**What is Java Executor Framework?**

With the increase in the number of cores available in the processors nowadays, coupled with the ever-increasing need to achieve more throughput, multi-threading APIs are getting quite popular. Java provides its own multi-threading framework called the Java Executor Framework.

Java executor framework (java.util.concurrent.Executor), released with the JDK 5 is used to run the Runnable objects without creating new threads every time and mostly re-using the already created threads. We all know that there are two ways to create a thread in java. If you want to read more about their comparison, read how to create threads in Java.

The java.util.concurrent.Executors provide factory methods that are being used to create ThreadPools of worker threads. Thread pools overcome this issue by keeping the threads alive and reusing the threads. Any excess tasks flowing in that the threads in the pool can handle are held in a Queue. Once any of the threads get free, they pick up the next task from this queue. This task queue is essentially unbounded for the out-of-box executors provided by the JDK.

**Some types of Java Executors are listed below:**

1. SingleThreadExecutor
2. FixedThreadPool(n)+
3. CachedThreadPool
4. ScheduledExecutor

Let us discuss these popular java executors to some details what exactly they do to get a better idea prior to implementing the same.

**Executor 1:** SingleThreadExecutor

A single thread pool can be obtained by calling the static *newSingleThreadExecutor()* method of the Executors class. It is used to execute tasks sequentially.

**Syntax:**

ExecutorService executor = Executors.newSingleThreadExecutor();

**Executor 2:** FixedThreadPool(n)

As the name indicates, it is a thread pool of a fixed number of threads. The tasks submitted to the executor are executed by the n threads and if there is more task they are stored on a LinkedBlockingQueue. It uses Blocking Queue.

**Syntax:**

ExecutorService fixedPool = Executors.newFixedThreadPool(2);

**Executor 3:** CachedThreadPool

Creates a thread pool that creates new threads as needed, but will reuse previously constructed threads when they are available. Calls to execute will reuse previously constructed threads if available. If no existing thread is available, a new thread will be created and added to the pool. It uses a SynchronousQueue queue.

ExecutorService executorService = Executors.newCachedThreadPool();

**Executor 4:**ScheduledExecutor

Scheduled executors are based on the interface ScheduledExecutorService which extends the ExecutorService interface. This executor is used when we have a task that needs to be run at regular intervals or if we wish to delay a certain task.

ScheduledExecutorService scheduledExecService = Executors.newScheduledThreadPool(1);

* The tasks can be scheduled using either of the two methods:
  + **scheduleAtFixedRate**: Executes the task with a fixed interval, irrespective of when the previous task ended.
  + **scheduleWithFixedDelay**: This will start the delay countdown only after the current task completes.`

**Syntax:**

scheduledExecService.scheduleAtFixedRate(Runnable command, long initialDelay, long period, TimeUnit unit)

scheduledExecService.scheduleWithFixedDelay(Runnable command, long initialDelay, long period, TimeUnit unit)

*Future Object*

*The result of the task submitted for execution to an executor can be accessed using the java.util.concurrent.The future object returned by the executor. Future can be thought of as a promise made to the caller by the executor. The future interface is mainly used to get the results of Callable results. whenever the task execution is completed, it is set in this Future object by the executor.*

**Syntax:**

Future<String> result = executorService.submit(callableTask);

**Implementation:** Creating and Executing a Simple Executor in which we will create a task and execute it in a fixed pool

* The Task class implements Callable and is parameterized to String type. It is also declared to throw Exception.
* Now in order to execute task in class “Task” we have to instantiate the Task class and are passing it to the executor for execution.
* Print and display the result that is returned by the Future object

**Example**

// Java Program demonstrating Introduction to Java Executor

// Framework

// Importing concurrent classes from java.util package

import java.util.concurrent.\*;

// Class 1

// Helper Class implementing runnable interface Callable

class Task implements Callable<String> {

// Member variable of this class

private String message;

// Constructor of this class

public Task(String message)

{

// This keyword refers to current instance itself

this.message = message;

}

// Method of this Class

public String call() throws Exception

{

return "Hiiii " + message + "!";

}

}

// Class 2

// Main Class

// ExecutorExample

public class GFG {

// Main driver method

public static void main(String[] args)

{

// Creating an object of above class

// in the main() method

Task task = new Task("testing executor");

// Creating object of ExecutorService class and

// Future object Class

ExecutorService executorService

= Executors.newFixedThreadPool(4);

Future<String> result

= executorService.submit(task);

// Try block to check for exceptions

try {

System.out.println(result.get());

}

// Catch block to handle the exception

catch (InterruptedException

| ExecutionException e) {

// Display message only

System.out.println(

"Error occurred while executing the submitted task");

// Print the line number where exception occurred

e.printStackTrace();

}

// Cleaning resource and shutting down JVM by

// saving JVM state using shutdown() method

executorService.shutdown();

}

}

|  |
| --- |
|  |

**Output:**

Hiiii testing executor