Java : ScheduledThreadPoolExecutor

"Concept && Coding" YT Video Notes

shutdown vs await Termination vs shutdownNow

Shutdown:

- Initiates orderly shutdown of the ExecutorService.
- After calling 'Shutdown', Executor will not accept new task submission.
- Already Submitted tasks, will continue to execute.

AwaitTermination:

- Its an Optional functionality. Return true/false.
- It is used after calling 'Shutdown' method.
- Blocks calling thread for specific timeout period, and wait for ExecutorService shutdown.
- Return true, if ExecutorService gets shutdown withing specific timeout else false.

shutdownNow:

- Best effort attempt to stop/interrupt the actively executing tasks
- Halt the processing of tasks which are waiting
- Return the list of tasks which are awaiting execution.

Scenario1: Task submission after Shutdown

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

    ExecutorService poolObj = Executors.newFixedThreadPool( nThreads: 5);
    poolObj.submit(() -> {
        System.out.println("Thread going to start its work");
    });

    poolObj.shutdown();

    poolObj.submit(() -> {
        System.out.println("Thread going to start its work");
    });

}
```

```
Exception in thread "main" java.util.concurrent.RejectedExecutionException Create breakpoint : Task java.util.concurrent.FutureTask@4b409585 rejected
    at java.util.concurrent.AbstractExecutorService.submit(AbstractExecutorService.java:112)
```

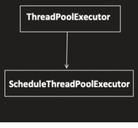
Scenario2: Shutdown do not impact the already submitted task

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

    ExecutorService poolExecutorObj = Executors.newFixedThreadPool( nThreads: 5);
    poolExecutorObj.submit(() -> {
        try {
            Thread.sleep( millis: 5000);
        }catch (Exception e){

        }
        System.out.println("new task");
    });

    poolExecutorObj.shutdown();
    System.out.println("Main thread unblocked and finished processing");
}
```

Scenario3: usage of 'awaitTermination'

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

    ExecutorService poolExecutorObj = Executors.newFixedThreadPool( nThreads: 5);
    poolExecutorObj.submit(() -> {
        try {
            Thread.sleep( millis: 6000);
        } catch (Exception e) {

        }
        System.out.println("new task");
    });

    poolExecutorObj.shutdown();
    try {
        boolean isExecutorTerminated = poolExecutorObj.awaitTermination( timeout: 1, TimeUnit.SECONDS);
        System.out.println("Main thread, isExecutorTerminated: " + isExecutorTerminated);
    } catch (Exception e) {

    }
}
```

ScheduledThreadPoolExecutor : Helps to schedule the tasks



S.No.	Method Name	Description
1.	schedule(Runnable command, long delay, TimeUnit unit)	Schedules a Runnable task after specific delay. Only one time task runs.
2.	schedule(Callable<V> callable, long delay, TimeUnit unit)	Schedules a Callable task after specific delay. Only one time task runs.
3.	scheduleAtFixedRate(Runnable command, long initialDelay, long period, TimeUnit unit) `public static void main(String args[]) { ... }`	Schedules a Runnable task for repeated execution with fixed rate. We can use cancel method to stop this repeated task. Also lets say, if thread1 is taking too much time to complete the task and next task is ready to run, till previous task will not get completed, new task can not be start (it will wait in queue).
4.	scheduleWithFixedDelay(Runnable command, long initialDelay, long delay, TimeUnit unit)	Schedules a Runnable task for repeated execution with a fixed delay (Means next task delay counter start only after previous one task completed)