**Multithreading in Java**

A thread is light weight process that can execute simultaneously. Multithreading in java allows multiple threads to run simultaneously.

A thread is light weight process that can execute simultaneously. Multithreading in java allows multiple threads to run simultaneously.

* A program can be divided into two or more parts that can run concurrently

and each part of such program is known as thread.

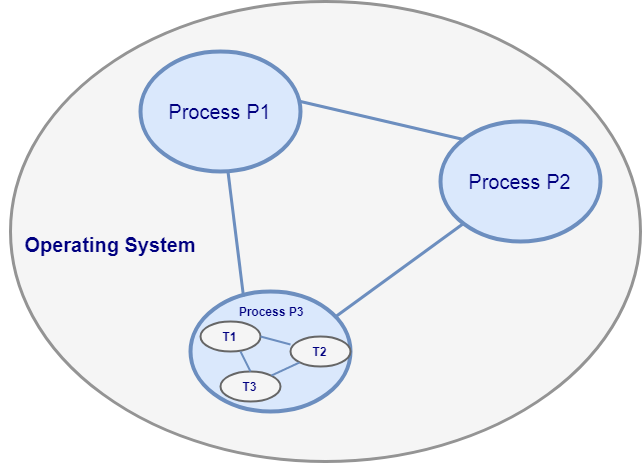
* In a simple word, thread is a lightweight sub-process.

 It is a smallest unit of processing.

* When these threads execute concurrently for accomplishing a task is

known multithreading.

* In multithreading, each thread defines a separate path of execution.
* In multithreading, every thread has a priority between 1 to 10. Thread with highest priority get preference over lower priority thread for execution.



Java Process and Thread Diagram

 In a java programming, Thread is a predefined class within a java.lang package.

Operating system (OS) can contain multiple process, in the above figure, there are three process  P1,P2 & P3, within OS and within P3 process there are three threads T1, T2& T3. There is context-switching between the threads.

## Thread Creation in Java

We can create a thread by two ways:

1. By extending Thread
2. By implementing Runnable interface

## C**reating Thread By Implementing Runnable Interface**

1. We can create a thread by creating a class that implements the Runnable interface.
2. To implement Runnable, a class only need to implement a single method called run().
3. Inside run () we will define the code that constitute the new thread.
4. When a new thread creates, it do not executes until we call its start () method. When start () method executes it call to run() method.

# Java Runnable Interface

The **Runnable interface** is a **functional interface** defined in **java**. lang package.

The runnable interface has an undefined method **run()** with void as return type, and it takes in no arguments. The method summary of the run() method is given below-

public void run().

The runnable interface provides a standard set of rules for the instances of classes which wish to execute code when they are active. The most common use case of the Runnable interface is when we want only to override the run method. When a thread is started by the object of any class which is implementing Runnable, then it invokes the run method in the separately executing thread.

## **Implementing Runnable**

It is the easiest way to create a thread by implementing Runnable. One can create a thread on any object by implementing Runnable. To implement a Runnable, one has only to implement the run method.

**public void run()**

In this method, we have the code which we want to execute on a concurrent thread. In this method, we can use variables, instantiate classes, and perform an action like the same way the main thread does. The thread remains until the return of this method. The run method establishes an entry point to a new thread.

## **How to create a thread using Runnable interface**

To create a thread using runnable, use the following code-

1. Runnable runnable = **new** MyRunnable();
3. Thread thread = **new** Thread(runnable);
4. thread.start();

The thread will execute the code which is mentioned in the run() method of the Runnable object passed in its argument.

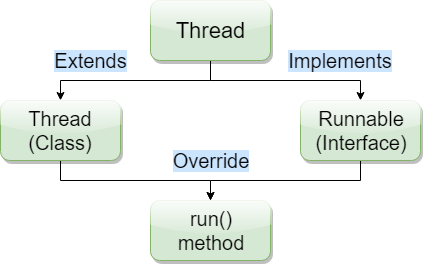
## **Use of Runnable class in network programming**

The runnable class is used to perform multi-thread programming, especially in server-side as a server may be getting several requests from different clients. To tackle this in a fast and resource-efficient way, we use multi-thread programming.

**Example of a networking program using Runnable-**

The following program shows a server program which creates a thread, then creates a socket and waits for a client to connect to it and asks for an input string

## **Thread vs. Runnable**



here are several differences between Thread class and Runnable interface based on their performance, memory usage, and composition.

* By extending thread, there is overhead of additional methods, i.e. they consume excess or indirect memory, computation time, or other resources.
* Since in Java, we can only extend one class, and therefore if we extend Thread class, then we will not be able to extend any other class. That is why we should implement Runnable interface to create a thread.
* Runnable makes the code more flexible as, if we are extending a thread, then our code will only be in a thread whereas, in case of runnable, one can pass it in various executor services, or pass it to the single-threaded environment.
* Maintenance of the code is easy if we implement the Runnable interface.

## **Creating Thread By Extending Thread** **Class**

1. To create a thread we will create a thread a class that extends Thread class.
2. To extend Thread class, a class only needs to implement a single method called run (). This extending class must override the run() method
3. Inside run () we will define the code that constitute the new thread.
4. When a new thread creates, it do not executes until we call its start () method. When start () method executes it call to run() method.

# Thread Life Cycle in Java

Life cycle of Thread tells the various information of thread form born to terminate. Thread life cycle contains the several stages and at a time any thread can be present  in a single state.

States of Thread life cycle

1. Born
2. Ready
3. Running
4. Blocked
5. Sleep
6. Wait
7. Dead