

Thread(ThreadGroup group, Runnable target, String name)     Allocates a new Thread object so that it has targeted as its run object, has the specified name as its name, and belongs to the thread group referred to by a group.

Thread(ThreadGroup group, Runnable target, String name, long stackSize) Allocates a new Thread object so that it has targeted as its run object, has the specified name as its name, and belongs to the thread group referred to by group, and has the specified stack size.

Thread(ThreadGroup group, String name)         Allocates a new Thread object.

**Methods of Thread class:**

activeCount()         Returns an estimate of the number of active threads in the current thread’s thread group and its subgroups

checkAccess()        Determines if the currently running thread has permission to modify this thread

clone()    Throws CloneNotSupportedException as a Thread can not be meaningfully cloned

currentThread()    Returns a reference to the currently executing thread object

dumpStack()  Prints a stack trace of the current thread to the standard error stream

enumerate(Thread[] tarray)      Copies into the specified array every active thread in the current thread’s thread group and its subgroups

getAllStackTraces()      Returns a map of stack traces for all live threads

getContextClassLoader()     Returns the context ClassLoader for this Thread

getDefaultUncaughtExceptionHandler() Returns the default handler invoked when a thread abruptly terminates due to an uncaught exception

getId()     Returns the identifier of this Thread

getName()     Returns this thread’s name

getPriority()  Returns this thread’s priority

getStackTrace()     Returns an array of stack trace elements representing the stack dump of this thread

getState()       Returns the state of this thread

getThreadGroup()        Returns the thread group to which this thread belongs

getUncaughtExceptionHandler()       Returns the handler invoked when this thread abruptly terminates due to an uncaught exception

holdsLock(Object obj) Returns true if and only if the current thread holds the monitor lock on the specified object

interrupt()      Interrupts this thread

interrupted() Tests whether the current thread has been interrupted

isAlive()  Tests if this thread is alive

isDaemon()    Tests if this thread is a daemon thread

isInterrupted()      Tests whether this thread has been interrupted

join()       Waits for this thread to die

join(long millis)     Waits at most millis milliseconds for this thread to die

run()        If this thread was constructed using a separate Runnable run object, then that Runnable object’s run method is called; otherwise, this method does nothing and returns

setContextClassLoader(ClassLoader cl)   Sets the context ClassLoader for this Thread

setDaemon(boolean on)     Marks this thread as either a daemon thread or a user thread

setDefaultUncaughtExceptionHandler( Thread.UncaughtExceptionHandler eh)  Set the default handler invoked when a thread abruptly terminates due to an uncaught exception, and no other handler has been defined for that thread

setName(String name)        Changes the name of this thread to be equal to the argument name.

setUncaughtExceptionHandler( Thread.UncaughtExceptionHandler eh)  Set the handler invoked when this thread abruptly terminates due to an uncaught exception

setPriority(int newPriority)        Changes the priority of this thread

sleep(long millis)  Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds, subject to the precision and accuracy of system timers and schedulers

start()     Causes this thread to begin execution; the Java Virtual Machine calls the run method of this thread

toString()        Returns a string representation of this thread, including the thread’s name, priority, and thread group

yield()     A hint to the scheduler that the current thread is willing to yield its current use of a processor

**Thread Synchronization: Synchronized Blocks and Methods**

**Synchronization**

In Java is the process that allows only one thread at a time to complete the task.

Synchronized method

Synchronized Block.

**Synchronized method**

package synchronization;  
  
class TableCreation  
{  
   synchronized public void  createTable(int n)  
    {  
        for(int i=1;i<=5;i++)  
        {  
            System.out.println(n+ " \* "+ i+ " = "+ n\*i);  
            try  
            {  
                Thread.sleep(2000);  
            }  
            catch(Exception e)  
            {  
                System.out.println("Thread Interrupted");  
            }  
        }  
    }  
}  
class Thread1 extends Thread  
{  
    TableCreation obj;  
    public Thread1(TableCreation o)  
    {  
        this.obj = o;  
    }  
    public void run()  
    {  
       obj.createTable(10);  
    }  
}  
class Thread2 extends Thread  
{  
    TableCreation obj;  
    public Thread2(TableCreation o)  
    {  
        this.obj = o;  
    }  
    public void run()  
    {  
        obj.createTable(20);  
    }  
}  
  
  
public class Main  
{  
    public static void main(String args[])  
    {  
        TableCreation object = new TableCreation();  
        Thread1 t1 = new Thread1(object);  
        Thread2 t2 = new Thread2(object);  
        t1.start();  
        t2.start();  
    }  
}

**Synchronized Block.**

When we want to synchronize few lines of the code than we can go with synchronized block.

In synchronized method lock is on the method but in synchronized block the lock is on the object.

package synchronization;  
class Table {  
    public void createTable(int n) {  
        System.out.println(Thread.currentThread().getName()); // This part is not synchronized.  
        synchronized (this) { //  This part is synchronized.  
            for (int i = 1; i <= 5; i++) {  
                System.out.println(n + " \* " + i + " = " + n \* i);  
                try {  
                    Thread.sleep(2000);  
                } catch (Exception e) {  
                    System.out.println("Thread Interrupted");  
                }  
            }  
        }  
  
    }  
}  
class Thread11 extends Thread  
{  
    Table obj;  
    public Thread11(Table o)  
    {  
        this.obj = o;  
    }  
    public void run()  
    {  
        obj.createTable(10);  
    }  
}  
class Thread22 extends Thread  
{  
    Table obj;  
    public Thread22(Table o)  
    {  
        this.obj = o;  
    }  
    public void run()  
    {  
        obj.createTable(20);  
    }  
}  
  
public class SynchronizedBlock {  
        public static void main(String [] args)  
        {  
           Table t = new Table();  
            Thread11 t1 = new Thread11(t);  
            t1.start();  
            Thread22 t2 = new Thread22(t);  
            t2.start();  
        }  
}

What is Java: A Beginners Guide To Java

The article helps you to understand what is Java, history pf Java, what is Java used for along with its features and concepts. So, click here to read more about Java