**Publish and Subscribe Events in Spring Boot – 2022**

**Create Simple Event class**

@Data

**public** **class** PersonEvent **extends** ApplicationEvent {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** **long** id;

**private** String name;

**public** PersonEvent(Object source) {

**super**(source);

}

}

**You can also create a class without extending ApplicationEvent.**

@Data

**public** **class** PersonEvent {

**private** String name;

}

**Create Event Listener**

@Service

**public** **class** EventConsumerService {

**@EventListener**

**public** **void** consumeEvent(PersonEvent data) {

System.***out***.println("Received Event for processing: "+data);

}

**@EventListener**

**public** **void** processPerson(PersonEvent pe) {

System.***out***.println("Person for processing: "+pe);

}

}

**Create Event Publisher**

@Service

**public** **class** EventPublisherService {

**@Autowired**

**private ApplicationEventPublisher publisher;**

**public** **void** performOperation() {

PersonEvent dataEvent = **new** PersonEvent (**this**);

dataEvent.setId(123);

dataEvent.setName("John Abraham");

publisher.**publishEvent**(dataEvent);

}

**public** **void** createPerson() {

PersonEvent pe = **new** PersonEvent();

pe.setName("Ramana Maharshi");

publisher.publishEvent(pe);

}

}

**A simple autostart to test the application**

@Component

**public** **class** AutoStart {

@Autowired

**private** EventPublisherService pubService;

@EventListener(ApplicationReadyEvent.**class**)

**public** **void** afterStartup() {

pubService.performOperation();

pubService.createPerson();

}

}

**Spring Boot Main Application**

@SpringBootApplication

**public** **class** EventPubSubMain {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(EventPubSubMain.**class**, args);

}

}

**Maven pom.xml**

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.5.RELEASE</version>

<relativePath />

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<java.version>11</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

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.

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

@Data  
**public class** CancelledOrderEvent {  
  
 **private** ItemOrder order;  
  
 **public** CancelledOrderEvent(ItemOrder order) {  
 **this**.order = order;  
 }  
}

@Component  
**public class** CancelledEventListner {  
  
 @Order(1)  
 @EventListener  
 **public void** cancelOrder(CancelledOrderEvent event) {  
 ItemOrder order = event.getOrder();  
 System.***out***.println("Order is cancelled ..."+order.getName());  
 }  
  
 @Order(2)  
 @EventListener  
 **public void** closeOrder(CancelledOrderEvent event) {  
 ItemOrder order = event.getOrder();  
 System.***out***.println("Order is closed for operation ..."+order.getName());  
 }  
  
 @Order(3)  
 @EventListener  
 **public void** putBackOrder(CancelledOrderEvent event) {  
 ItemOrder order = event.getOrder();  
 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")) {  
 CancelledOrderEvent event = **new** CancelledOrderEvent(order);  
 eventPublisher.publishEvent(event);  
 }  
 }  
}

@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();

}

}