THREADS TUTORIAL :

Thread basics :

Ref : javatpoint

1. Life cycle of thread

* New -- new instance of thread
* Runnable - .start()
* Running - in run()
* Blocked - thread is alive but waitng for something
* Terminated - when run() method exits

1. How to create thread

* By implementing Runnable interface
* By extending Thread class

Choosing between these 2 depends on user

* Runnable interface helps for multiple inheritance

END of session :

* How to perform two tasks by two threads ?
* How to perform multithreading by annonymous class ?
* What is the Thread Schedular and what is the difference between preemptive scheduling and time slicing ?
* What happens if we start a thread twice ?
* What happens if we call the run() method instead of start() method ?
* What is the purpose of join method ?
* Why JVM terminates the daemon thread if there is no user threads remaining ?
* What is the shutdown hook?
* What is garbage collection ?
* What is the purpose of finalize() method ?
* What does gc() method ?
* What is synchronization and why use synchronization ?
* What is the difference between synchronized method and synchronized block ?
* What are the two ways to perform static synchronization ?
* What is deadlock and when it can occur ?
* What is interthread-communication or cooperation ?

Demon Threads :

T1.setDeamon(Boolean ss )

isDaemon()

these methds should be applied before starting thread i.e before t1.start();

Here is a short list of when you may want to use a daemon thread:

* **Collecting statistics and performing the status monitoring tasks** - Sending and receiving network heartbeats, supplying the services to monitoring tools, and so on.
* **Performing asynchronous I/O tasks** - You can create a queue of I/O requests, and set up a group of daemon threads servicing these requests asynchronously.
* **Listening for incoming connections** - daemon threads are very convenient in situations like this, because they let you program a simple "forever" loop, rather than creating a setup that pays attention to exit requests from the main thread.

What happens when main ends and if the user thread or daemon is running ?

When main thread(user thread ) ends …all daemon threads will die .

**public** **class** DaemonTest {

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

Thread t = **new** Thread(**new** Runnable() {

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

**while** (**true**) {

**try** {

Thread.*sleep*(10);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("I'm still alive");

}

}

});

// Try uncommenting/commenting this line

t.setDaemon(**true**);

t.start();

Thread.*sleep*(20);

System.***out***.println("Main thread has finished");

}

}

JAVA THREAD POOL example ? rela time use case

Involving executorservice and blocking queue etc ????/

What you mean by interrupting a thread or threadgroup ?

How to stop a thread ? why stop is deprecated ?

Daemon thread vs addshutdownhook() methods

Daemon thread will die as soon as program ends

Addshutdownhook comes when the programs starts shtting down …

So daemon will die , other memory cleaning will be done then shutdownhooks will be called before JVM stops internally .so here we will override using addshutdownhook() method .

addshutdownhook() will be called on

--- .exit() method

----- ctl+c or some other commands

Why to use shutdownhook ? use case ?

Use case of finalize () methos ? used for clearing the objects which are not created with new keywrd bcz Gc clears only objects created by new keyword

Whats the table space?