Multithreading in Java

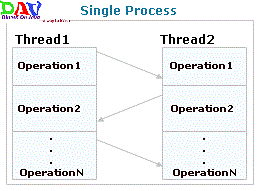
**Multithreading**is a process of executing multiple threads simultaneously. A program can be divided into a number of small processes. Each small process can be addressed as a single thread (a lightweight process). **Multithreaded**programs contain two or more threads that can run concurrently. This means that a single program can perform two or more tasks simultaneously. **For example**, one thread is writing content on a file at the same time another thread is performing spelling check.

**Thread:**

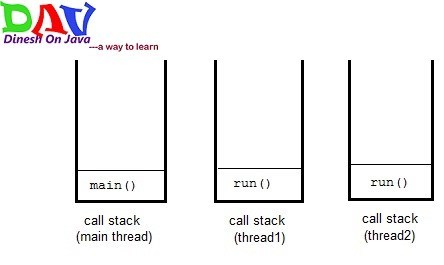
**Thread**is basically a lightweight **subprocess**, a smallest unit of processing. Multiprocessing and **multithreading**, both are used to achieve **multitasking**. But we use **multithreading**than **mulitprocessing** because threads share a common memory area. They don’t allocate separate memory area so save memory, and context-switching between the threads takes less time than processes.  
**Multithreading**is mostly used in games, animation etc.

**Multitasking**is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved by two ways:

* Process-based Multitasking(**Multiprocessing**)
* Thread-based Multitasking(**Multithreading**)



In this diagram, two threads are being executed having more than one task. The task of each thread is switched to the task of another thread.



In this diagram during multi processing each process contain separate memory call stack

**1)Process-based Multitasking (Multiprocessing)**

* Each process have its own address in memory i.e. each process allocates separate memory area.
* Process is heavyweight.
* Cost of communication between the process is high.
* Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.

**2)Thread-based Multitasking (Multithreading)**

* Threads share the same address space.
* Thread is lightweight.
* Cost of communication between the thread is low.
* Note:At least one process is required for each thread.

**Advantages of multithreading over multiprocessing :**

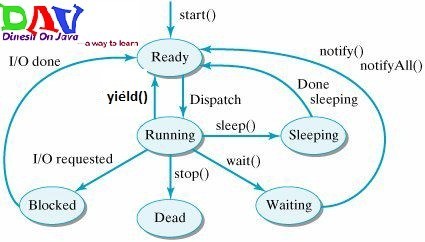
* Reduces the computation time.
* Improves performance of an application.
* Threads share the same address space so it saves the memory.
* Context switching between threads is usually less expensive than between processes.
* Cost of communication between threads is relatively low.

Life Cycle of A Thread

When you are programming with threads, understanding the life cycle of thread is very valuable. While a thread is alive, it is in one of several states.

A **thread**can be in one of the five states in the thread. According to sun, there is only 4 states **new**, **runnable**, **non-runnable** and **terminated**. There is no **running**state. But for better understanding the threads, we are explaining it in the 5 states. The life cycle of the thread is controlled by [**JVM**](https://www.dineshonjava.com/java-virtual-machine/). The thread states are as follows:

1. **New**
2. **Runnable(Ready)**
3. **Running**
4. **Non-Runnable (Blocked, Sleeping, Waiting)**
5. **Terminated(Dead)**



|  |
| --- |
| 1) NewThe thread is in new state if you create an instance of Thread class but before the invocation of start() method. |

### 2) Runnable

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

### 3) Running

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

### 4) Non-Runnable (Blocked)

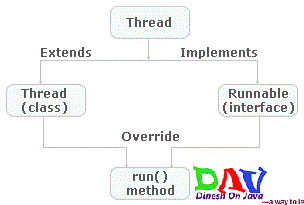
This is the state when the thread is still alive, but is currently not eligible to run.

### 5) Terminated

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

Creating a thread in Java :

1. Extending the ***java.lang.Thread*** Class
2. Implementing the ***java.lang.Runnable*** Interface



**Extending the *java.lang.Thread* Class**

For creating a thread a class have to extend the ***Thread***Class. For creating a thread by this procedure you have to follow these steps:

1. Extend the ***java.lang.Thread*** Class.
2. Override the ***run( )*** method in the subclass from the Thread class to define the code executed by the thread.
3. Create an instance of this subclass. This subclass may call a Thread class constructor by subclass constructor.
4. Invoke the ***start( )*** method on the instance of the class to make the thread eligible for running.

}

**Commonly used Constructors of Thread class:**

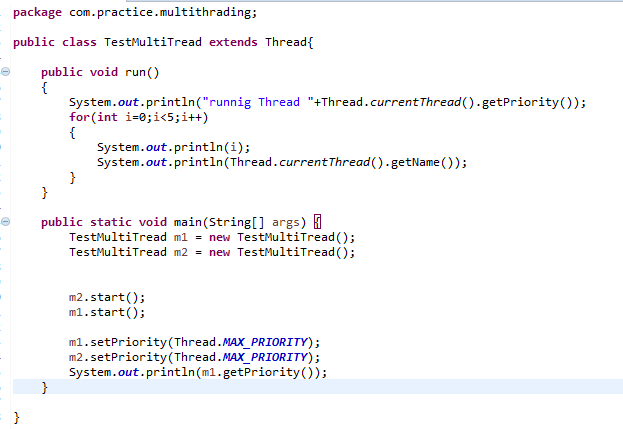
* Thread()
* Thread(String name)
* Thread(Runnable r)
* Thread(Runnable r,String name)

**Commonly used methods of Thread class:**

* **public void run():** is used to perform action for a thread.
* **public void start():** starts the execution of the thread.JVM calls the run() method on the thread.
* **public void sleep(long miliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
* **public void join():**waits for a thread to die.
* **public void join(long miliseconds):**waits for a thread to die for the specified miliseconds.
* **public int getPriority():**returns the priority of the thread.
* **public int setPriority(int priority):** changes the priority of the thread.
* **public String getName():**returns the name of the thread.
* **public void setName(String name):** changes the name of the thread.
* **public Thread currentThread():**returns the reference of currently executing thread.
* **public int getId():** returns the id of the thread.
* **public Thread.State getState():**returns the state of the thread.
* **public boolean isAlive():** tests if the thread is alive.
* **public void yield():**causes the currently executing thread object to temporarily pause and allow other threads to execute.
* **public void suspend():** is used to suspend the thread(depricated).
* **public void resume():** is used to resume the suspended thread(depricated).
* **public void stop():** is used to stop the thread(depricated).
* **public boolean isDaemon():** tests if the thread is a daemon thread.
* **public void setDaemon(boolean b):** marks the thread as daemon or user thread.
* **public void interrupt():**interrupts the thread.
* **public boolean isInterrupted():** tests if the thread has been interrupted.
* **public static boolean interrupted():**tests if the current thread has been interrupted.

Note:

* All Java threads have a priority and the thread with he highest priority is scheduled to run by the [**JVM**](https://www.dineshonjava.com/2013/02/java-virtual-machine.html).
* In case two threads have the same priority a FIFO ordering is followed.



Sleep Thread

class MultiThreadDemo extends Thread{

public void run(){

for(int i=1;i<5;i++){

try{

Thread.sleep(500);

}catch(InterruptedException e){

System.out.println(e);

}

System.out.println("Dinesh on Java Thread Application "+i);

}

}

public static void main(String args[]){

MultiThreadDemo t1 = new MultiThreadDemo();

MultiThreadDemo t2 = new MultiThreadDemo();

t1.start();

t2.start();

}

}

Join Method:

* Java Join method join the next thread at the end of the current thread
* After current thread stops execution then next thread executes.

# Java Garbage Collection

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management

It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.

How can object be unreferenced:

By nulling the object reference.

**Car c = new Car();**

**c= null**

By assigning a reference to another

**Car c1 = new Car();**

**Car c2 = new Car();**

**c1 =c2**

By anonymous object

new car();

Finalize Method :

The finalize method is invoked each time before object is garbage collected . this method is using for cleaning process. This method is define in object class.

**protected** **void** finalize(){}

#### Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

**Synchronization in java**

Synchronization in java is the capability to control the access of multiple thread in any shared resource .java synchronization is better option where we want to allow single resource.

Why use :

To prevent Thread interference.

To prevent Consistency Problem.

### Types of Synchronization

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

Here, we will discuss only thread synchronization.

### Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive
   1. Synchronized method.
   2. Synchronized block.
   3. static synchronization.
2. Cooperation (Inter-thread communication in java)