Executor Framework in Java

This framework contains components that are used to efficiently manage multiple threads.

This is centered around the Executor Interface and its subinterface ExecutorService and the class ThreadPoolExecutor.

By using the executor, one has to implement the runnable objects and send them to the executor for execution.

The different types of Executors are:

① Single Thread Executor

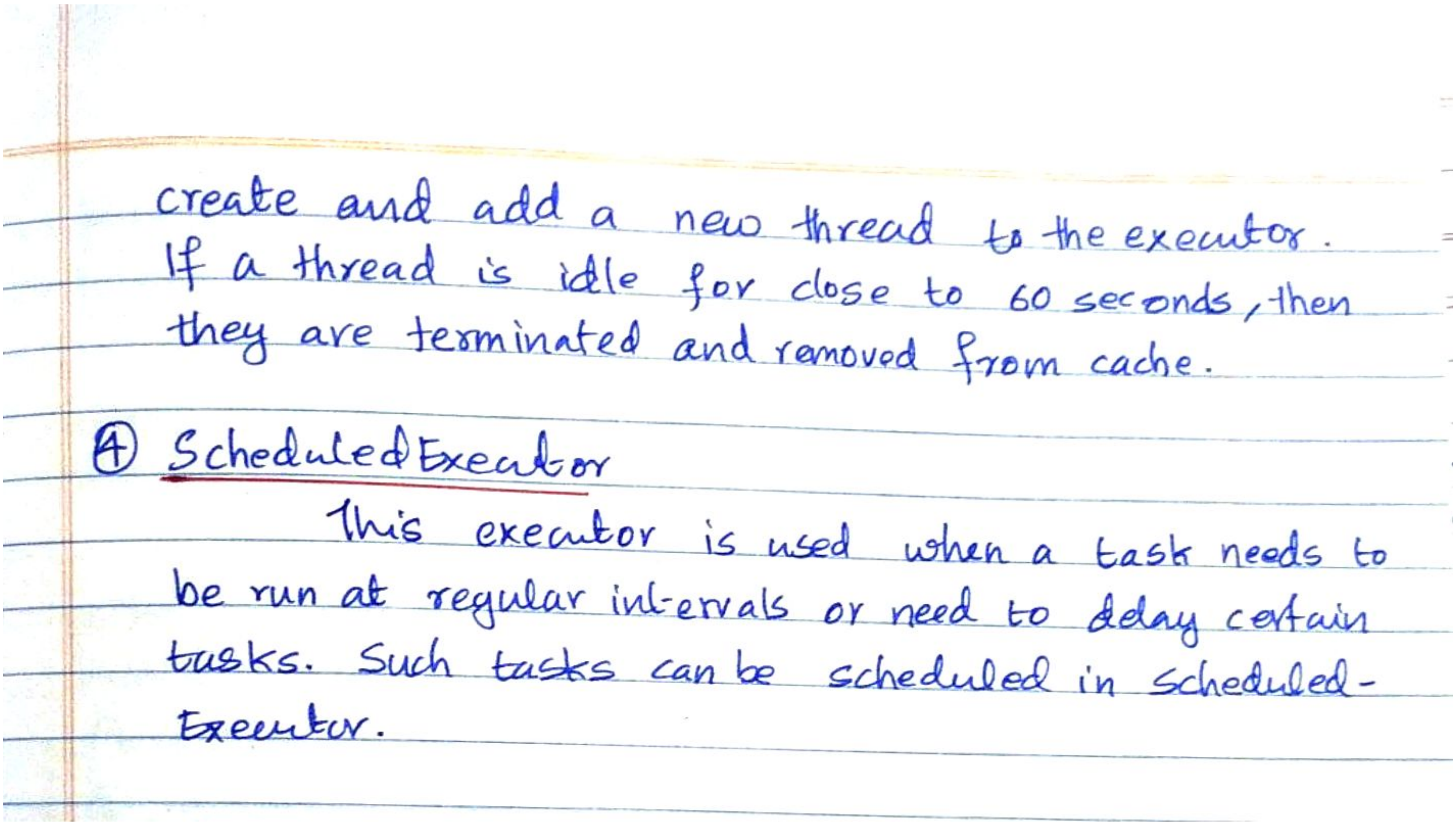
This executor has only one thread and is used to execute tasks in a sequential manner.

② FixedThreadPool - FixedPoolExecutor

This's is a pool of fixed number of threads. The tasks will be executed by these 'n' threads and if there are more than 'n' tasks then they are stored in LinkedBlockingQueue.

③ CachedThreadExecutor

This threadpool is not bounded by the number of threads. If all the threads are busy executing some tasks and when a new task arrives, it will



create and add a new thread to the executor. If a thread is idle for close to 60 seconds, then they are terminated and removed from cache.

④ Scheduled Executor

This executor is used when a task needs to be run at regular intervals or need to delay certain tasks. Such tasks can be scheduled in ScheduledExecutor.

Executor : This interface is used to submit new task. It has a method called "execute".

ExecutorService : This is the sub interface of Executor. This interface provides methods to manage and execute tasks

Executors : This class provides factory methods for creating thread pool.

① newFixedThreadPool ( )

② newCachedThreadPool ( )

③ newSingleThreadedExecutor ()

④ newScheduledThreadPool ()

ThreadPoolExecutor : This is actual implementation of ThreadPool. ThreadPoolExecutor can be created using factory methods of Executors.

Java Progoam to show the use of Exeutor Framewark

```
① import java.util.concurrent.*;

④ class for main()
public class ExeutorTest {
    public static void main (String[]args) {
    ⑤ ExeutorServics es = Exeautors.newFixedThreadPool(3);
       for (int i = 1; i<=5; i++) {
                Creating tasks
        ⑥ WorkerThread work = new WorkerThread(" "+i);
       ⑦ es.execute (worker);
        }
      ⑧ es.shutdown();
       while (! es.isTerminated()) { }
        System.out.println(" All threads finished");
    }
}
② Create class for task using Runnable
class WorkerThread Implements Runnable {
    private Stoing msg;
    public WorkerThread (Stoing m) { msg=m; }
    use run() for implemonting task
③  public void run() {
        System.out.pointln (Thread.currentThread().getName() +
                          " message = " + message);
        try { Thread.sleep(1000); }
        catch (InterruptedException e) { System.out.println(e);}
        System.out.println (Thread.currentThread.getName() +
                          " End");
    }
}
```

Java Program using Callable for Task

```
① Importing classes/ interfaces from concurrent package
import java. concurrent.*;
② Create a class for assigning task by implementing Callable
class Task implements Callable <String> {

      private String msg;
      ③ Constructor of class to initalize members (optional)
      public Task (String m) { msg=m; }

      ④ Method to implement task
      public String call () throws Exception {
            return "Hi"+ msg + "!";
      }
}

public class DemoCallable {
   public static void main (String args[]) {
         ⑤ Creating object of the task class
         Task t = new Task ("Amrita");
      ⑥ ExecutorService es = Executors. newFixedThreadPool(3)
      ⑦ Future <string> f = es.submit (t);
         try {
         ⑧    S.O.P (f. get());
         } catch (Exception e) { system.out.println(e);}
         es.shutdown();
   }
}
```

Difference between Callable & Runnable

Runnable	Callable
① Introduced in Java 1.0	Introduced in Java 1.5
② Does not return any value	Has a return type and can get the result using get() of Future class.
③ Use run()	Use call()
④ Doesn't throw any exception	Throws checked exceptions