Multithreading in Java

**Multithreading in**[**Java**](https://www.javatpoint.com/java-tutorial) is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation, etc.

 Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.

2) You **can perform many operations together, so it saves time**.

3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

**Multitasking**

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

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

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

* Each process has an address in memory. In other words, each process allocates a separate memory area.
* A process is heavyweight.
* Cost of communication between the process is high.
* Switching from one process to another requires some time for saving and loading [registers](https://www.javatpoint.com/register-memory), memory maps, updating lists, etc.

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

* Threads share the same address space.
* A thread is lightweight.
* Cost of communication between the thread is low.

Life cycle of a Thread (Thread States)

1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)
5. Terminated

**1) New**

The 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.

**How to create thread**

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

**Thread class:**

|  |
| --- |
| Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface. |

Commonly used methods of Thread class:

|  |
| --- |
| **1. public void run():**is used to perform action for a thread.  **2. public void start():**starts the execution of the thread.JVM calls the run() method on the thread.  **3. public void sleep(long miliseconds):**Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.  **4. public void join():**waits for a thread to die.  **5. public void join(long miliseconds):**waits for a thread to die for the specified miliseconds.  **6. public int getPriority():**returns the priority of the thread.  **7. public int setPriority(int priority):**changes the priority of the thread.  **8. public String getName():**returns the name of the thread.  **9. public void setName(String name):**changes the name of the thread.  **10. public Thread currentThread():**returns the reference of currently executing thread.  **11. public int getId():**returns the id of the thread.  **12. public Thread.State getState():**returns the state of the thread.  **13. public boolean isAlive():**tests if the thread is alive.  **14. public void yield():**causes the currently executing thread object to temporarily pause and allow other threads to execute.  **15. public void suspend():**is used to suspend the thread(depricated).  **16. public void resume():**is used to resume the suspended thread(depricated).  **17. public void stop():**is used to stop the thread(depricated).  **18. public boolean isDaemon():**tests if the thread is a daemon thread.  **19. public void setDaemon(boolean b):**marks the thread as daemon or user thread.  **20. public void interrupt():**interrupts the thread.  **21. public boolean isInterrupted():**tests if the thread has been interrupted.  **22. public static boolean interrupted():**tests if the current thread has been interrupted. |

**Thread Scheduler in Java**

**Thread scheduler** in java is the part of the JVM that decides which thread should run.

There is no guarantee that which runnable thread will be chosen to run by the thread scheduler.

Only one thread at a time can run in a single process.

**The thread scheduler mainly uses preemptive or time slicing scheduling to schedule the threads.**

**Difference between preemptive scheduling and time slicing**

Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence. Under time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

**Sleep method in java**

The sleep() method of Thread class is used to sleep a thread for the specified amount of time.

**Syntax of sleep() method in java**

The Thread class provides two methods for sleeping a thread:

* public static void sleep(long miliseconds)throws InterruptedException
* public static void sleep(long miliseconds, int nanos)throws InterruptedException

**Can we start a thread twice**

No. After starting a thread, it can never be started again. If you does so, an *IllegalThreadStateException* is thrown. In such case, thread will run once but for second time, it will throw exception.

**What if we call run() method directly instead start() method?**

|  |
| --- |
| ◦ Each thread starts in a separate call stack.  ◦ Invoking the run() method from main thread, the run() method goes onto the current call stack rather than at the beginning of a new call stack. |

**The join() method**

The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.

**Naming Thread**

The Thread class provides methods to change and get the name of a thread. By default, each thread has a name i.e. thread-0, thread-1 and so on. By we can change the name of the thread by using setName() method. The syntax of setName() and getName() methods are given below:

1. **public String getName():** is used to return the name of a thread.
2. **public void setName(String name):** is used to change the name of a thread.

**Current Thread**

The currentThread() method returns a reference of currently executing thread.

Priority of a Thread (Thread Priority):

|  |
| --- |
| Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses. |

3 constants defined in Thread class:

|  |
| --- |
| 1. public static int MIN\_PRIORITY  2. public static int NORM\_PRIORITY  3. public static int MAX\_PRIORITY |

**Daemon Thread in Java**

**Daemon thread in java** is a service provider thread that provides services to the user thread. Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.

There are many java daemon threads running automatically e.g. gc, finalizer etc.

You can see all the detail by typing the jconsole in the command prompt. The jconsole tool provides information about the loaded classes, memory usage, running threads etc.

**Points to remember for Daemon Thread in Java**

* It provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.
* Its life depends on user threads.
* It is a low priority thread.

**Why JVM terminates the daemon thread if there is no user thread?**

The sole purpose of the daemon thread is that it provides services to user thread for background supporting task. If there is no user thread, why should JVM keep running this thread. That is why JVM terminates the daemon thread if there is no user thread.

**Methods for Java Daemon thread by Thread class**

The java.lang.Thread class provides two methods for java daemon thread.

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1) | public void setDaemon(boolean status) | is used to mark the current thread as daemon thread or user thread. |
| 2) | public boolean isDaemon() | is used to check that current is daemon. |

**Java Thread Pool**

**Java Thread pool** represents a group of worker threads that are waiting for the job and reuse many times.

In case of thread pool, a group of fixed size threads are created. A thread from the thread pool is pulled out and assigned a job by the service provider. After completion of the job, thread is contained in the thread pool again.

**Advantage of Java Thread Pool**

**Better performance** It saves time because there is no need to create new thread.

**Real time usage**

It is used in Servlet and JSP where container creates a thread pool to process the request.

**ThreadGroup in Java**

Java provides a convenient way to group multiple threads in a single object. In such way, we can suspend, resume or interrupt group of threads by a single method call.

Java thread group is implemented by *java.lang.ThreadGroup* class.

A ThreadGroup represents a set of threads. A thread group can also include the other thread group. The thread group creates a tree in which every thread group except the initial thread group has a parent.

**A thread is allowed to access information about its own thread group, but it cannot access the information about its thread group's parent thread group or any other thread groups.**

**Java Shutdown Hook**

The shutdown hook can be used to perform cleanup resource or save the state when JVM shuts down normally or abruptly. Performing clean resource means closing log file, sending some alerts or something else. So if you want to execute some code before JVM shuts down, use shutdown hook.

**When does the JVM shut down?**

The JVM shuts down when:

* user presses ctrl+c on the command prompt
* System.exit(int) method is invoked
* user logoff
* user shutdown etc.

**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.

**Advantage of Garbage Collection**

* It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
* It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

**Java Runtime class**

**Java Runtime** class is used *to interact with java runtime environment*. Java Runtime class provides methods to execute a process, invoke GC, get total and free memory etc. There is only one instance of java.lang.Runtime class is available for one java application.