Future ⇒ It is a placeholder / bucket for the value that callable might return in the future, this unblocks the current to complete initialisation and move ahead.

Array adder ⇒ [1, 2, 3, 4, 5]

Integer call();

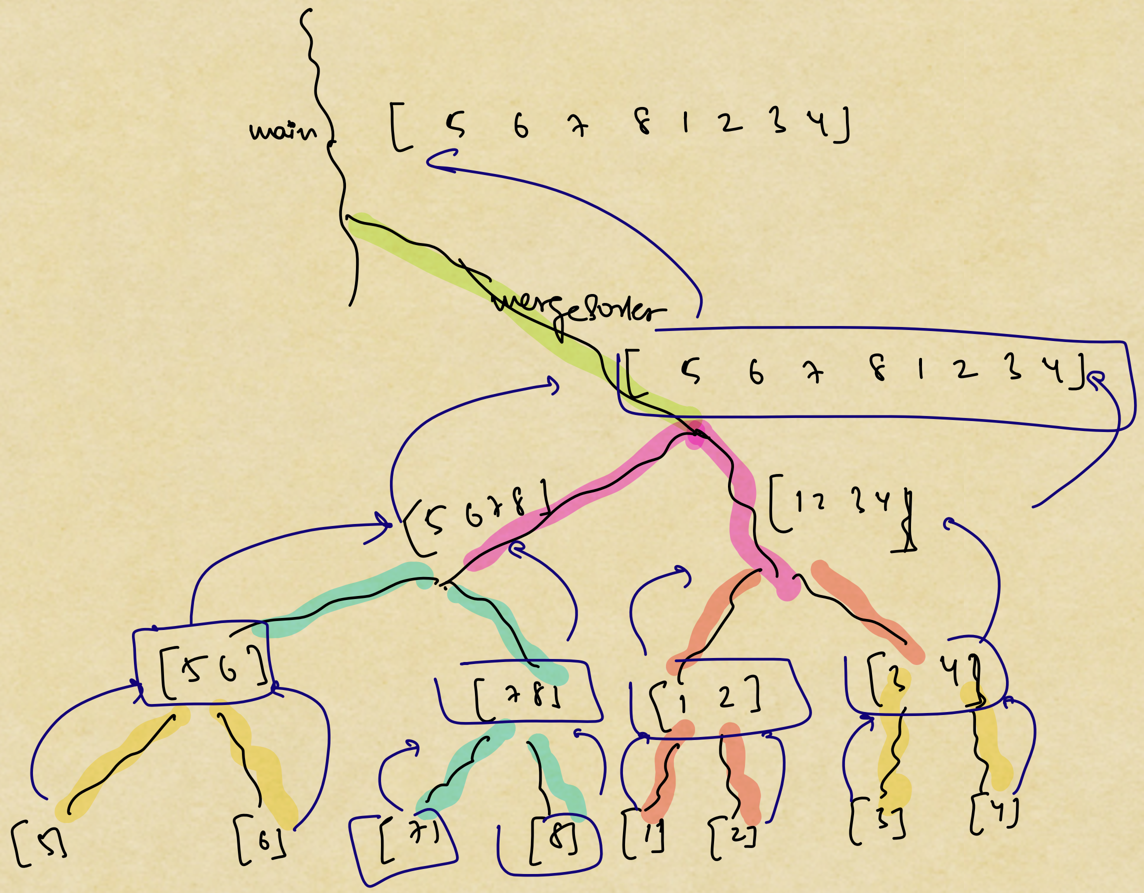
$$\text{int sum} = \text{executor.submit}(\overset{\text{adder}}{\downarrow}) ;$$

(I) calling the executor the start executing the adder task

(II) whatever adder will return will be initialised to sum variable

Future < Integer > sum = executor. submit (sum);

$$\boxed{\text{sum.}\underline{\underline{\text{get}}}()} \leftarrow \text{gets the actual } \underline{\underline{\text{value}}}$$



Main

```
Thread t1 =    t1.start();
Thread t2 =    t2.start();
sout("  ———  ")
```

Main thread doesnt wait for t1 & t2 to execute completely

JOIN : If we call join from a thread to another thread. it will wait until the next thread doesnt end.

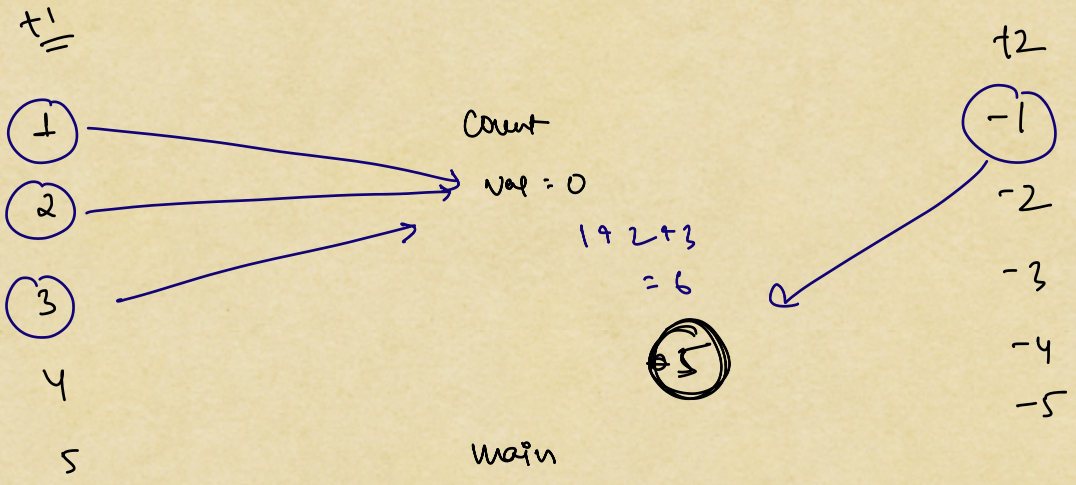
Main

```
Thread t1 =    t1.start();
Thread t2 =    t2.start();

t1.join();

sout(" ————");
```

main will wait until t1 completes.



main

t1

t2

$\rightarrow$ print(val) $\rightarrow$ $\underline{\underline{5}}$

multiple threads try to read and write on the same datapoint there is a high chance of ambiguous results.

$\boxed{500}$ → actual final value = 600

t1

debit 100

read initial balance = 500

debit = 500-100 = 400

save new balance = 400

t2

UPI => 200

read initial balance = 500

Credit = 500+200 = 700

save new balance = 700.

$\boxed{54}$

↓

lt

P1 → t1

read initial status = available

P2 → t2

read initial status = available

Friday	Sat	Sun

OOPS → 1, 2, 3, 4, Wit. final (6)

Threads ⇒ 1 | 2 | 3

task =>

- P0 * complete lecture => Threads 1/2/3
- P1 * complete assign => Thread 1/2
- P2 * watch lectures => OOPs
  (pending)

  + assignment