Day 14 – Executor Framework & Thread Group, Thread Pool

Day 13 Revisit – Concurrency in JAVA.

Serial vs parallel Operation (Sync vs Async)

What is Process??

What is Thread??

Life Cycle of Thread.

Different ways of creating Thread in JAVA (Using Thread class, Runnable interface)

Managing multiple Threads.

Properties of Threads – id, Name, priority, State, (NEW, RUNNABLE, RUNNING, BLOCKED/WAITING, TERMINATED), isAlive(), isDaemon(), isTerminated()

Start() & run()

Runnable interface is a Functional Interface. (Only one abstract method – run())

<https://www.geeksforgeeks.org/java/what-is-java-executor-framework/> -- Executor Framework.

RxJAVA – Reactive JAVA. <https://reactivex.io/>

Web Server – Tomcat –

Design Pattern – Is a proven way to resolve a particular problem/challenge.

Chennai – Mumbai

1) By Air

2) By Road

3) By Train

4) By Walk

5) By ship.

1. Creational – Factory, AbstractFactory, Builder,
2. Behavioral

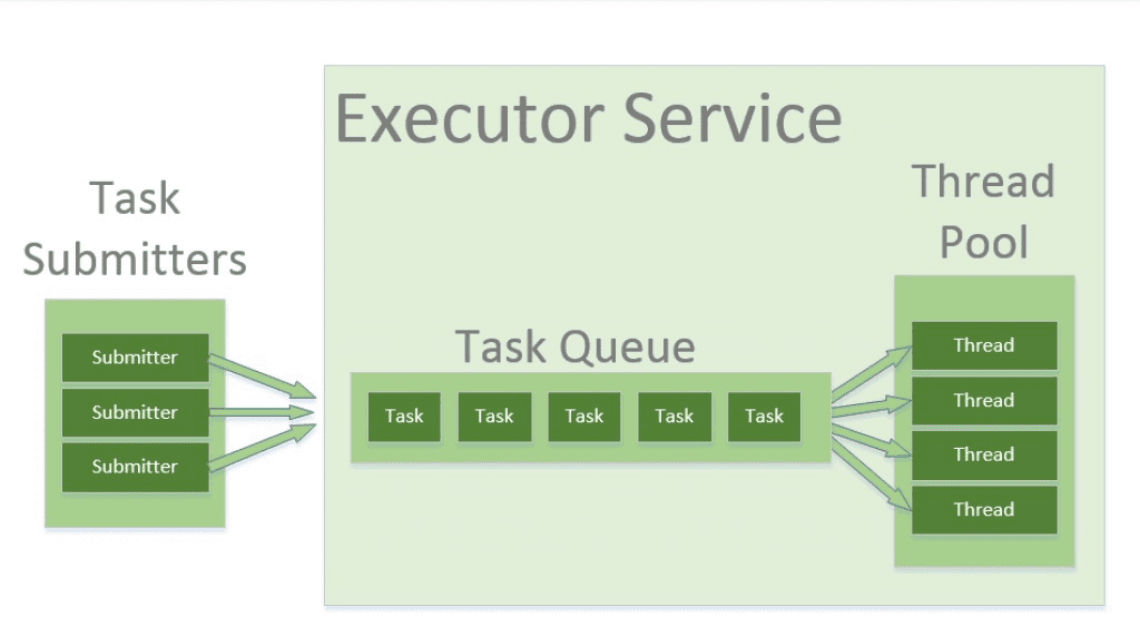
Observer – Async/ Parallel Programming.

Executor Framework – It’s a way of performing parallel processing.

Java provides the Executor Framework (java.util.concurrent.Executor, introduced in JDK 5) to run Runnable tasks without creating new threads every time, by reusing existing ones.

The Executors class offers factory methods to create thread pools.

* Threads stay alive and are reused.
* Extra tasks are placed in a queue and free threads pick them up later.
* By default, these queues are unbounded in JDK executors.



Data Structures

1. Array (Helps to store data in sequential memory location) [adding/removing element at the end of array --- adding/removing element in the middle or beginning]
2. LinkedList (Adding/removing element in the beginning or middle --- adding/removing element at the end)
3. Stack [LIFO – Last in First Out]
4. Queue [FIFO – First In First Out] – Event Driven Arch.

Types of Executor Service

Below are the commonly used executor types:

1. SingleThreadExecutor -- *ExecutorService executor = Executors.newSingleThreadExecutor();*
2. FixedThreadPool(n) --- *ExecutorService fixedPool = Executors.newFixedThreadPool(2);*
3. CachedThreadPool -- *ExecutorService executorService = Executors.newCachedThreadPool(); [Creates new Threads when no other Thread is available for processing the Task]*
4. ScheduledExecutor -- *ScheduledExecutorService scheduledExecService = Executors.newScheduledThreadPool(1); [Run at particular interval or schedule it later time]*

**package** com.example;

**import** java.util.concurrent.Callable;

**import** java.util.concurrent.ExecutionException;

**import** java.util.concurrent.ExecutorService;

**import** java.util.concurrent.Executors;

**import** java.util.concurrent.Future;

**class** Task **implements** Callable<String> {

**private** String message;

**public** Task(String message)

{

**this**.message = message;

}

**public** String call() **throws** Exception

{

**return** "Hi " + message + "!";

}

}

**public** **class** ExecutorDemo {

**public** **static** **void** main(String[] args)

{

Task task = **new** Task("First Task");

// 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) {

System.***out***.println("Error occurred while executing the submitted task");

e.printStackTrace();//Resource consuming operation or costly operation

System.***out***.println("Exception :" + e.getMessage());

}

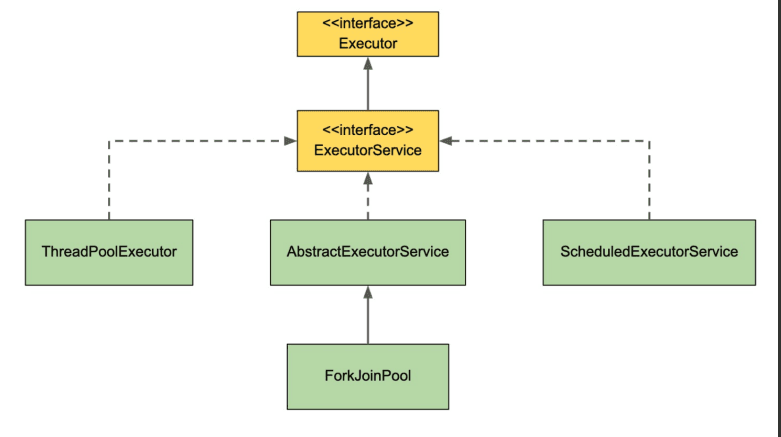
// Cleaning resource and shutting down JVM by saving JVM state using shutdown() method

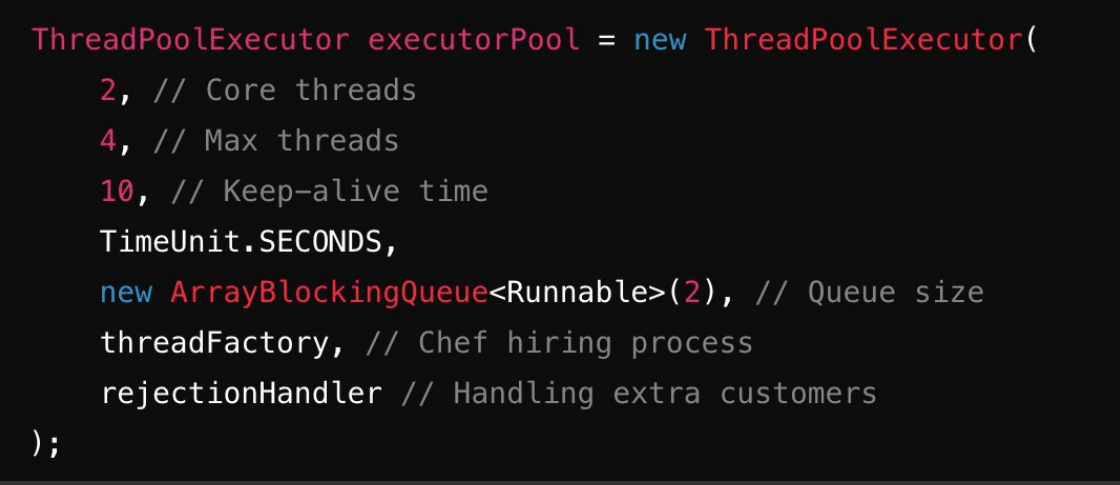
executorService.shutdown();

}

}

<https://www.codingshuttle.com/spring-boot-handbook/multi-threading-java-executor-framework/>





Dead Lock Example

**package** com.example;

**public** **class** DeadLockExample {

**public** **static** Object *Lock1* = **new** Object();

**public** **static** Object *Lock2* = **new** Object();

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

ThreadDemo1 T1 = **new** ThreadDemo1();

ThreadDemo2 T2 = **new** ThreadDemo2();

T1.start();

T2.start();

}

**private** **static** **class** ThreadDemo1 **extends** Thread {

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

**synchronized** (*Lock1*) {

System.***out***.println("Thread 1: Holding lock 1...");

**try** {

Thread.*sleep*(10); // Simulate some work

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.***out***.println("Thread 1: Waiting for lock 2...");

**synchronized** (*Lock2*) {

System.***out***.println("Thread 1: Holding lock 1 & 2...");

}

}

}

}

**private** **static** **class** ThreadDemo2 **extends** Thread {

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

**synchronized** (*Lock2*) {

System.***out***.println("Thread 2: Holding lock 2...");

**try** {

Thread.*sleep*(10); // Simulate some work

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.***out***.println("Thread 2: Waiting for lock 1...");

**synchronized** (*Lock1*) {

System.***out***.println("Thread 2: Holding lock 2 & 1...");

}

}

}

}

}

Instead of Manually handling Group of thread, Java has given an API called Executor Framework to manage Multiple Threads effectively.

java.util.concurrent package.

1. Executor
2. ExecutorService
3. Executors (utility Class)

Helps to run Multiple Threads effectively.

Reactive refers Parallel or Async Operations.