**Event Management and @Order Annotation Usage in Spring Boot-2022**

Spring Main application

@SpringBootApplication  
public class SampleMainApplication {  
 public static void main(String[] args) {  
 SpringApplication.*run*(SampleMainApplication.class, args);  
 }  
}

**Custom Event Creation in Spring**

public class **CustomSpringEvent extends ApplicationEvent** {  
 private String message;  
 public CustomSpringEvent(Object source, String message) {  
 super(source);  
 this.message = message;  
 }  
 public String getMessage() {  
 return message;  
 }  
}

**Event Publisher**

@Component  
public class CustomSpringEventPublisher

**@Autowired  
 private ApplicationEventPublisher applicationEventPublisher;**  
 public void publishCustomEvent(final String message) {  
 System.*out*.println("Publishing custom event. ");  
 CustomSpringEvent customSpringEvent = new CustomSpringEvent(this, message);  
 applicationEventPublisher.publishEvent(customSpringEvent);  
 }  
}

**Event Listener**

@Component  
public class CustomSpringEventListener implements ApplicationListener<CustomSpringEvent> {  
 @Override  
 public void onApplicationEvent(CustomSpringEvent event) {  
 System.*out*.println("Received spring custom event - " + event.getMessage());  
 }  
}

**There is a better way to create a listener**

@Component  
public class AnotherListener {

**@EventListener(condition = "")** public void onMyEvent (CustomSpringEvent event) {  
 System.*out*.print("event received: " + event.getMessage());  
 }  
}

**Controller Layer**

@RestController  
public class SampleController {  
 @Autowired  
 private SampleService service;  
  
 @PostMapping(path = "/user")  
 public ResponseEntity<String> createUser(@RequestBody User user) {  
 service.createUser(user);  
 return new ResponseEntity<>("User Creation Successful", HttpStatus.*CREATED*);  
 }  
}

**Service Layer**

public interface SampleService {  
 void createUser(User user);  
}

@Service  
public class SampleServiceImpl implements SampleService {

**@Autowired  
 private CustomSpringEventPublisher publisher;**  
 public void createUser(User user) {  
 **publisher.publishCustomEvent("User Created successfully ...");** }  
}

**Usage of @Order Annotation in Spring**

@Component  
public class ApplicationStartUp {  
  
 @Autowired  
 private List<Validation> validationList;  
  
 @EventListener(ApplicationReadyEvent.class)  
 public void afterStartup() {  
 System.*out*.println("Application Started at : " + CommonUtil.*timeNow*());  
 check();  
 }  
  
 public void check() {  
 for(int i = 0; i < validationList.size(); i++) {  
 System.*out*.println(validationList.get(i).validate());  
 }  
 }  
}

public interface Validation {  
 String validate();  
}

**Validations in Order**

@Component  
@Order(1)  
public class PanCardValidation implements Validation {  
 @Override  
 public String validate() {  
 return "Success";  
 }  
}

@Component  
@Order(2)  
public class AadharValidationImpl implements Validation {  
 @Override  
 public String validate() {  
 return "Sucess";  
 }  
}

@Component  
@Order(3)  
public class VoterIdValidationImpl implements Validation {  
 @Override  
 public String validate() {  
 return "Sucess";  
 }  
}