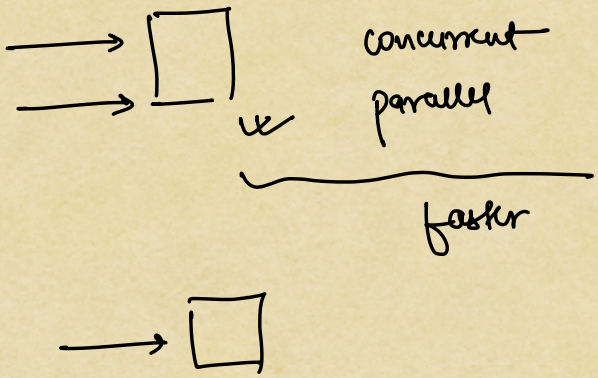
* Implementation of multi-threaded code
* Executors
* Callbacks
* Synchronization (Intro)

=> Multi-threaded code :-



↓

dividing the tasks into multiple threads

=> What task I want to achieve in parallel execution

- print "HelloWorld"

Step 1 → Create a class for the task

```
class HelloWorldPrinter {

}
```

Step 2 → Implement 'Runnable' interface

```
class HelloWorldPrinter implements Runnable{

    void run() {

    }

}
```

Step 3 → Write logic inside run() method

```
class HelloWorldPrinter implements Runnable{
    void run() {
        System.out("Hello World");
    }
}
```

Step 1, 2, 3 → setup

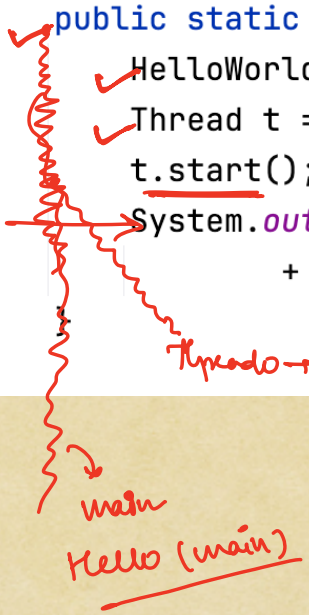
Step 4 → At execution code, create a thread object

```
HelloWorldPrinter hwp = new HelloWorldPrinter();
Thread t = new Thread(hwp);
```

Step 5 → Start thread

```
t.start();
```

```java
public class Main {
    public static void main(String[] args) {
        HelloWorldPrinter hwp = new HelloWorldPrinter();
        Thread t = new Thread(hwp);
        t.start();
        System.out.println("Hello World from thread : "
                + Thread.currentThread().getName());
    }
}
```



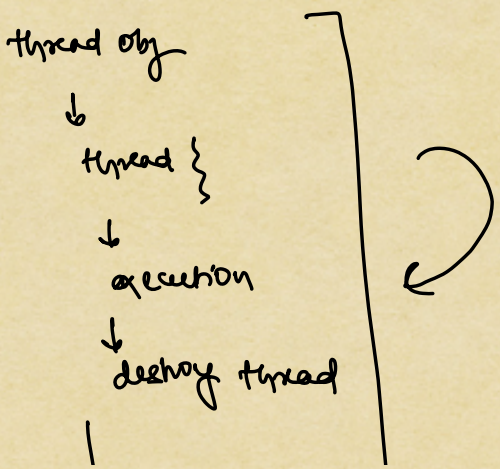
run()

- execute the run() method on top the current thread

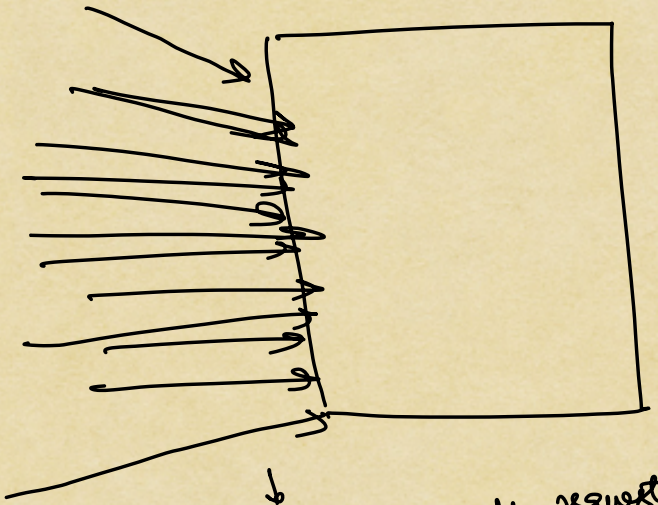
start()

- Create a new thread and,
- calls run() on top of that new thread

⇒ Executors and thread pool :-



destroy object



parallelly handle requests for every customer

* every time any request comes, we assign a thread to it

→ reuse the thread

↓

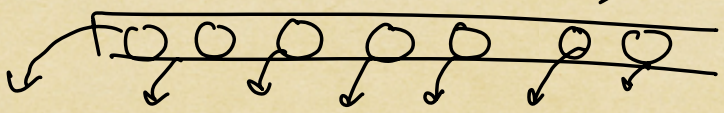
Spaces ⇒ reusable socket

↓

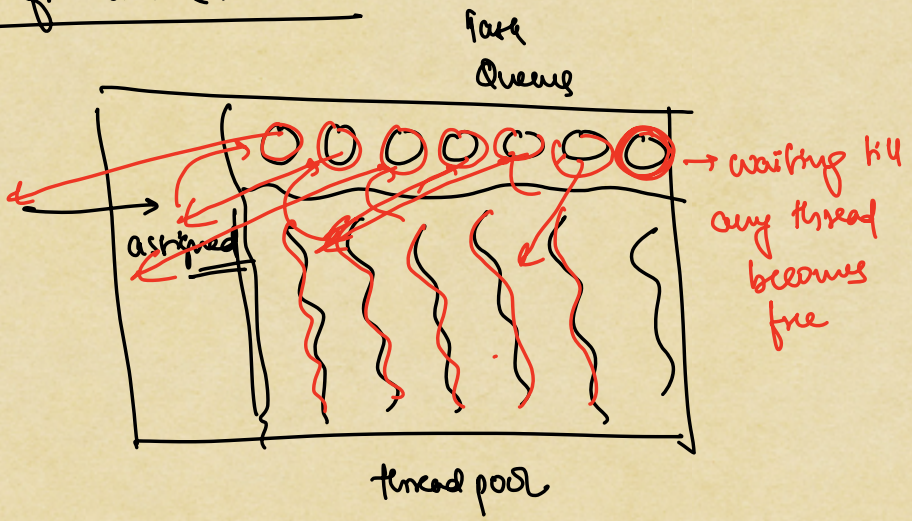
famous ⇒ getting a list of orders

↓

wait (FCFS)



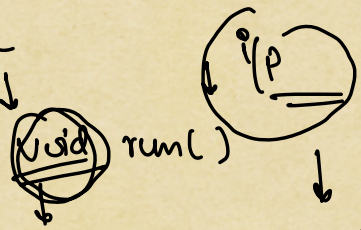
* Executor framework :-



* reusable threads

* Optimisation by removing the overhead of creation & destruction of threads

=> runnable



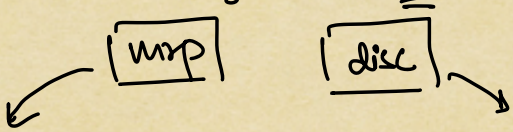
variables / attributes

↓

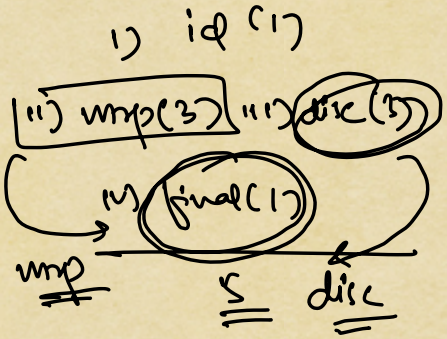
constructor

counter → shop => item => mrp disc rack

barcode => id

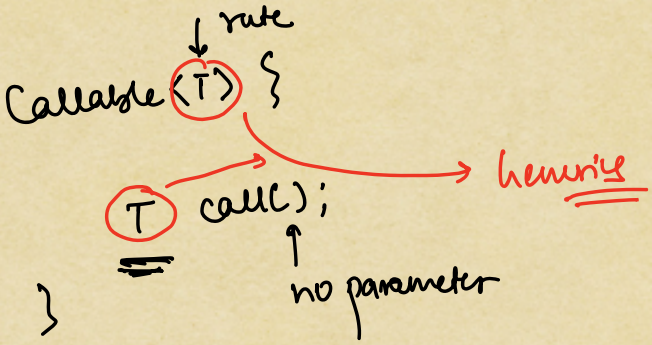


1) id (1)
2) wrp (3)
3) disc (3)
4) calculate final (1)

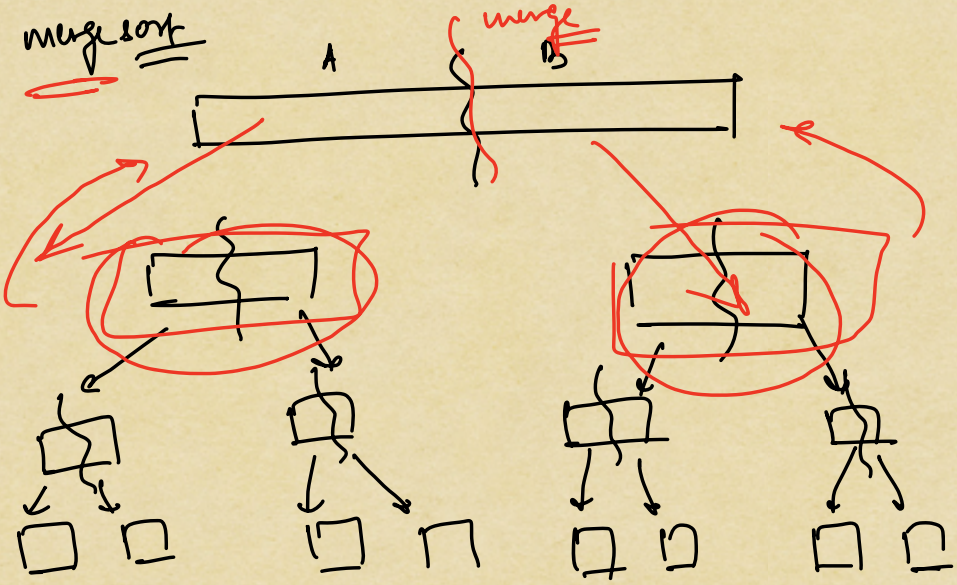


8

⇒ Runnable || Callable



H.W

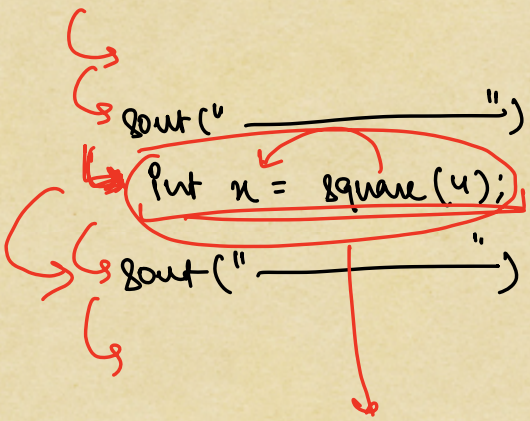


```java
public class Main {
    public static void main(String[] args) {
        /*...*/

        MRPCalculator mrpCalculator = new MRPCalculator( id: 10);
        DiscountCalculator discountCalculator = new DiscountCalculator( id: 10);
        ExecutorService executor = Executors.newFixedThreadPool( nThreads: 2);
        Integer mrp = executor.submit(mrpCalculator);
        Double discount = executor.submit(discountCalculator);

        double finalPrice = mrp - (mrp * discount/100);
        System.out.println("Final price : " + finalPrice);
    }
}
```

main (future)

execute the call() mrpCalc.

mrp value



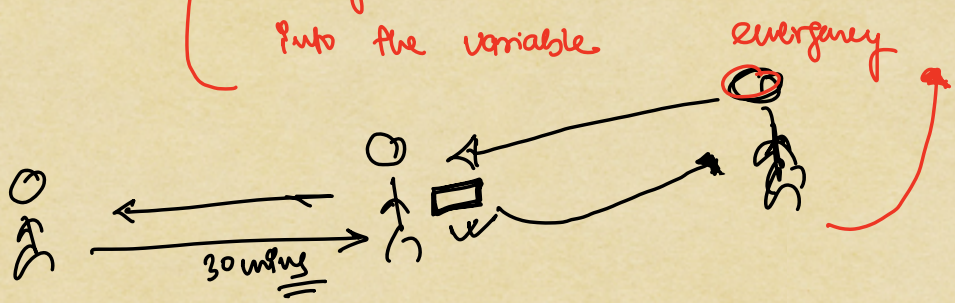
i) Square

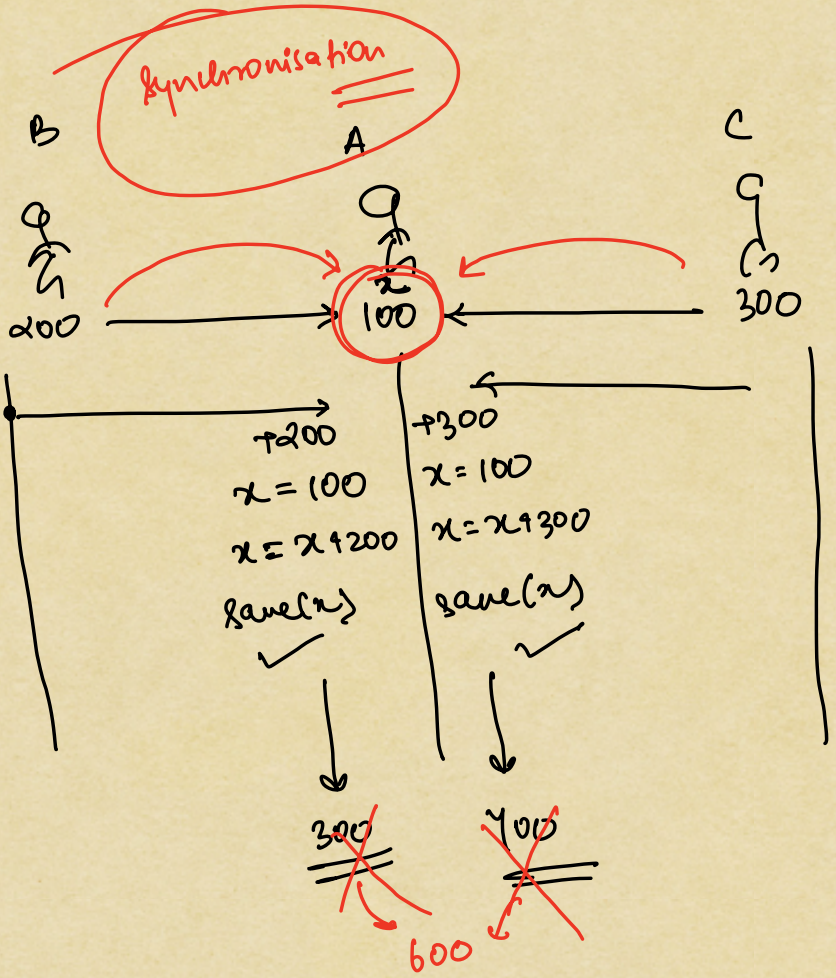
ii) return

iii) assign to x

i) Calculation of result

ii) assign the result into the variable




Synchronisation
B
A
C
200
x
100
300
+200
x = 100
x = x + 200
save(x)
+300
x = 100
x = x + 300
save(x)
300
400
600