Better way of Defining Event in Spring Boot-2024

If the order is cancelled, event should be published and action will be performed to cancel and to close the event. While creating multiple listener, make sure that for a particular purpose, create an Event class or create a simple Java Record class to wrap the actual entity, otherwise if there are multiple listeners and all the listeners are receiving the same entity, it will create weird result.

@Data  
**public class** ItemOrder {  
 **private** String name;  
 **private** String status;  
  
 **public** ItemOrder(String name) {  
 **this**.name = name;  
 }  
}

// Only for cancelling the order

**public** **record** CancelEventRecord(ItemOrder itemOrder) {

}

// Only for creating new order

**public** **record** NewOrderRecord(ItemOrder itemOrder) {

}

@Component

**public** **class** CancelledEventListener { 🡸 Listener for cancelling the order

@Order(1)

@EventListener

**public** **void** cancelOrder(CancelEventRecord eventRecord) {

System.***out***.println("Order is cancelled ..."+eventRecord.itemOrder().getName());

}

@Order(2)

@EventListener

**public** **void** closeOrder(CancelEventRecord eventRecord) {

System.***out***.println("Order is closed for operation ..."+eventRecord.itemOrder().getName());

}

@Order(3)

@EventListener

**public** **void** putBackOrder(CancelEventRecord eventRecord) {

System.***out***.println("Item has been put back in Inventory ..."+eventRecord.itemOrder().getName());

}

}

@Component

**public** **class** NewOrderEventListener { 🡸 Listener for creating new order

@EventListener

**public** **void** createOrder(NewOrderRecord orderEvent) {

System.***out***.println("Order is created and placed ..."+orderEvent.itemOrder().getName());

}

}

**public** **interface** OrderService { 🡸 Base Interface for create/cancel order

**void** createOrder(ItemOrder itemOrder);

**void** cancelOrder(ItemOrder itemOrder);

}

**Service Implementation Class**

@Service

**public** **class** OrderServiceImpl **implements** OrderService {

@Autowired

**private** **ApplicationEventPublisher eventPublisher**;

@Override

**public** **void** createOrder(ItemOrder itemOrder) {

//Wrap the order inside a record for the listener

NewOrderRecord newOrder = **new** NewOrderRecord(itemOrder);

**eventPublisher.publishEvent(newOrder);**

}

@Override

**public** **void** cancelOrder(ItemOrder itemOrder) {

//Wrap the order inside a record for the listener

CancelEventRecord cancelOrder = **new** CancelEventRecord(itemOrder);

**eventPublisher.publishEvent(cancelOrder);**

}

}

@Component

**public** **class** AutoRun {

@Autowired

**private** OrderService orderService;

@EventListener(ApplicationReadyEvent.**class**)

**public** **void** run() {

System.***out***.println("Application running ...");

// Get the order details from Database

ItemOrder order = **new** ItemOrder("Samsung Mobile");

order.setStatus("cancelled");

orderService.cancelOrder(order);

//Create a New Order

ItemOrder newItemOrder = **new** ItemOrder("Nokia Mobile");

order.setStatus("New Order");

orderService.createOrder(newItemOrder);

}

}

**Usage of @Order Annotation in Spring**

**@Order Annotation specifies the order of loading the bean by spring container. The lower the order(integer), higher is the precedence. So order of 0 will have more precedence than order of 10.**

**Use Case-1: Create an object and send that object in the validation phase in a serial order. An object contains Aadhar No, Pan No, Passport no, write different validation and let the object gets validated in a specific order.**

**Use Case-2: Send notification in a serial order like SMS, Email etc.**

**Complete code is given below.**

@Data

**public** **class** CoreData {

**private** String aadharNo;

**private** String passportNo;

**private** String panNo;

}

**public** **interface** Validation {

**boolean** validate(CoreData data);

}

@Component

@Order(2)

**public** **class** AadharValidation **implements** Validation {

@Override

**public** **boolean** validate(CoreData data) {

String aadharNo = data.getAadharNo();

System.***out***.println("Validating Aadhar No: "+aadharNo);

**return** **true**;

}

}

@Component

@Order(2)

**public** **class** PassportValidation **implements** Validation {

@Override

**public** **boolean** validate(CoreData data) {

String passportNo = data.getPassportNo();

System.***out***.println("Validating Passport No: "+passportNo);

**return** **true**;

}

}

@Component

@Order(3)

**public** **class** PanValidation **implements** Validation {

@Override

**public** **boolean** validate(CoreData data) {

String panNo = data.getPassportNo();

System.***out***.println("Validating PAN No: " + panNo);

**return** **true**;

}

}

**Notification Related**

**public** **interface** Notifier {

**void** notify(String message);

}

@Component

@Order(1)

**public** **class** EmailNotifier **implements** Notifier {

@Override

**public** **void** notify(String message) {

System.***out***.println(message + " sent on email to the user");

}

}

@Component

@Order(2)

**public** **class** SmsNotifier **implements** Notifier {

@Override

**public** **void** notify(String message) {

System.***out***.println(message + " sent via sms to the user");

}

}

**AutoRun class**

@Component

**public** **class** AutoRun {

@Autowired

**private** List<Validation> valdnList;

@Autowired

**private** List<Notifier> notificationList;

@Autowired

**private** List<Handler> handlers;

**public** **void** check1() {

CoreData data = **new** CoreData();

data.setAadharNo("AA-123456-7890");

data.setPassportNo("PP-45893-332");

data.setPanNo("P-78WSD-934");

valdnList.forEach( vldn -> System.***out***.println("Output From Valditation: "+vldn.validate(data)));

}

**public** **void** check2() {

String message = "Account created successfully ...";

notificationList.forEach( val -> val.notify(message));

}

**public** **void** check3() {

handlers.forEach( handler -> handler.handle());

}

@EventListener(ApplicationReadyEvent.**class**)

**public** **void** run() {

System.***out***.println("Application running ...");

check1();

check2();

check3();

}

}

**Example of using @Order in @Bean injection**

**public** **interface** Handler {

**void** handle();

}

@Data

**public** **class** AccountHandler **implements** Handler {

**private** String accountNo;

**private** String type;

**public** AccountHandler(String accountNo, String type) {

**this**.accountNo = accountNo;

**this**.type = type;

}

@Override

**public** **void** handle() {

System.***out***.println("Checking the account for account no: "+accountNo+" of type: "+type);

}

}

@Configuration

**public** **class** HandlerConfig {

@Bean @Order(1)

**public** AccountHandler savings() {

**return** **new** AccountHandler("1234", "Savings");

}

@Bean @Order(2)

**public** AccountHandler loan() {

**return** **new** AccountHandler("1234", "Loan");

}

@Bean @Order(3)

**public** AccountHandler demat() {

**return** **new** AccountHandler("1234", "Demat");

}

}