**Multithreading**

Table of Contents

[Multithreading 3](#_Toc70339138)

[Multitasking 3](#_Toc70339139)

[1) Process-based Multitasking (Multiprocessing) 3](#_Toc70339140)

[2) Thread-based Multitasking (Multithreading) 3](#_Toc70339141)

[Thread 4](#_Toc70339142)

[Life cycle of a Thread (Thread States) 4](#_Toc70339143)

[1) New 5](#_Toc70339144)

[2) Runnable 5](#_Toc70339145)

[3) Running 5](#_Toc70339146)

[4) Non-Runnable (Blocked) 5](#_Toc70339147)

[5) Terminated 5](#_Toc70339148)

[How to create thread 5](#_Toc70339149)

[Runnable Interface 6](#_Toc70339150)

[Definition 6](#_Toc70339151)

[Thread class 6](#_Toc70339152)

[Definition 6](#_Toc70339153)

[Commonly used Constructors of Thread class: 6](#_Toc70339154)

[Starting a thread: 6](#_Toc70339155)

[getId() 7](#_Toc70339156)

[Thread Class v/s Runnable Interface 8](#_Toc70339157)

[Thread Scheduler in Java 8](#_Toc70339158)

[Sleep method in java 8](#_Toc70339159)

[Syntax of sleep() method in java 8](#_Toc70339160)

[IllegalThreadStateException 8](#_Toc70339161)

[The join() method 8](#_Toc70339162)

[Thread class methods: 8](#_Toc70339163)

[Synchronization in Java 10](#_Toc70339164)

[Why use Synchronization 10](#_Toc70339165)

[Types of Synchronization 10](#_Toc70339166)

[Thread Synchronization 10](#_Toc70339167)

[Concept of Lock in Java 11](#_Toc70339168)

[Mutual Exclusive 11](#_Toc70339169)

[Java Synchronized Method 11](#_Toc70339170)

[Synchronized Block 11](#_Toc70339171)

[Syntax 11](#_Toc70339172)

[Inter-thread communication in Java 12](#_Toc70339173)

[1) wait() method 12](#_Toc70339174)

[2) notify() method 12](#_Toc70339175)

[3) notifyAll() method 12](#_Toc70339176)

[Differences between wait() and sleep() 13](#_Toc70339177)

[Garbage Collection in Java 13](#_Toc70339178)

[Ways for requesting JVM to run Garbage Collector 14](#_Toc70339179)

# Multithreading

## Multitasking

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)

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

* Each process has its own address in memory i.e. each process allocates separate memory area.
* Process is heavyweight.
* Cost of communication between the processes 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 threads is low.

#### Note: At least one thread is required for each process.

## Thread

A thread is a lightweight sub process, a smallest unit of processing. It is a separate path of execution.

Threads are independent, if there occurs exception in one thread, it doesn't affect other threads. It shares a common memory area.



As shown in the above figure, thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS and one process can have multiple threads.

#### Note: At a time one thread is executed only

### Life cycle of a Thread (Thread States)

A thread can be in one of the five states. According to sun, there are **only 4 states** in thread life cycle:

* New
* Runnable
* Non-runnable
* Terminated.

But for better understanding the threads, we are **explaining it in the 5 states**.

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

* New
* Runnable
* Running
* Non-Runnable (Blocked)
* Terminated



### 1) New

The thread is in new state when we create an instance of Thread class before start () method is invoked.

### 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:

* By implementing Runnable interface.
* By extending Thread class

## Runnable Interface

* A class must implement Runnable interface whose instances are intended to be executed by a thread.
* Runnable interface has only one method named run(). Hence, it is a functional interface
* The class must define a method of no arguments called run.
* **public void run():** is used to perform action for a thread.
* This interface is designed to provide a common protocol for objects that wish to execute code while they are active.
* For example, Runnable is implemented by class Thread.
* Being active simply means that a thread has been started and has not yet been stopped.

### Definition

public interface Runnable {

void run();

}

### Thread class

**Thread class** is the main class on which java's multithreading system is based. Thread class provide constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

### Definition

public class Thread

extends Object

implements Runnable

### Commonly used Constructors of Thread class:

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

### Starting a thread:

Thread class method start() is used to start a newly created thread. It performs following tasks:

* A new thread starts.
* The thread moves from new state to the runnable state.
* When the thread gets a chance to execute, its target run () method will run.

### getId()

* The java.lang.Thread.getId() method returns the identifier of this Thread.
* The thread ID is a positive long number generated when this thread was created.
* **The thread ID is unique and remains unchanged during its lifetime.**
* When a thread is terminated, this thread ID may be reused.

### Thread Class v/s Runnable Interface

We must extend thread class when we need to use the methods of the thread class.

If we implement Runnable interface, then inbuilt methods can’t be used.

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

#### Pre-emptive scheduling

The highest priority task executes until it enters the waiting or dead states or another higher priority task comes into existence.

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 - sleep method haults thread execution for a specific period of time

### IllegalThreadStateException

* It is never legal to start a thread more than once.
* Throws: IllegalThreadStateException - if one thread is already started.

### The join() method

* The join() method waits for a thread to die. It causes the currently running threads to stop executing until the thread it joins with completes its task.
* If t is a Thread object whose thread is currently executing, then t. **join()** will make sure that t is terminated before the next instruction is executed by the program.
* join() can be called on a thread after its starts not before.

### Thread class methods:

* public String getName()
* public void setName(String name)
* public long getId()

### Synchronization in Java

Synchronization in java is the capability to control the access of multiple threads to any shared resource.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

### Why use Synchronization

The synchronization is mainly used to

* To prevent thread interference.
* To prevent consistency problem.

### Types of Synchronization

There are two types of synchronization

* Process Synchronization
* Thread Synchronization

Here, we will discuss only thread synchronization.

### Thread Synchronization

There are two types of thread synchronization:

* Mutual Exclusive
* Cooperation (Inter-thread communication in java)

### Concept of Lock in Java

* Synchronization is built around an internal entity known as the lock or monitor.
* Every **object** has a lock associated with it.
* A thread that needs consistent access to an object's fields has to acquire the **object's lock.**
* The **thread releases the lock** when it's done with it.

### Mutual Exclusive

Mutual Exclusive helps keep threads from interfering with one another while sharing data. This can be done by three ways in java:

* By synchronized method
* By synchronized block

### Java Synchronized Method

* If we declare any method as synchronized, it is known as synchronized method.
* Synchronized method is used to **lock an object** for any shared resource.
* When a thread invokes a synchronized method, it **automatically acquires the lock for that object**.
* The **thread releases the lock** when the it completes its task.

### Synchronized Block

* Synchronized block can be used to perform synchronization on any specific resource of the method.
* If we have 50 lines of code in our method, but if we want to synchronize only 5 lines, we can use synchronized block.
* If we put all the codes of the method in the synchronized block, it will work same as the synchronized method.
* Scope of synchronized block is smaller than the method.

### Syntax

synchronized (object reference) {

}

## Inter-thread communication in Java

Inter-thread communication or Co-operation is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed. It is implemented by following methods of **Object class**:

* wait()
* notify()
* notifyAll()

### 1) wait() method

Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public final void wait()throws InterruptedException | waits until object is notified. |
| public final void wait(long timeout)throws InterruptedException | waits for the specified amount of time. |

### 2) notify() method

This method wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. Syntax:

public final void notify()

### 3) notifyAll() method

This method wakes up all threads that are waiting on this object's monitor. Syntax:

public final void notifyAll()

### Differences between wait() and sleep()

|  |  |
| --- | --- |
| **wait()** | **sleep()** |
| wait() method releases the lock | sleep() method doesn't release the lock. |
| is the method of Object class | is the method of Thread class |
| is the non-static method | is the static method |
| should be notified by notify() or notifyAll() methods | after the specified amount of time, sleep is completed. |

* A thread is like a separate program, executing concurrently.
* We can write Java programs that deal with many tasks at once by defining multiple threads.
* The main advantage of multi-threading is that it doesn't occupy memory for each thread.
* It shares a common memory area.
* Threads are important for multi-media, web applications, servers etc

# Garbage Collection in Java

In C/C++, programmer is responsible for both creation and destruction of objects. Small negligence in doing so can cause the entire program to terminate abnormally**.**

In Java, this freeing up of memory is not the programmar's job, its internally handled by Garbage collector

Main objective of Garbage Collector is to free **heap memory** by destroying **unreachable objects.**

**Unreachable objects:** An object is said to be unreachable if it doesn’t contain any reference to it.

|  |
| --- |
| Integer i = new Integer(4);  // the new Integer object is reachable via the reference in 'i'  i = null;  // the Integer object is no longer reachable. |

**Eligibility for garbage collection:** An object is said to be eligible for GC (garbage collection) if it is unreachable. In above example, after i = null; integer object in heap area is eligible for garbage collection.

### ****Ways for requesting**** [JVM](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/) ****to run Garbage Collector****

* Once an object is eligible for garbage collection, it may or may not be destroyed immediately by garbage collector. Whenever JVM runs Garbage Collector program, then only object will be destroyed.
* We can also request JVM to run Garbage Collector. There are two ways to do it :
  + **Using System.gc() method** : System class contain static method gc() for requesting JVM to run Garbage Collector.
  + **Using Runtime.getRuntime().gc() method** : Runtime class allows the application to interface with the JVM in which the application is running. Hence by using its gc() method, we can request JVM to run Garbage Collector.