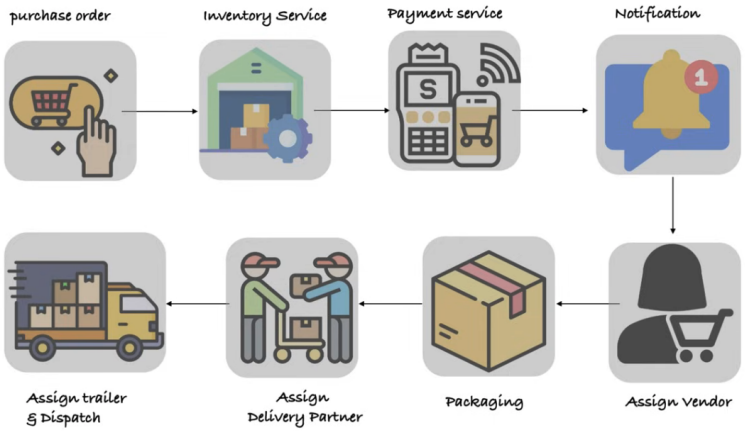
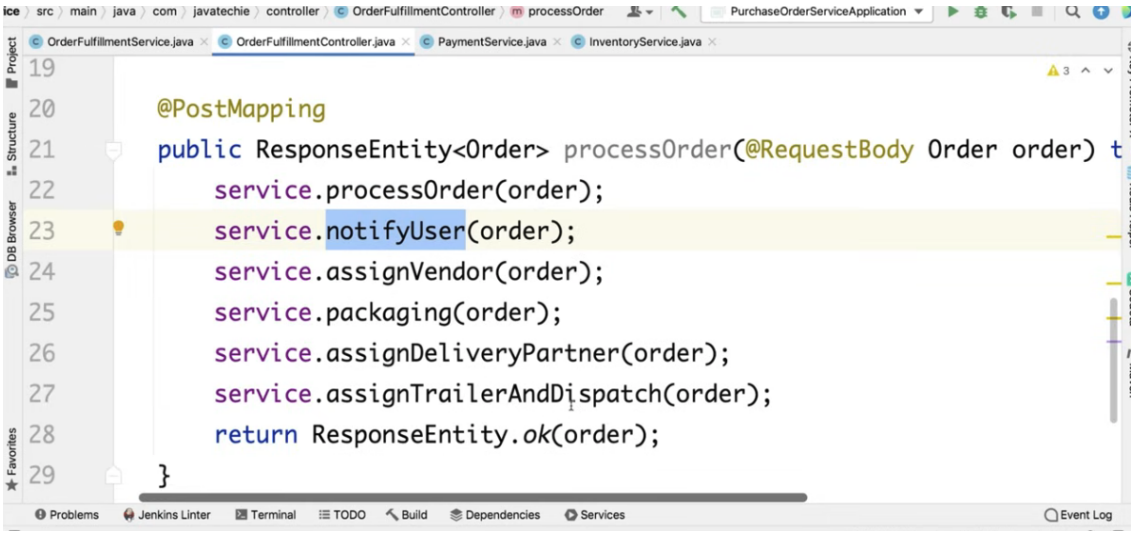
@Async Annotation

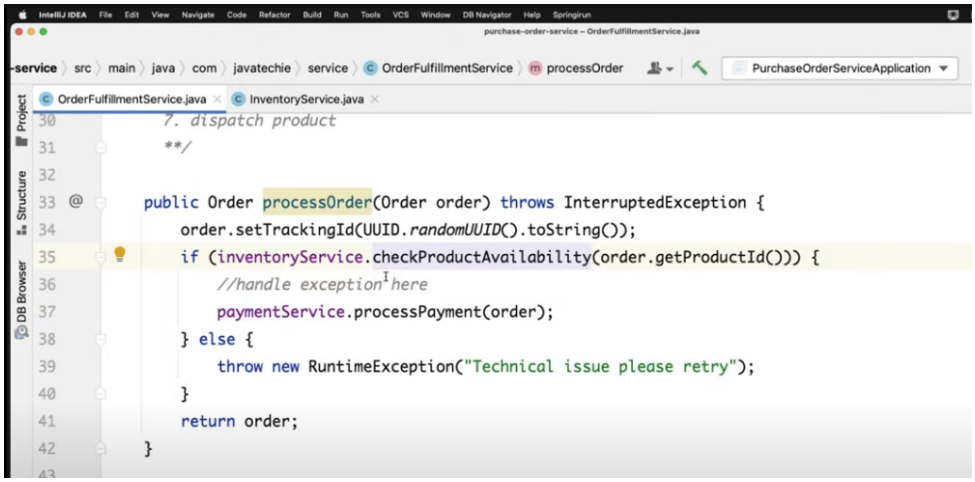
Fulfillment service Each one is micro service

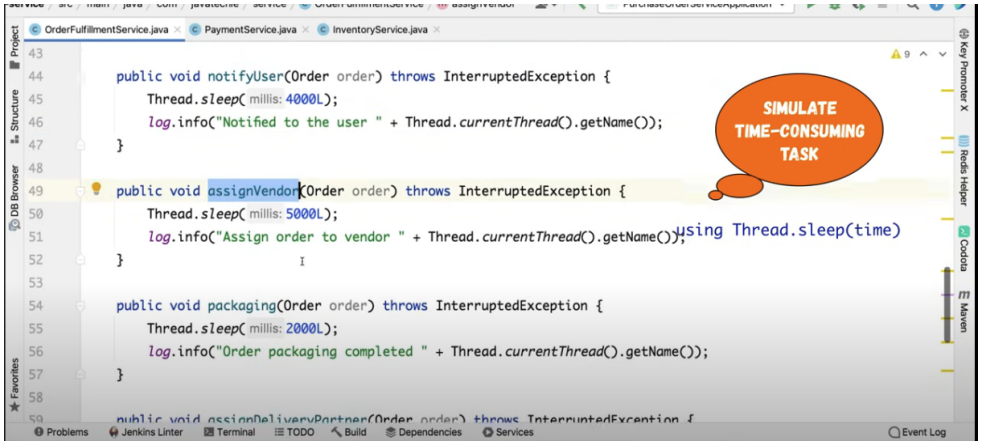


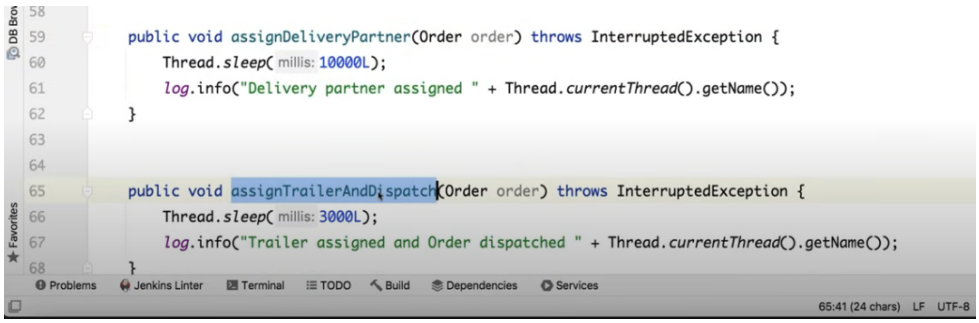

```
/*
  1. Inventory service check order availability
  2. Process payment for order
  3. Notify to the user
  3. Assign to vendor
  4. packaging
  5. assign delivery partner
  6. assign trailer
  7. dispatch product
  **/
```

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

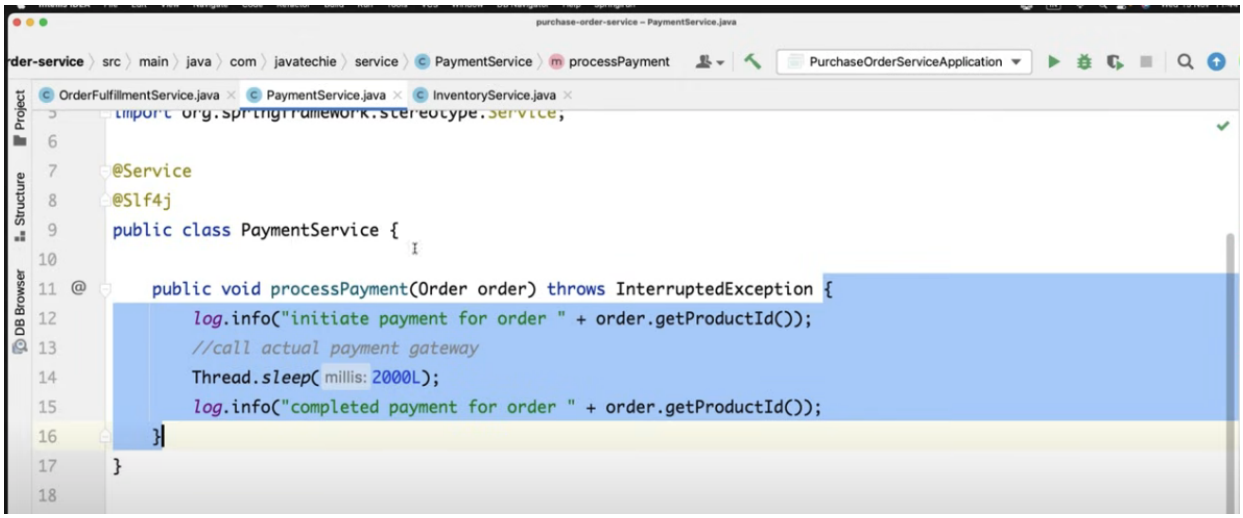


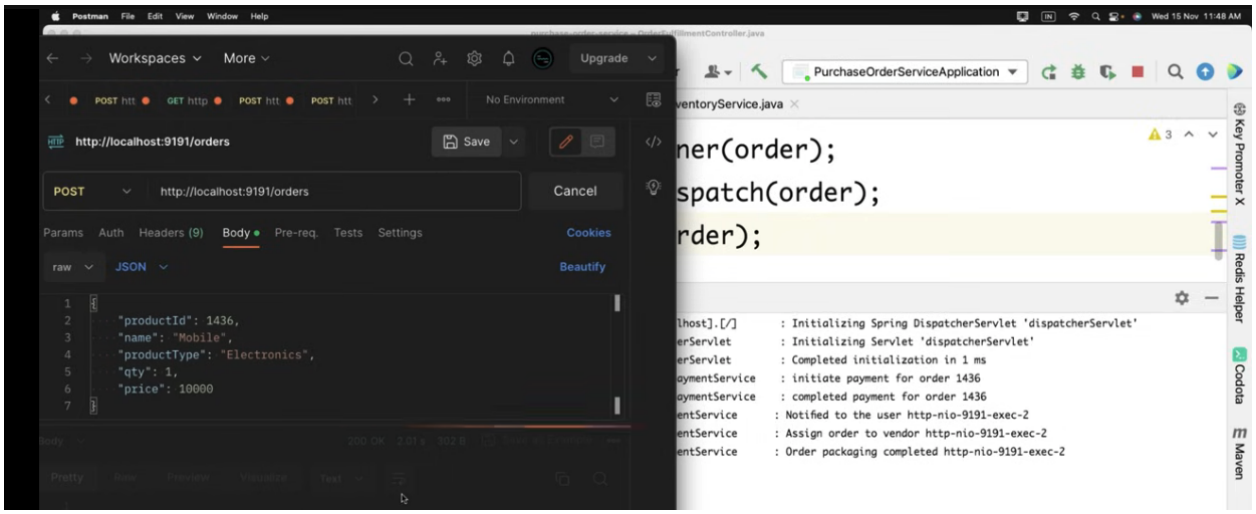

OrderFulfillmentService.java
InventoryService.java
7. dispatch product
**/
public Order processOrder(Order order) throws InterruptedException {
order.setTrackingId(UUID.randomUUID().toString());
if (inventoryService.checkProductAvailability(order.getProductId())) {
//handle exception here
paymentService.processPayment(order);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


OrderFulfillmentService.java
PaymentService.java
InventoryService.java
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());
}
SIMULATE TIME-CONSUMING TASK
using Thread.sleep(time)

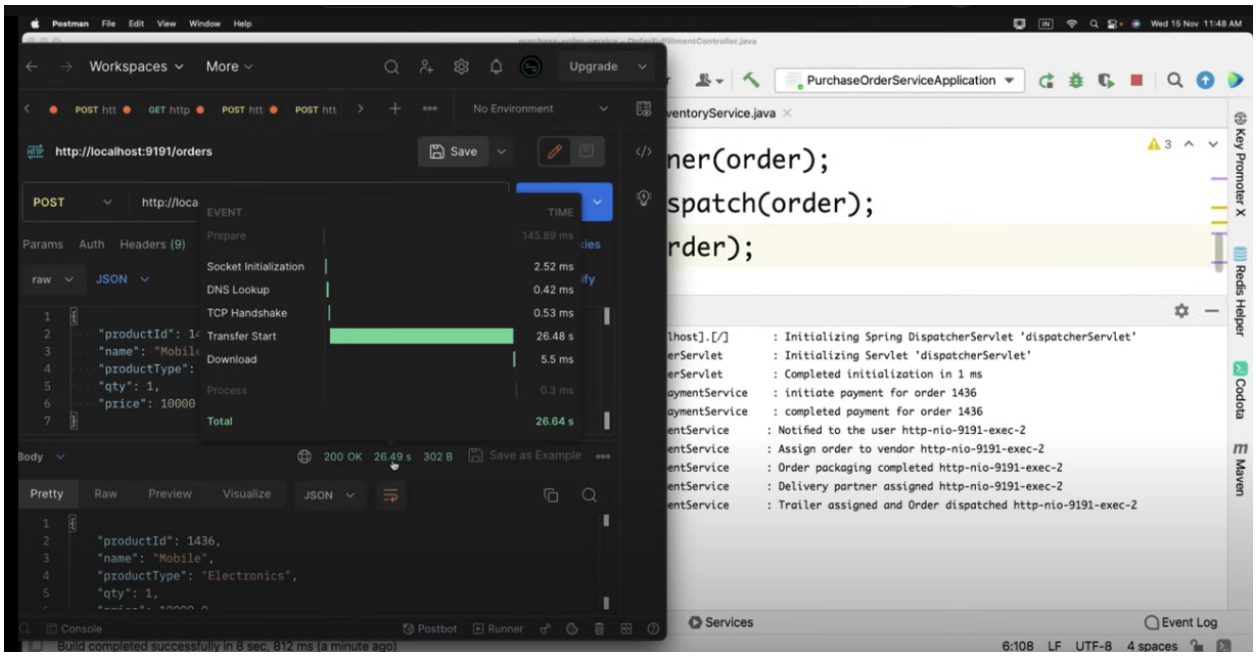

public void assignDeliveryPartner(Order order) throws InterruptedException {
Thread.sleep(millis: 10000L);
log.info("Delivery partner assigned " + Thread.currentThread().getName());
}
public void assignTrailerAndDispatch(Order order) throws InterruptedException {
Thread.sleep(millis: 3000L);
log.info("Trailer assigned and Order dispatched " + Thread.currentThread().getName());
}


OrderFulfillmentService.java
InventoryService.java
@Service
@Slf4j
public class InventoryService {
public boolean checkProductAvailability(int productId) {
return true;
}
}

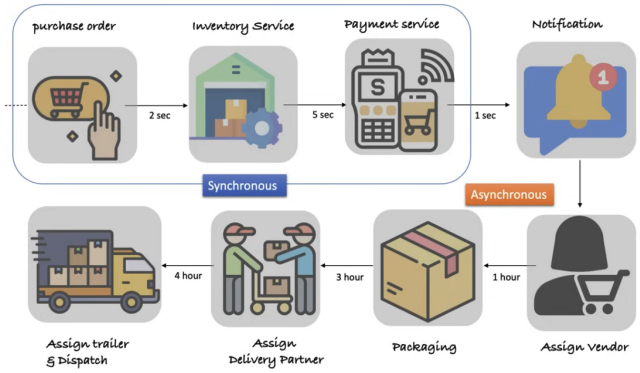

purchase-order-service – PaymentService.java
PurchaseOrderServiceApplication
OrderFulfillmentService.java
PaymentService.java
InventoryService.java
@Service
@Slf4j
public class PaymentService {
public void processPayment(Order order) throws InterruptedException {
log.info("initiate payment for order " + order.getProductId());
//call actual payment gateway
Thread.sleep(millis: 2000L);
log.info("completed payment for order " + order.getProductId());
}
}

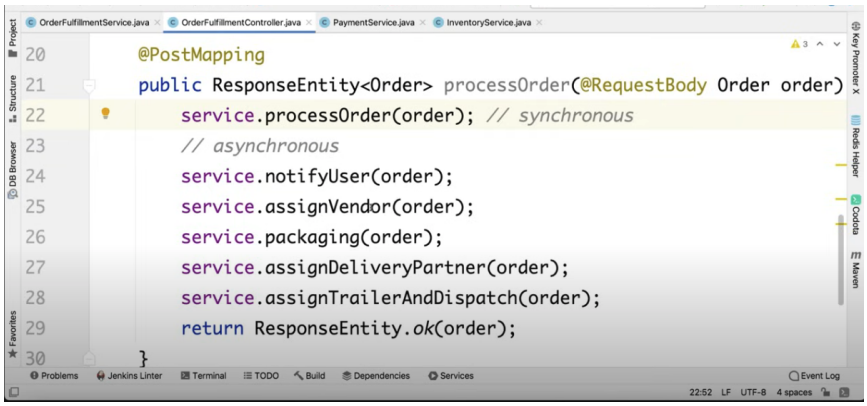

Postman
Workspaces
More
Upgrade
http://localhost:9191/orders
Save
POST
http://localhost:9191/orders
Cancel
Params
Auth
Headers (9)
Body
Pre-req.
Tests
Settings
Cookies
raw
JSON
Beautify
{
"productId": 1436,
"name": "Mobile",
"productType": "Electronics",
"qty": 1,
"price": 10000
}
PurchaseOrderServiceApplication
ner(order);
spatch(order);
rder);
: Initializing Spring DispatcherServlet 'dispatcherServlet'
: Initializing Servlet 'dispatcherServlet'
: Completed initialization in 1 ms
: initiate payment for order 1436
: completed payment for order 1436
: Notified to the user http-nio-9191-exec-2
: Assign order to vendor http-nio-9191-exec-2
: Order packaging completed http-nio-9191-exec-2
Key Promoter X
Redis Helper
Codota
Maven

It took 26 seconds to get response.

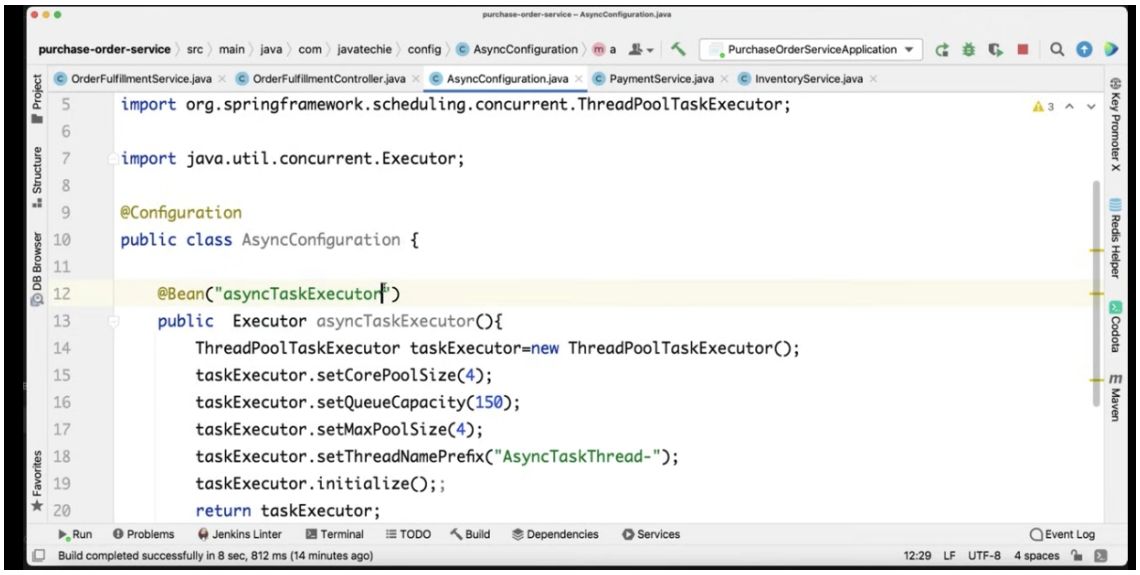


Now change it to some part async as shown below.





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.



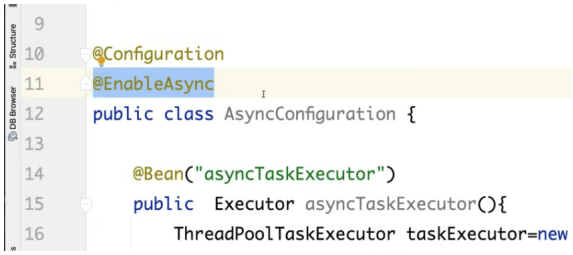
```java
import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;

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;
```

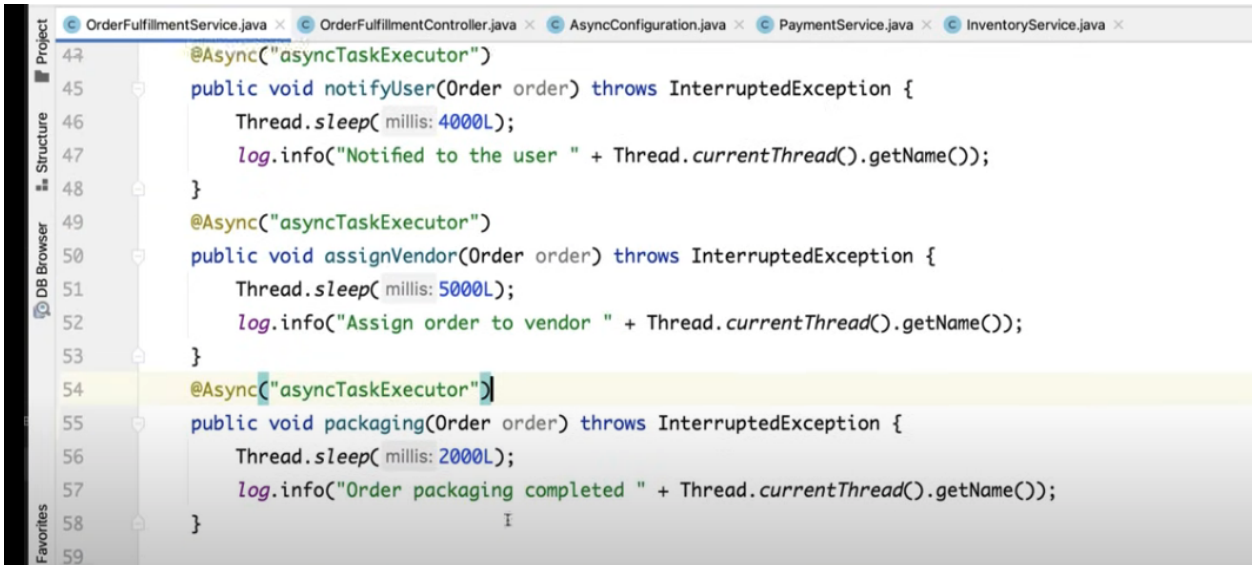
Add @EnableAsync to this class => what are are all @Async will execute by this config



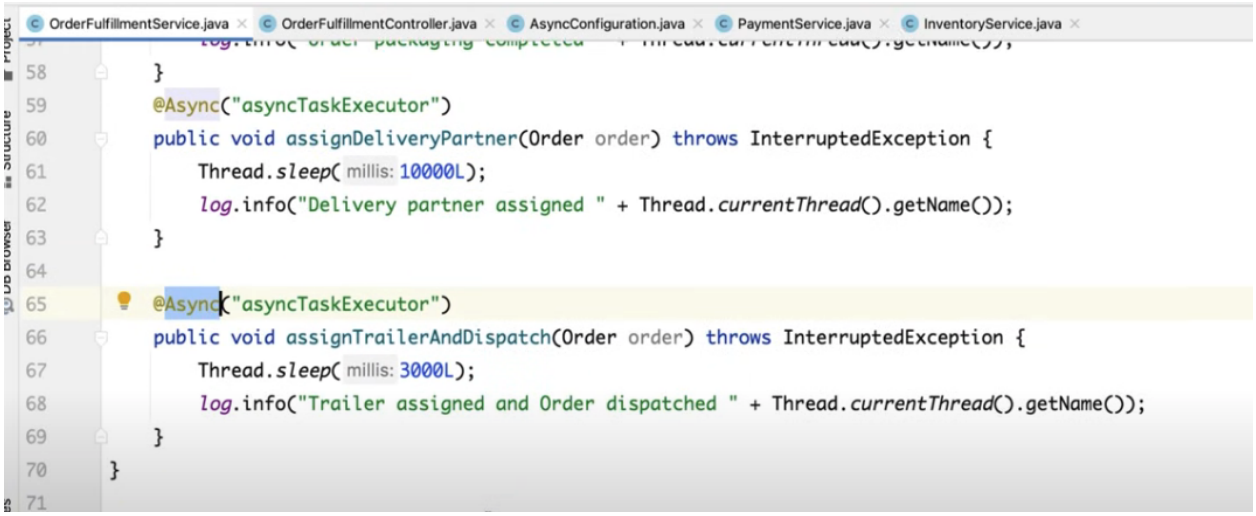
```java

@Configuration
@EnableAsync
public class AsyncConfiguration {

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



```java
    @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());
    }
```



For @Async tag method return type either void or CompletableFuture.
This takes only 2.4 seconds. Till processOrder alone sync rest are all async.