Agenda :-

1) Executor framework

2) Callables

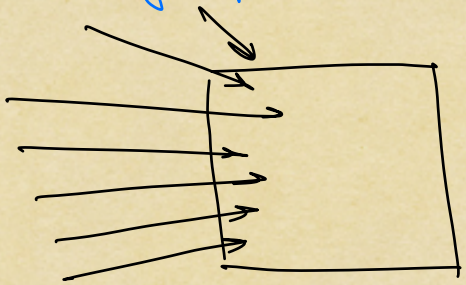
```
Thread t = new Thread ( Obj );
t.start();
```

→ spawns a new thread and calls the run method on it

* once the run() method completes execution, thread is dead. thread object will get cleared, and thread is not reusable

* Building a server :-

every request → we need one separate thread



with high number of requests no of thread creation goes very high, hence

→ lot of cpu usage

→ lot of memory usage

→ lot of context switching

When we write a multi-threaded appn, there are 2 things, we need to do:-

[business logic]

* figuring out the tasks you need to do, in multiple threads independently

* creation of threads and running the threads

↳ give up responsibility for this or yes

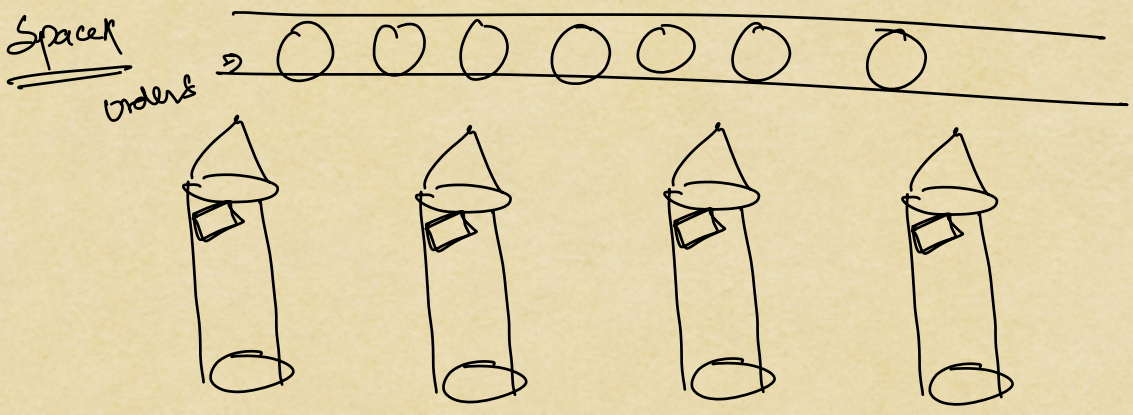
Problems

1) Multiple new thread creations is not an optimised way of doing things

2) Dev. should not worry about thread creation & thread start.

Solution

* Executor framework or, Executor Service



* till now, we were creating a new thread for each task, each time

* now, by using Executor framework, we will reuse the threads

→ Executor framework maintains a ThreadPool

* each thread after completion of its task, doesn't get destroyed but is kept in thread pool for further reuse

→ Executor framework maintains a waiting queue

* all the tasks that needs a thread, waits in queue until a thread becomes available

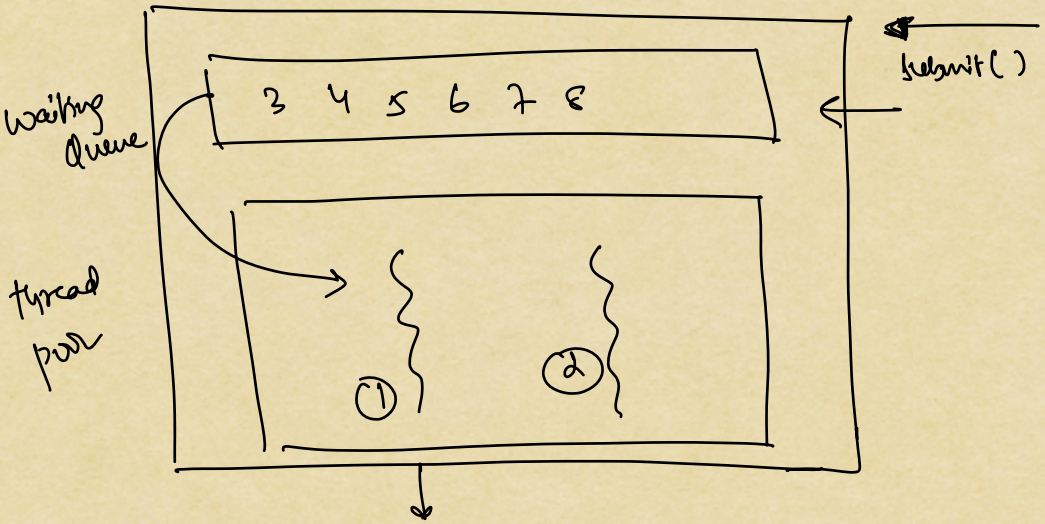
→ Executor framework sets a no. of threads for thread pool.

(EF)
→ Executor Frameworks
→ maintains
- ThreadPool [worker]
- WaitingQueue for task
→ [WorkQueue]

→ We submit of runnable task to executor.

→ EF creates the thread

→ EF assigns a task to a thread.

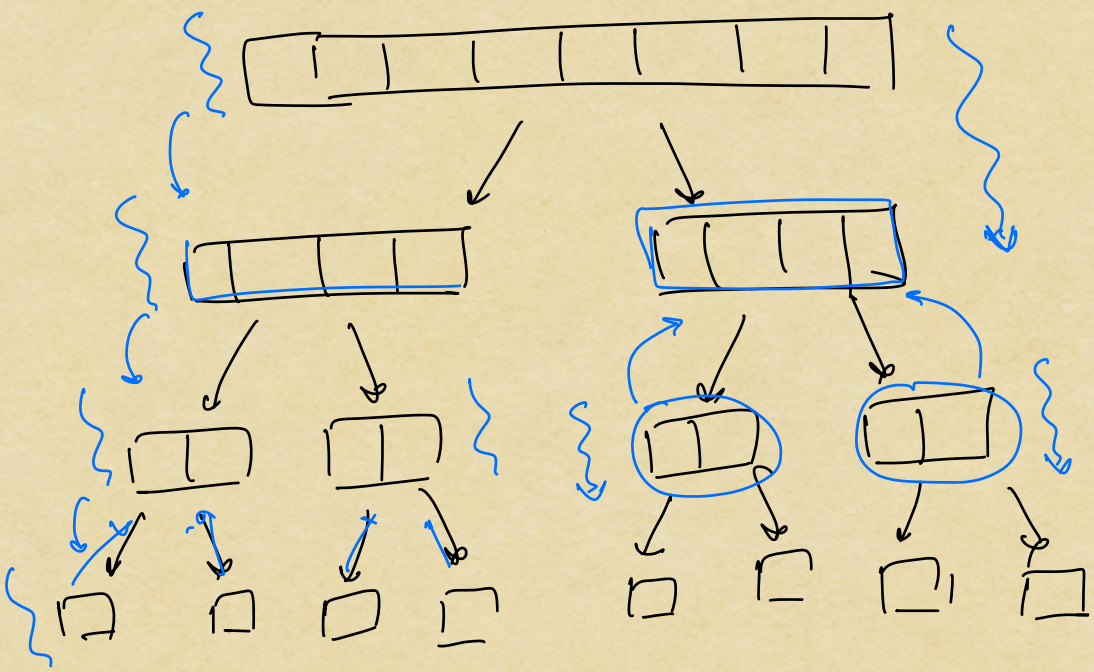


print (1 - 8)

print (1)

threads = 2

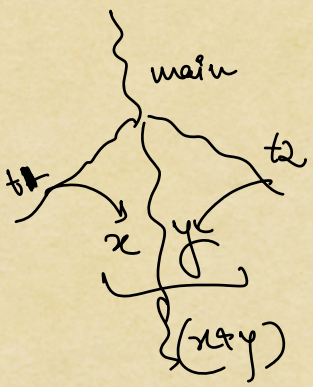
→ Merge Sort



⇒ Since, Runnable run() doesn't allow us to return anything, we use Callable → call() to write multi-threaded which can return data.

Question

→ Generate 2 random numbers and add them, each random number should be generated by a diff thread.



Step 1 Identify the task that you want to do parellely

Generate a random number

Step 2 Create classes for all the tasks that has been identified

class RandomNumberGenerator

* name of the class should be a noun doing a work.

Step 3 Make the class implement Callable interface

class RandomNumberGenerator implements Callable<Integer>

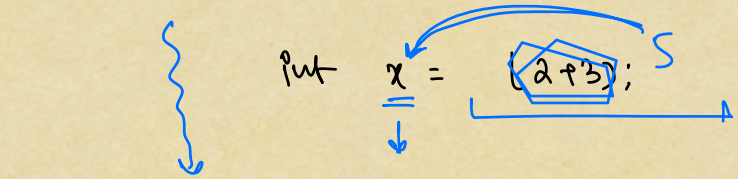
* Callable interface has a method called Call(); this method holds the code for the task that we want to do, and returns the data.

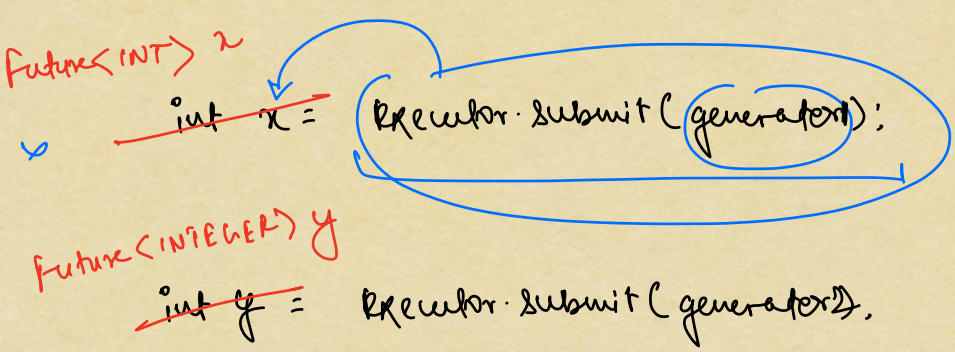
Step 3 ⇒ Implement the call method

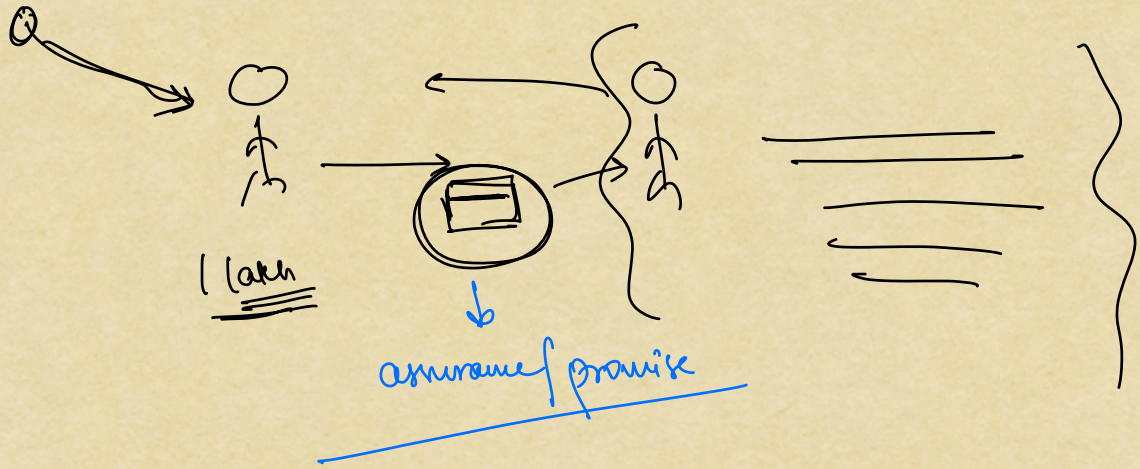
```
class RandomNumberGenerator implements Callable<Integer>{

    public Integer call() {

    }

}
```

⇒ Task defination from Step 1 to 4

⇒ Submit the task to Executor framework






Llakta
assurance / promise