1. **ClickpostShipmentsConsumer**

**onMessage(final ConsumerRecord<String, String> consumerRecord): This method processes the Kafka messages.**

* **consumerRecord.value(): Extracts the message from the Kafka record.**
* **Logging: It logs the reception of the message.**
* **if (StringUtils.isBlank(message)): Skips processing if the message is blank.**

### **Shipment Message Parsing and Validation**

* **Shipment Data Object (ShipmentDO): The message is parsed into a ShipmentDO object (presumably representing a shipment) using JsonXmlUtil.convertJsonToObject().**
* **Validation: It checks for invalid or missing fields, including tracking\_id, carrier, account\_id, or whether the shipment is already registered for tracking.**
  + **If any of these checks fail, the method logs a warning and skips further processing.**

### **Carrier Lookup and Registration**

* **Carrier Lookup: The carrier name from the shipment is used to find a corresponding carrier object (CarrierDO) from the database (carrierDao.findByNameOrAltName()).**
  + **The carrier's raw name is used to determine if the shipment should be processed by Clickpost.**
* **Clickpost Shipment Registration: If the carrier matches "clickpost" and shipment registration is enabled (config.isClickpostShipmentRegistrationEnabled()), the method attempts to register the shipment by invoking a web service.**

### **Web Service Call and Shipment Registration**

* **populateShipRequestAndInvokeWs(): This helper method constructs a ShipmentRegistrationRequest with the shipment details (such as tracking\_id, carrier, etc.) and makes a web service call using clickpostWsClient.invokeCarrierWebservice().**
  + **Carrier Partner ID: It retrieves the carrier's partner ID using clickPostUtil.getCarrierId().**
  + **Shipment Registration: If the carrier is enabled and all necessary data is present, it attempts to register the shipment with Clickpost.**

### **Saving and Publishing Events**

* **Database Save: If the shipment is successfully registered, it updates the ShipmentDO object (shipmentDO.setRegisteredForTracking(true)) and saves it to the database using shipmentDao.save().**
* **Event Publishing: It publishes the shipment event to a change feed topic (shipmentEventPublisher.accept(shipmentDO)), indicating the successful registration.**

### **Error Handling**

* **JSON Parsing Errors: If an exception occurs during JSON parsing (JsonXmlParsingException), it logs the error and continues processing other messages without crashing the consumer.**

1. **CAFmsShipmentEventKafkaMessageProcessor**

### **Message Processing Method: onMessage()**

* **onMessage(final ConsumerRecord<String, String> consumerRecord): This method is triggered when a new Kafka message is received.**
  + **Extract Message: String message = consumerRecord.value(); retrieves the payload (JSON string) from the Kafka message.**
  + **Parse Message: The message is parsed into a FMSShipmentWrapper object using JsonXmlUtil.convertJsonToObject(). This wrapper likely contains both the event header and the payload.**
  + **Retrieve Event Name: The event name is extracted from the wrapper's header using shipmentWrapper.getHeader().getEventName(). If either the wrapper or the header is null, the event name will also be null.**

### **Event Validation**

* **Check Validity: The isValidEvent() method is called to check whether the event is of a valid type (i.e., "PO\_SHIPMENT", "ORDER\_SHIPMENT", or "SO\_SHIPMENT").**
  + **Predicate: The method uses a Predicate<String> that tests whether the event name matches one of these types.**

### **Processing Valid Events**

* **Log and Process:**
  + **Logging: If the event is valid, the method logs the message and the event name using logMessage(). This method decides how much detail to log based on the configuration setting (appConfig.isCaFMSShippedEventLogsEnable()).**
  + **Delegation: The message is then processed by the CAShipmentEventHandler via handleFMSShipmentPayload(), which performs the actual shipment-related logic.**
  + **Success Log: If processing is successful, it logs a success message (orderId processed successfully).**

### **Invalid Event Handling**

* **Invalid Events: If the event is not valid or the shipment wrapper is null, the processor logs that the event is "not eligible for processing."**

### **Helper Methods**

* **logMessage(String message, String eventName): This method logs the received message. If detailed logs are enabled via the AppConfig, it logs the event name and the entire message; otherwise, it only logs the event name.**
* **isValidEvent(): This method returns a Predicate to validate event names. The event is considered valid if it is either "PO\_SHIPMENT", "ORDER\_SHIPMENT", or "SO\_SHIPMENT".**

### **Summary:**

* **Message Consumption: The CAFmsShipmentEventKafkaMessageProcessor listens for Kafka messages and processes them.**
* **Event Validation: It checks if the event is of a valid type (purchase order shipment, order shipment, or sales order shipment).**
* **Delegation: If valid, the shipment payload is processed by the CAShipmentEventHandler, which handles business logic like enriching, saving, and publishing shipment data.**
* **Logging: It logs event information, with detailed logging configurable based on the application settings.**
* **Efficiency: The code is streamlined to only process relevant events and avoids unnecessary processing of invalid or unrecognized event types.**

1. **ShipmentKafkaMessageProcessor**

### **Message Processing Method: onMessage()**

* **onMessage(final ConsumerRecord<String, String> consumerRecord): This is the method called whenever a new Kafka message is received. Here's the breakdown of its steps:**

#### **Logging Configuration**

* **Check log configuration: The method first checks if logging for shipment processing is enabled using a configuration flag (shipmentProcessor\_logs\_enabled), and if so, logs the received message.**

#### **Parsing and Mapping the Message**

* **Convert JSON to Object: The received Kafka message (a JSON string) is converted into an EventWrapper<Shipment> object using JsonXmlUtil.convertJsonToObject(). This wrapper likely contains the shipment payload and event metadata.**
* **Extract Account ID: The account ID is retrieved from the event header. If it's available, it's used to fetch the corresponding Account object from the database (accountDao.findByPrimaryId(accountId)), and it is then mapped to the Account class.**
* **Extract Shipment Details: The tracking ID, order number, and carrier are extracted from the shipment payload, and these values are logged for debugging purposes.**

#### **Validation**

* **Account Validation: The method checks if the Account object is not null using Validate.notNull(). If the account is null, the message processing will fail with an error indicating the account configuration wasn't found.**

#### **Logging Kafka Headers**

* **Extract Kafka Headers: The method uses getHeaderMap(consumerRecord) to extract headers from the Kafka message. If Kafka header logging is enabled (appConfig.isKafkaHeaderLoggingEnabled()), the headers are logged.**
* **Set Tenant ID: If the header map contains a TENANT\_ID, this value is assigned to the tenant ID in the shipment payload.**

#### **Shipment Handling**

* **Delegate to Shipment Handler: The shipmentHandlerImpl.handle() method is called, passing the Account, the shipment payload, and the ShipmentEventKafkaPublisher. This method performs the core business logic for processing the shipment.**
* **Log Success: Once the shipment event is processed successfully, a success log is generated, noting the tracking ID, order number, and carrier.**

#### **Shipment Validation**

* **Validate Shipment: After handling, the validateShipment.validateShipment() method is called to validate the shipment event. It ensures that the shipment data and related account configuration are correct.**
* **Log Validation Success: If validation is successful, a log message indicates that the shipment event was validated.**

### **Helper Methods**

* **getHeaderMap(ConsumerRecord<String, String> consumerRecord): This method (not shown in the code snippet) likely extracts Kafka headers and converts them into a map format for logging or further processing.**

### **Summary:**

1. **Message Consumption: It consumes the shipment event message from the Kafka topic.**
2. **Message Logging: Optionally logs the raw message and Kafka headers based on configuration.**
3. **Message Parsing: Converts the JSON message into a structured object (event wrapper containing shipment details).**
4. **Account Lookup: Looks up the account associated with the shipment using the AccountDao and validates its existence.**
5. **Shipment Processing: Delegates the actual shipment processing to ShipmentHandlerImpl, where business logic related to the shipment is executed.**
6. **Event Publishing: After processing, it uses ShipmentEventKafkaPublisher to publish shipment events to a Kafka topic.**
7. **Validation: Validates the shipment using validateShipment to ensure that the data adheres to business rules.**
8. **Logging: Logs all critical steps and decisions, ensuring traceability of the processing steps.**

**4. CARmsShipmentEventKafkaMessageProcessor**

### **Message Processing Method: onMessage()**

* **onMessage(final ConsumerRecord<String, String> consumerRecord): This is the core method that gets invoked whenever a Kafka message is received. It performs the following steps:**

#### **Extracting and Parsing the Message**

* **Extract message: The Kafka message (consumerRecord.value()) is retrieved and stored in the message variable.**
* **Convert message to Object: If the message is non-empty (StringUtils.isNotEmpty(message)), it is converted into an RMSShipmentWrapper object using JsonXmlUtil.convertJsonToObject(message, RMSShipmentWrapper.class). This wrapper contains both the event header and payload.**
* **Extract event details:**
  + **Event name: The event name is extracted from the header (shipmentWrapper.getHeader().getEventName()).**
  + **Source type: The source of return for the shipment is extracted from the payload (shipmentWrapper.getPayload().getSourceOfReturn()).**

#### **Validating the Event**

* **Event validation: The event is processed only if both the event name and the source type are valid.**
  + **The isValidEvent() method returns a BiPredicate that checks if:**
    - **The source type is present in the allowed list (appConfig.getRmsSourceTypeAllowList()).**
    - **The event name equals "RETURN\_SHIPPED".**
  + **If both conditions are met, the event is considered valid for processing.**

#### **Logging the Event Message**

* **Log the message: The logMessage() method logs the message depending on the configuration (appConfig.isCaRMSShippedEventLogsEnable()).**
  + **If logging is enabled, the entire message and the event name are logged.**
  + **If logging is disabled, only the event name is logged.**

#### **Handling the Shipment Event**

* **Handle the shipment: If the event is valid, the payload is passed to the caShipmentEventHandler.handleRMSShipmentPayload() method for further processing. This handler manages the core logic related to RMS shipment payloads.**
* **Log success: Once the shipment event is successfully processed, a log message is generated indicating that the shipment event (with the event name, source type, and order ID) has been processed successfully.**

#### **Handling Invalid Events**

* **Invalid events: If the event is not valid, the method logs that the event is not eligible for processing, including the event name, source type, and the list of allowed source types from the configuration.**

### **Helper Methods**

* **isValidEvent(): This method returns a BiPredicate that checks whether both the source type and the event name match the allowed criteria. Specifically:**
  + **The sourceType must be in the allowed list (appConfig.getRmsSourceTypeAllowList()).**
  + **The eventName must be "RETURN\_SHIPPED". This ensures that only return shipment events are processed.**
* **logMessage(): This method logs the message depending on whether logging is enabled in the configuration (appConfig.isCaRMSShippedEventLogsEnable()). If logging is disabled, it logs only the event name.**

### **Summary:**

1. **Message Consumption: The method consumes a shipment event message from Kafka.**
2. **Message Parsing: The message is parsed into an RMSShipmentWrapper object, and the event name and source type are extracted.**
3. **Event Validation: It validates whether the source type and event name meet specific conditions (e.g., source type must be in an allowed list, and the event must be a return shipment).**
4. **Logging: Logs the received message, depending on the configuration.**
5. **Event Processing: If valid, the event payload is passed to the CAShipmentEventHandler for handling.**
6. **Success Logging: Logs the success of the event processing, including details like the order ID, event name, and source type.**
7. **Invalid Event Handling: If the event is not valid, it logs that the event is not eligible for processing.**

**CarrierTrackingPollerConsumer**

### **Message Processing Logic**

**Message Consumption:**

* + **The main logic occurs in the onMessage() method, which processes messages received from Kafka.**
  + **A Transaction is created to track the operation.**

**Message Logging:**

* + **If logging is enabled in the application configuration, it logs the received message.**

**JSON Parsing:**

* + **The message is expected to be in JSON format, which is deserialized into a CarrierTrackingInfo object using JsonXmlUtil.convertJsonToObject().**
  + **Extracts the carrier name and transaction ID from the deserialized object.**

**Carrier Validation:**

* + **The code checks if the carrier name exists in the database using carrierDao.findByNameOrAltName().**
  + **If carrier processing is enabled in the configuration, it validates if the carrier is allowed to be processed.**

**Tracking ID Check:**

* + **If the tracking ID in the CarrierTrackingInfo is null, it logs a warning and marks the transaction as failed.**

**Carrier-Specific Logic:**

* + **The code handles various carriers (FedEx, UPS, LaserShip, etc.) with specific logic for each:**
    - **It checks if the carrier name matches known carriers and executes corresponding web service calls using their specific web service clients.**
    - **For each carrier, it creates a new CarrierTrackingExecutor task to process the tracking information concurrently using the executor.**

### **Example of Carrier Processing Logic**

* **FedEx Logic:**
  + **It checks if FedEx's SOAP or REST services are enabled and processes the tracking information accordingly.**
  + **It also checks if a package intercept is requested and executes the corresponding service call.**
* **UPS Logic:**
  + **Similar logic is applied for UPS, where it decides whether to use SOAP or REST based on configurations and receiver zip codes.**

**And other logics.**