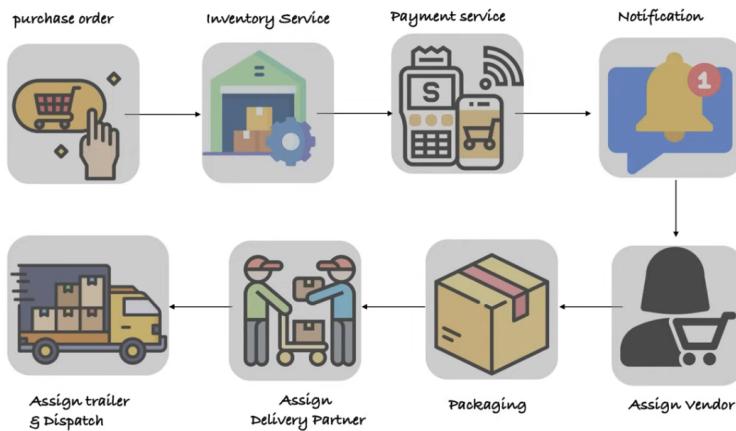


@Async Annotation

Fulfillment service Each one is micro service



```
/*
1. Inventory service check order availability
2. Process payment for order
3. Notify to the user
4. Assign to vendor
5. packaging
6. assign delivery partner
7. assign trailer
8. dispatch product
*/
```

The above one is the fulfillment process. Let us check how we can implement async here.

A screenshot of an IDE (IntelliJ IDEA) showing a Java code editor. The code is part of an OrderFulfillmentController class:

```
19
20 @PostMapping
21 public ResponseEntity<Order> processOrder(@RequestBody Order order) t
22     service.processOrder(order);
23     service.notifyUser(order);
24     service.assignVendor(order);
25     service.packaging(order);
26     service.assignDeliveryPartner(order);
27     service.assignTrailerAndDispatch(order);
28     return ResponseEntity.ok(order);
29 }
```

The code implements a POST endpoint for processing an order. It calls several methods on a service object: processOrder, notifyUser, assignVendor, packaging, assignDeliveryPartner, and assignTrailerAndDispatch. The notifyUser method is highlighted with a yellow background, indicating it is the target for asynchronous implementation.

Screenshot of IntelliJ IDEA showing the code for `OrderFulfillmentService.java`. The code implements the `processOrder` method:

```
    7. dispatch product
    /**
     * ...
     */
    public Order processOrder(Order order) throws InterruptedException {
        order.setTrackingId(UUID.randomUUID().toString());
        if (inventoryService.checkProductAvailability(order.getProductId())) {
            //handle exception here
            paymentService.processPayment(order);
        } else {
            throw new RuntimeException("Technical issue please retry");
        }
        return order;
    }
```

Screenshot of IntelliJ IDEA showing the code for `OrderFulfillmentService.java`. A callout bubble with the text "SIMULATE TIME-CONSUMING TASK" points to the `Thread.sleep` calls in the `notifyUser`, `assignVendor`, and `packaging` methods.

```
public void notifyUser(Order order) throws InterruptedException {
    Thread.sleep(millis: 4000L);
    log.info("Notified to the user " + Thread.currentThread().getName());
}

public void assignVendor(Order order) throws InterruptedException {
    Thread.sleep(millis: 5000L);
    log.info("Assign order to vendor " + Thread.currentThread().getName());
}

public void packaging(Order order) throws InterruptedException {
    Thread.sleep(millis: 2000L);
    log.info("Order packaging completed " + Thread.currentThread().getName());
}
```

Screenshot of IntelliJ IDEA showing the code for `OrderFulfillmentService.java`. The `assignDeliveryPartner` and `assignTrailerAndDispatch` methods both contain `Thread.sleep` calls with different millisecond values.

```
public void assignDeliveryPartner(Order order) throws InterruptedException {
    Thread.sleep(millis: 10000L);
    log.info("Delivery partner assigned " + Thread.currentThread().getName());
}

public void assignTrailerAndDispatch(Order order) throws InterruptedException {
    Thread.sleep(millis: 3000L);
    log.info("Trailer assigned and Order dispatched " + Thread.currentThread().getName());
}
```

```
7  
8     @Service  
9     @Slf4j  
10    public class InventoryService {  
11  
12        public boolean checkProductAvailability(int productId) {  
13            return true;  
14        }  
15    }  
16
```

```
6  
7     @Service  
8     @Slf4j  
9     public class PaymentService {  
10  
11        @  
12        public void processPayment(Order order) throws InterruptedException {  
13            log.info("initiate payment for order " + order.getProductId());  
14            //call actual payment gateway  
15            Thread.sleep( millis: 2000L );  
16            log.info("completed payment for order " + order.getProductId());  
17        }  
18    }
```

The screenshot shows a dual-pane developer interface. On the left, the Postman application is open, displaying a POST request to `http://localhost:9191/orders`. The request body is a JSON object:

```
1  {  
2      "productId": 1436,  
3      "name": "Mobile",  
4      "productType": "Electronics",  
5      "qty": 1,  
6      "price": 10000  
7  }
```

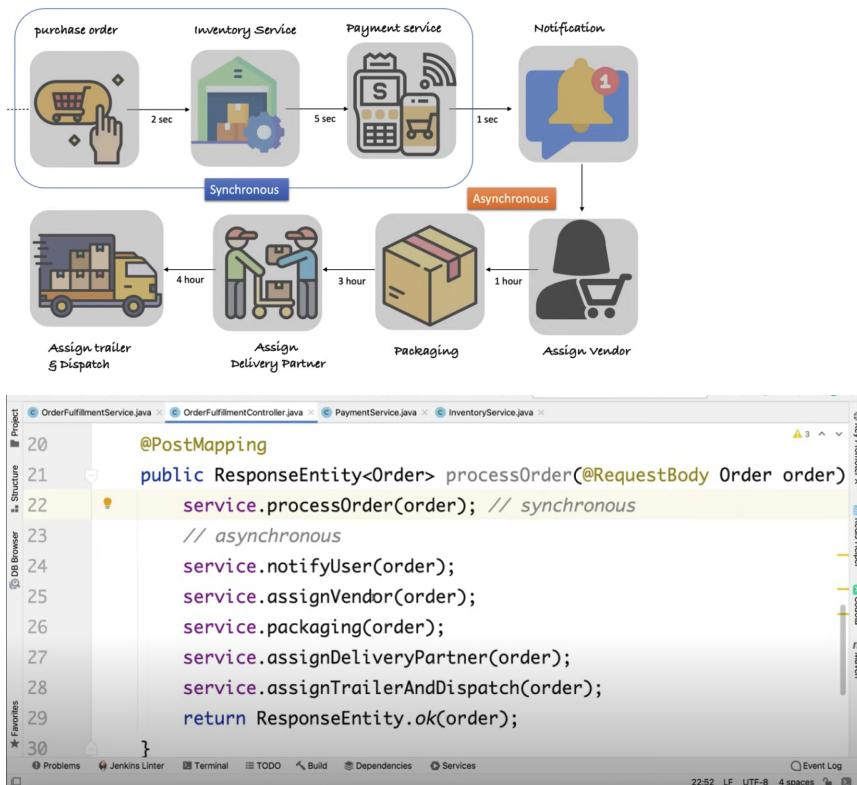
On the right, the IntelliJ IDEA IDE is open, showing the PurchaseOrderServiceApplication project. The `InventoryService.java` file is selected, containing the following code:

```
1  lhost].[]/ : Initializing Spring DispatcherServlet 'dispatcherServlet'  
2  erServlet : Initializing Servlet 'dispatcherServlet'  
3  erServlet : Completed initialization in 1 ms  
4  aymentService : initiate payment for order 1436  
5  aymentService : completed payment for order 1436  
6  entService : Notified to the user http-nio-9191-exec-2  
7  entService : Assign order to vendor http-nio-9191-exec-2  
8  entService : Order packaging completed http-nio-9191-exec-2
```

It took 26 seconds to get response.

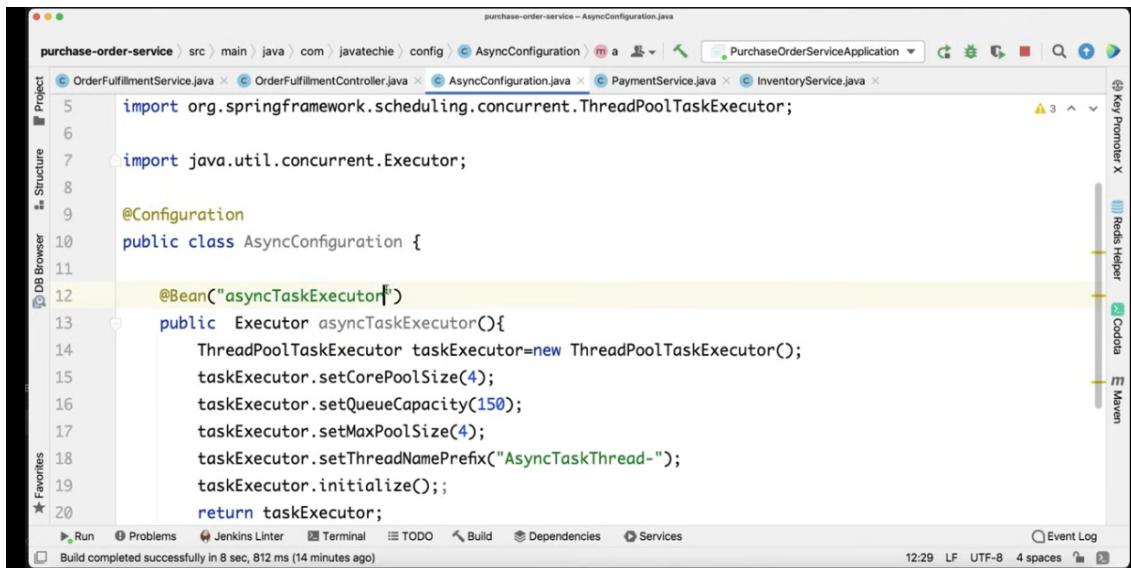
The screenshot shows a Java development environment with a code editor and a terminal. In the code editor, there is a Java file named `PurchaseOrderServiceApplication.java` with some code related to order processing. In the terminal, a log file is displayed, showing the execution of the application and various service interactions. A separate window shows a POST request in Postman to `http://localhost:9191/orders` with a JSON payload containing a product ID, name, type, quantity, and price. The response status is 200 OK with a total time of 26.49 seconds.

Now change it to some part async as shown below.



Create below configuration Java file. The first 3 setters are important.

- 1 first setter keeps active threads even in idle case
2. Second setter keeps maximum task in a queue
3. Third setter - how many overall threads we can have.



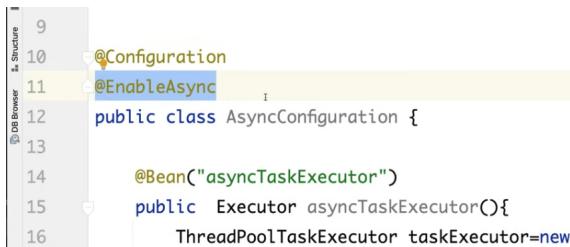
The screenshot shows the IntelliJ IDEA interface with the file `AsyncConfiguration.java` open. The code defines a class `AsyncConfiguration` with a single method `asyncTaskExecutor()`. This method creates a `ThreadPoolTaskExecutor` with a core pool size of 4, a queue capacity of 150, and a maximum pool size of 4. It also sets a thread name prefix of "AsyncTaskThread-".

```
import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;
import java.util.concurrent.Executor;

@Configuration
public class AsyncConfiguration {

    @Bean("asyncTaskExecutor")
    public Executor asyncTaskExecutor(){
        ThreadPoolTaskExecutor taskExecutor=new ThreadPoolTaskExecutor();
        taskExecutor.setCorePoolSize(4);
        taskExecutor.setQueueCapacity(150);
        taskExecutor.setMaxPoolSize(4);
        taskExecutor.setThreadNamePrefix("AsyncTaskThread-");
        taskExecutor.initialize();
        return taskExecutor;
    }
}
```

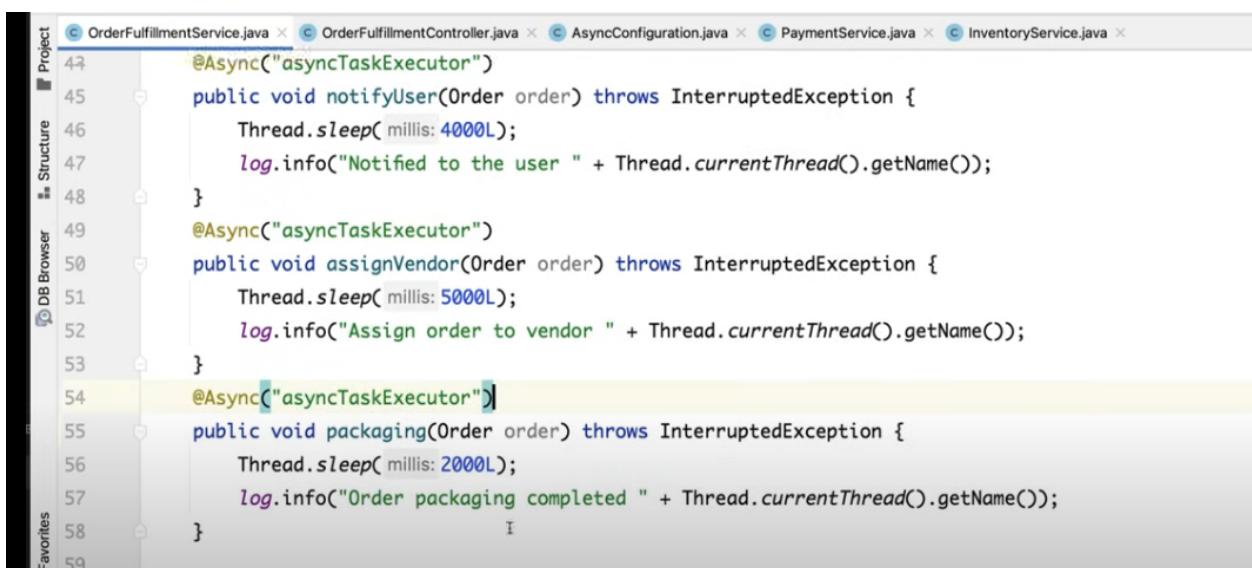
Add `@EnableAsync` to this class => what are all `@Async` will execute by this config



The screenshot shows the same `AsyncConfiguration.java` file, but now it includes the `@EnableAsync` annotation on the class definition. This annotation enables the execution of methods marked with `@Async`.

```
@Configuration
@EnableAsync
public class AsyncConfiguration {

    @Bean("asyncTaskExecutor")
    public Executor asyncTaskExecutor(){
        ThreadPoolTaskExecutor taskExecutor=new
```

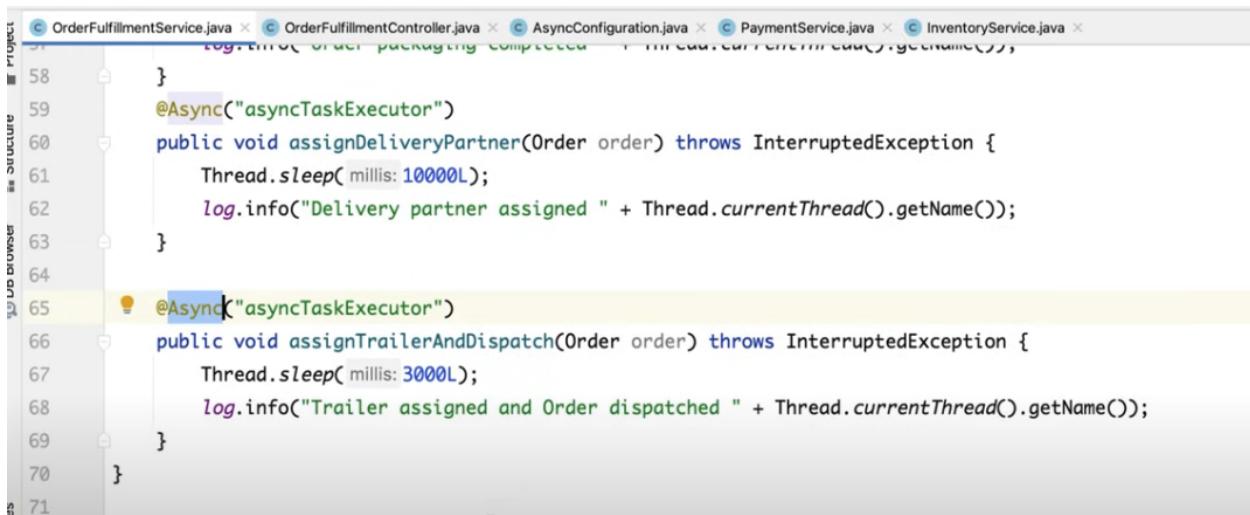


The screenshot shows the `OrderFulfillmentService.java` file containing three methods annotated with `@Async`: `notifyUser`, `assignVendor`, and `packaging`. Each method performs a short sleep operation and logs a message indicating the thread name.

```
@Async("asyncTaskExecutor")
public void notifyUser(Order order) throws InterruptedException {
    Thread.sleep( millis: 4000L );
    log.info("Notified to the user " + Thread.currentThread().getName());
}

@Async("asyncTaskExecutor")
public void assignVendor(Order order) throws InterruptedException {
    Thread.sleep( millis: 5000L );
    log.info("Assign order to vendor " + Thread.currentThread().getName());
}

@Async("asyncTaskExecutor")
public void packaging(Order order) throws InterruptedException {
    Thread.sleep( millis: 2000L );
    log.info("Order packaging completed " + Thread.currentThread().getName());
}
```



```
58     }
59     @Async("asyncTaskExecutor")
60     public void assignDeliveryPartner(Order order) throws InterruptedException {
61         Thread.sleep( millis: 1000L );
62         log.info("Delivery partner assigned " + Thread.currentThread().getName());
63     }
64
65     @Async("asyncTaskExecutor")
66     public void assignTrailerAndDispatch(Order order) throws InterruptedException {
67         Thread.sleep( millis: 3000L );
68         log.info("Trailer assigned and Order dispatched " + Thread.currentThread().getName());
69     }
70 }
71
```

For `@Async` tag method return type either void or `CompletableFuture`.

This takes only 2.4 seconds. Till processOrder alone sync rest are all async.

