**Spring @ConditionalOnProperty @Order- 2022**

The annotation is used to conditionally create a Spring bean depending on the configuration of a property.

**public** **interface** NotificationSender {

**void** send(String message); 🡸 Interface

}

**public** **class** EmailNotification **implements** NotificationSender {

@Override

**public** **void** send(String message) { 🡸 Interface Implmentation

System.***out***.println("Email Notification: " + message);

}

}

**public** **class** SmsNotification **implements** NotificationSender {

@Override

**public** **void** send(String message) { 🡸 Interface Implmentation

System.***out***.println("SMS Notification: " + message);

}

}

**Configuration to create bean based upon the value in application.properties**

@Configuration

**public** **class** NotificationConfig {

@Bean(name = "emailNotification")

**@ConditionalOnProperty(prefix = "notification", name = "service", havingValue = "email")**

**public** NotificationSender emailNotificationSender() {

**return** **new** EmailNotification();

}

@Bean(name = "smsNotification")

**@ConditionalOnProperty(prefix = "notification", name = "service", havingValue = "sms")**

**public** NotificationSender smsNotificationSender() {

**return** **new** SmsNotification();

}

}

@Component

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

@Autowired

**private** NotificationSender sender;

@EventListener(ApplicationReadyEvent.**class**)

**public** **void** onStartUp() {

System.***out***.println("I am ready to run now...");

sender.send("Sending telemarketing message: ");

}

}

**application.properties**

server.port=8080

spring.mvc.pathmatch.matching-strategy = ANT\_PATH\_MATCHER

#notification.service=email

notification.service=sms

**@Order in Spring**

@Order Annotation specifies the order of loading the bean by spring container. The lower the order(integer), higher is the precedence.

Use Case: Perform the validation in proper order, first Aadhar, then passport and then pan no validation.

**public** **interface** Validator {

**void** validate(Person person); 🡸 Interface

}

**@Order(1)**

@Service

**public** **class** PassportValidation **implements** Validator {

@Override 🡸 Interface Implementation

**public** **void** validate(Person person) {

System.***out***.println("Validating person passport no: "+person.getPassportNo());

}

}

**@Order(2)**

@Service

**public** **class** AadharValidation **implements** Validator {

@Override 🡸 Interface Implementation

**public** **void** validate(Person person) {

System.***out***.println("Validating person addhar no: "+person.getAadharNo());

}

}

@Data

**public** **class** Person {

**private** String passportNo;

**private** String aadharNo; 🡸 Data Model

**private** String panNo;

}

@Component

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

@Autowired

**private** List<Validator> validators;

@EventListener(ApplicationReadyEvent.**class**)

**public** **void** onStartUp() {

System.***out***.println("I am ready to run now...");

Person p = **new** Person();

p.setAadharNo("AA-12345"); 🡸 Actual Implementation

p.setPanNo("PA-12789-QRS");

p.setPassportNo("PP-IND-34567");

**for**(Validator validator : validators)

validator.validate(p);

}

}

## @Lookup

**A method annotated with @Lookup tells Spring to return an instance of the method's return type when we invoke it. *@Lookup annotation in Spring*** returns a new bean on every call without we implementing the method.

Example is given below.

@Component("a")

@Scope("prototype") 🡸 You have to use to get different instances

**public** **class** A {

**public** **void** show() {

System.***out***.println("A class ...");

}

}

@Component

**public** **class** ObjectRetriever {

@Lookup(value="a")

**public** A getObject() {

**return** **null**; 🡸 Mark it, it returns null, creates instance by CGlib

}

}

How to use

@Autowired

**private** ObjectRetriever retriever;

@EventListener(ApplicationReadyEvent.**class**)

**public** **void** onStartUp() {

System.***out***.println("retriever 1 ---->"+retriever.getObject()); //new instance

retriever.getObject().show();

System.***out***.println("retriever 1 ---->"+retriever.getObject()); //new instance

retriever.getObject().show();

}