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;  
 }  
}

@Component

**public** **class** CancelledEventListner {

**@Order(1)**

**@EventListener**

**public** **void** cancelOrder(ItemOrder order) { 🡸 No need to create another event object

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

}

**@Order(2)**

**@EventListener**

**public** **void** closeOrder(ItemOrder order) { 🡸 No need to create another event object

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

}

**@Order(3)**

**@EventListener**

**public** **void** putBackOrder(ItemOrder order) { 🡸 No need to create another event object

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

}

}

**public interface** OrderService {  
 **void** cancellOrder(ItemOrder order);  
}

@Service

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

@Autowired

**private** ApplicationEventPublisher eventPublisher;

@Override

**public** **void** cancellOrder(ItemOrder order) {

**if**(order.getStatus().equalsIgnoreCase("cancelled")) {

eventPublisher.publishEvent(order);

}

}

}

@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 Item**Order("Samsung Mobile");  
 order.setStatus("cancelled");  
 orderService.cancellOrder(order);  
 }  
}

**Usage of @Order Annotation in Spring**

**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**;

}

}

**AutoRun class**

@Component

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

@Autowired

**private** List<Validation> valdnList;

@Autowired

**private** List<Notification> notificationList;

**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.send(message));

}

@EventListener(ApplicationReadyEvent.**class**)

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

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

check1();

check2();

}

}