AGENDA

1.) Threads continue

2.) Executors

3.) callables / futures

4.) Adder subtractor

start by 9:05 PM

Print 1-1000 - Each in New Thread.

S1.) PrintNo

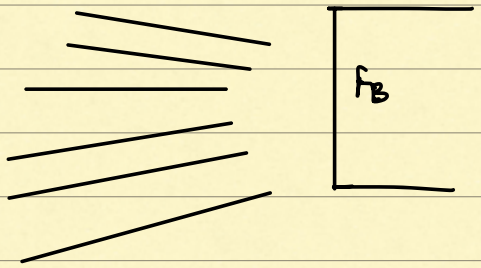
S2.) PrintNumberTask

S3.) implement runnable

S4.) run() { .... }

S5.) create threads.

PROBLEM:

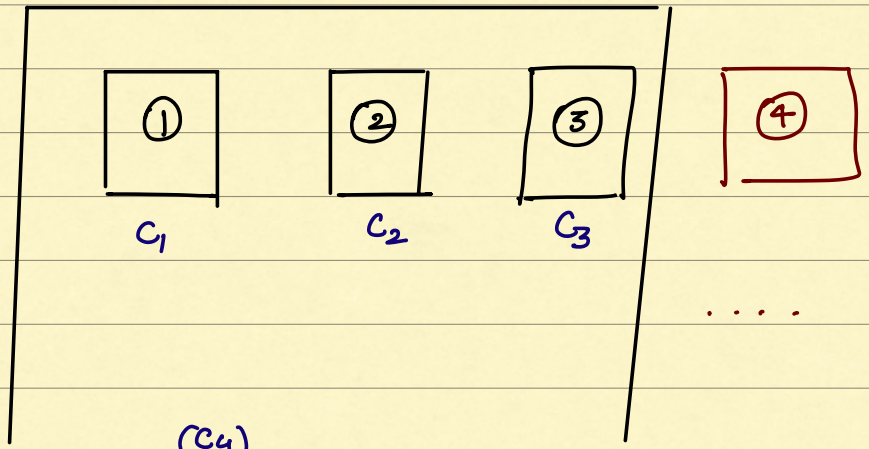


1 request = 1 thread

XXXX

Analogy →

Car factory

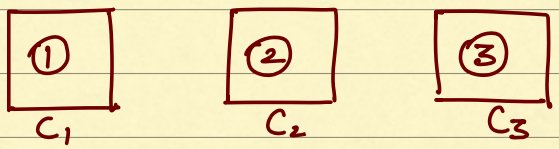


(C4)

22

XUV 700.

10



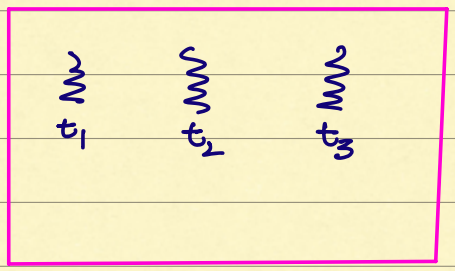
C4

C5

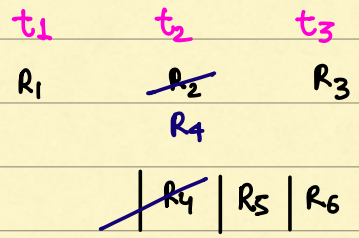
C6

C7

•) re-using Production line



Thread Pool

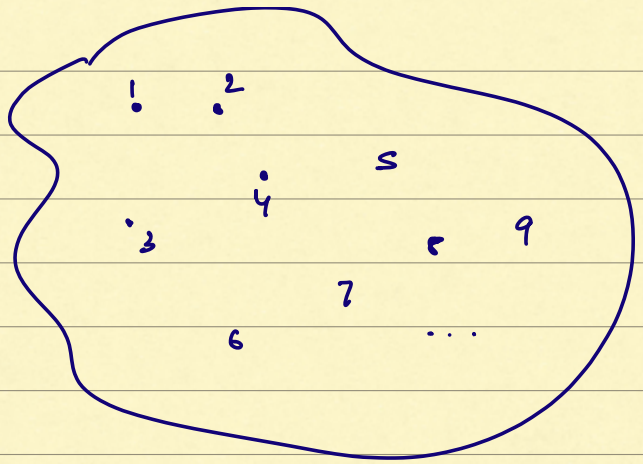


Executor service → framework for Thread Pool.

Advantages:

1.) saves cost

2.) optimising CPU



1 1000

t_1 ..... t_{1000}

t_1 —— ....

t_2 ..... —

(N) 1150

(50 7)

| 57 | ... -

void sum(){

.....

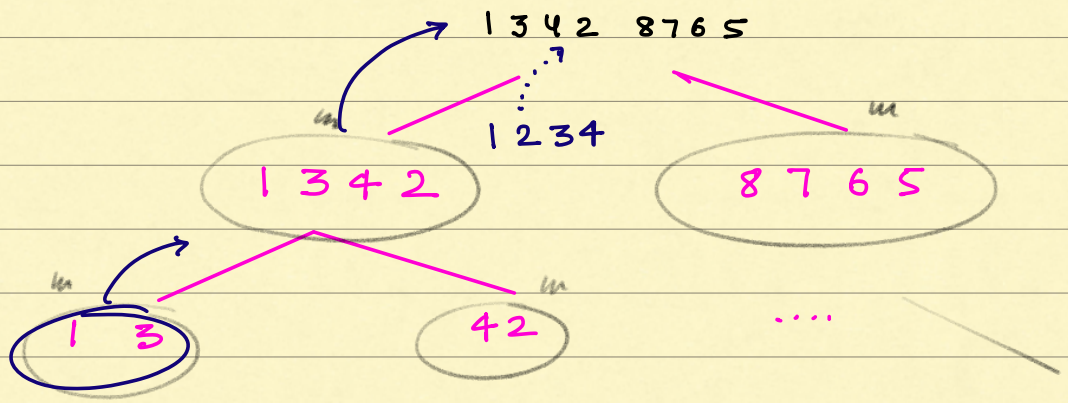
}

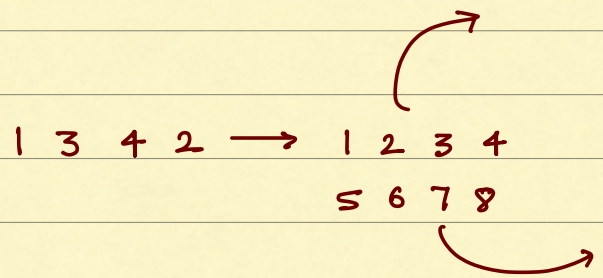
*) CALLABLE:

Interfaces which return data.

MergeSort

divide N conq.





Returntype → [] int

steps:

S1.) Identify task

S2.) create class

S3.) Implement callable*

S4.) Add code

S5.) Create threads

*) FUTURE:

object in multithreaded env.

① Print(a)

② → t_1 int i = es.execute(task); #

③ Print(b)

④ Print(i)

main t_1

solution → use future objects.

Future<int> i

① Print(a)

② Future<Int>i= es.execute(task); #

③ Print(b)

④ Print(i) (blocking call)

main → ④

t_1 → ....

your code (main thread) will wait

(blocking call)

i.get()

Print(i.get())

Note: whenever using callables, use future<v>
as well.