

How to remember Structural Design patterns:

BD🡪Budedu , PFAC-🡪PF acoount

BD 🡪(Behaviroul, Decorator)Budedu, PFFAC🡪 (Proxy, Flyweight, Façade,Adapter,Composite)PF account

**Saga Orchestration Pattern**

**🎯 What is it?**

A **central coordinator** (orchestrator) manages the entire saga by telling each service **what to do next**.

Example

**OrchestratorService is the orchestrator ✅**

java

CopyEdit

@Service

public class OrchestratorService {

@Autowired OrderService orderService;

@Autowired PaymentService paymentService;

@Autowired InventoryService inventoryService;

public String placeOrder(OrderRequest request) {

boolean orderCreated = orderService.createOrder(request);

if (!orderCreated) return "Order Failed";

boolean paid = paymentService.deductPayment(request);

if (!paid) {

orderService.cancelOrder(request);

return "Payment Failed";

}

boolean stock = inventoryService.reduceStock(request);

if (!stock) {

paymentService.refund(request);

orderService.cancelOrder(request);

return "Inventory Failed";

}

return "Order Successful!";

}

}

## 💃 Choreography Pattern

### 🎯 What is it?

There is **no central coordinator** — each service **reacts to events** from the previous service and decides what to do next.

## 🧩 Scenario: Order Placement Flow (Choreography Style)

**Services:**

1. OrderService → emits OrderCreatedEvent
2. PaymentService → listens to OrderCreatedEvent, emits PaymentCompletedEvent
3. InventoryService → listens to PaymentCompletedEvent, emits InventoryUpdatedEvent
4. ShippingService → listens to InventoryUpdatedEvent

## Shared Model (Used across services)

java

CopyEdit

// OrderEvent.java

public class OrderEvent {

private String orderId;

private String status; // CREATED, PAID, FAILED, etc.

private String source; // ORDER, PAYMENT, INVENTORY

// + Getters/Setters

}

## ✅ 1. Order Service

java

CopyEdit

@RestController

@RequestMapping("/order")

public class OrderController {

@Autowired KafkaTemplate<String, OrderEvent> kafkaTemplate;

@PostMapping

public ResponseEntity<String> placeOrder(@RequestBody OrderEvent event) {

event.setStatus("CREATED");

event.setSource("ORDER");

kafkaTemplate.send("order-events", event);

return ResponseEntity.ok("Order created and event published.");

}

}

## ✅ 2. Payment Service

java

CopyEdit

@Component

public class PaymentListener {

@Autowired KafkaTemplate<String, OrderEvent> kafkaTemplate;

@KafkaListener(topics = "order-events", groupId = "payment-group")

public void listen(OrderEvent event) {

if ("CREATED".equals(event.getStatus())) {

System.out.println("Processing payment for order: " + event.getOrderId());

// Simulate payment success

event.setStatus("PAID");

event.setSource("PAYMENT");

kafkaTemplate.send("order-events", event);

}

}

}

## ✅ 3. Inventory Service

java

CopyEdit

@Component

public class InventoryListener {

@Autowired KafkaTemplate<String, OrderEvent> kafkaTemplate;

@KafkaListener(topics = "order-events", groupId = "inventory-group")

public void listen(OrderEvent event) {

if ("PAID".equals(event.getStatus())) {

System.out.println("Reducing stock for order: " + event.getOrderId());

// Simulate stock update

event.setStatus("INVENTORY\_UPDATED");

event.setSource("INVENTORY");

kafkaTemplate.send("order-events", event);

}

}

}

## ✅ 4. Shipping Service

java

CopyEdit

@Component

public class ShippingListener {

@KafkaListener(topics = "order-events", groupId = "shipping-group")

public void listen(OrderEvent event) {

if ("INVENTORY\_UPDATED".equals(event.getStatus())) {

System.out.println("Shipping order: " + event.getOrderId());

// Final service, no new event needed

}

}

}