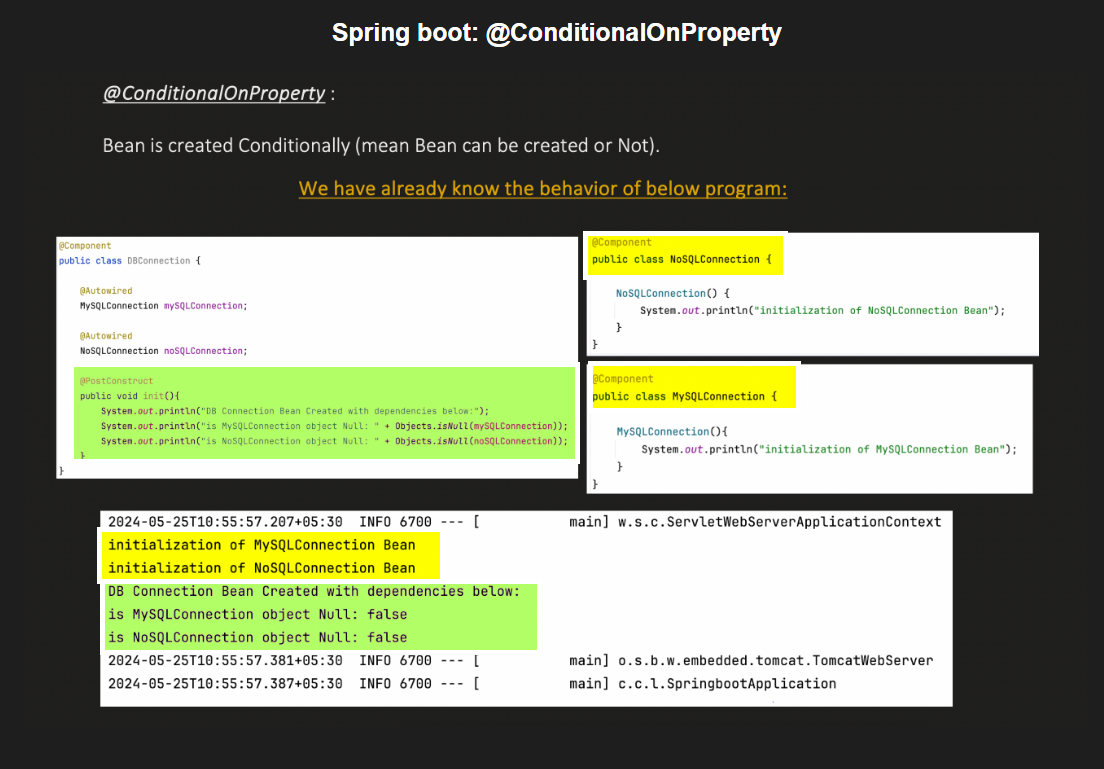
Udemy Url - [Notebook](https://notebook.zohopublic.in/public/notes/dcr5z97c7e76aa1454ea6a57e1a2dde4f8785)



@ConditionalOnProperty is a Spring Boot annotation used to **conditionally enable or disable a bean** based on the presence and value of a specific property in the application configuration (application.properties or application.yml).

**📌 Syntax:**

@ConditionalOnProperty(

name = "property.name",

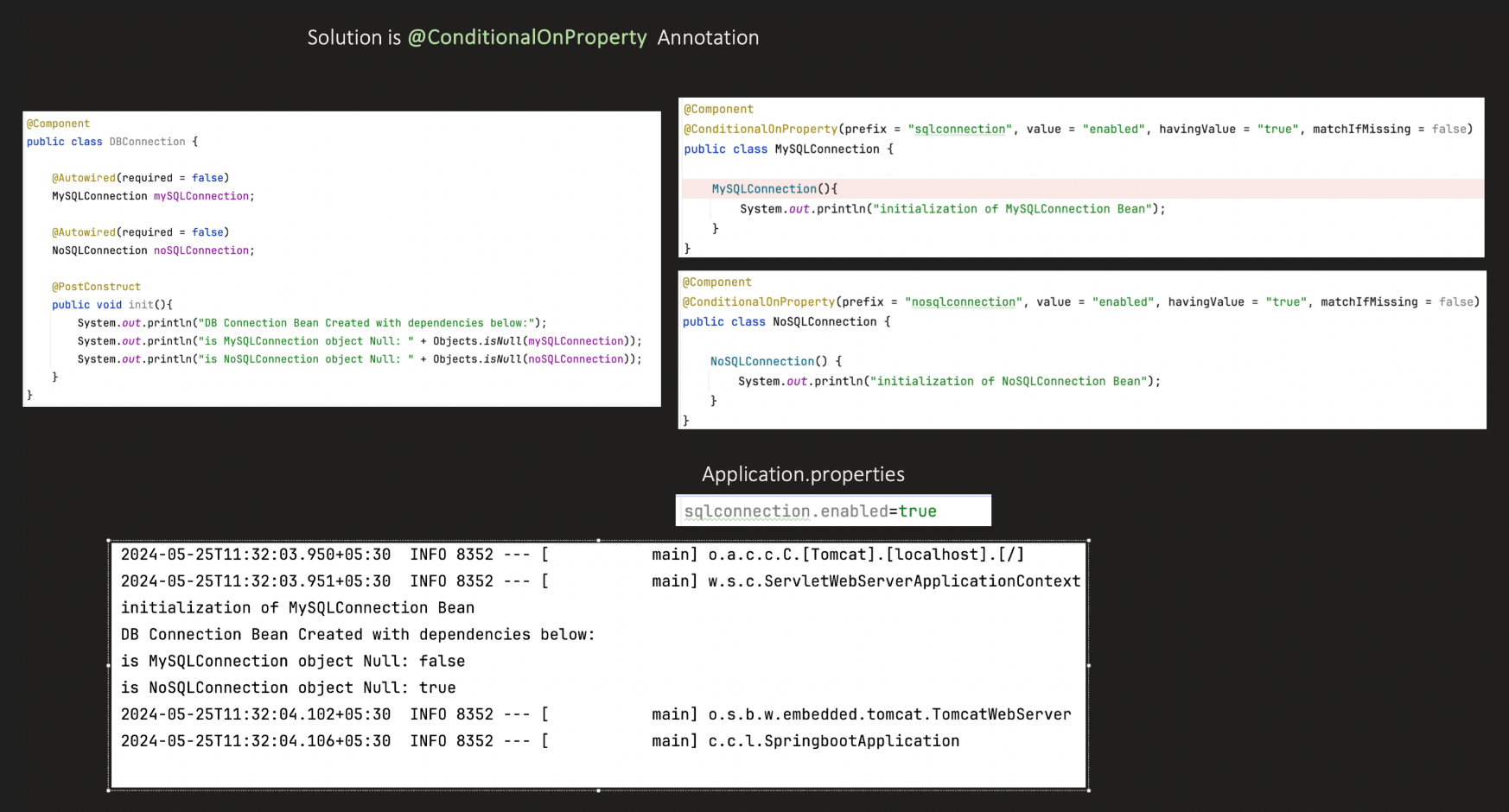
havingValue = "expectedValue",

matchIfMissing = false

)

A screenshot of a computer

AI-generated content may be incorrect.



Here @Autowired(**required = false**) is required as if condition on satisfy then the bean will not get created and then the application will fail to start.

A black screen with white text

AI-generated content may be incorrect.

# 2. @ConditionalOnMissingBean

@ConditionalOnMissingBean is a **Spring Boot conditional annotation** used to **register a bean only if a specific bean is *not already present*** in the Spring context.

**📌 Syntax:**

@ConditionalOnMissingBean(

value = MyService.class

)

**✅ Purpose:**

It helps prevent duplicate bean creation by allowing a bean to be registered **only if a matching bean hasn't already been defined elsewhere** (e.g., user config, test config, or auto-configuration).

**🔧 Example:**

@Configuration

public class MyConfig {

@Bean

@ConditionalOnMissingBean(MyService.class)

public MyService myService() {

return new MyService();

}

}

* If **no other MyService bean** exists, Spring will create this one.
* If **any other bean of type MyService is already defined**, this bean will be skipped.

**📋 Parameters:**

| **Parameter** | **Description** |
| --- | --- |
| value | The type(s) of bean(s) to check. |
| name | (Optional) Check for bean with specific name. |
| type | (Optional) Fully-qualified class name as String. |
| annotation | (Optional) Only check beans annotated with a specific annotation. |
| ignored | (Optional) Beans to ignore during matching. |

# 3. @ConditionalOnBean

@ConditionalOnBean is a **Spring Boot conditional annotation** that tells Spring to **create a bean only if another bean of a specific type or name is already present** in the application context.

**📌 Syntax:**

@ConditionalOnBean(MyService.class)

**✅ Purpose:**

It is used when your bean depends on another bean and should only be created **if that dependent bean exists** in the context.

**🔧 Example:**

@Configuration

public class MyConfig {

@Bean

public MyService myService() {

return new MyService();

}

@Bean

@ConditionalOnBean(MyService.class)

public MyController myController() {

return new MyController();

}

}

In this case:

* myController() will only be registered if myService() exists.

**📋 All Parameters:**

| **Parameter** | **Type** | **Description** |
| --- | --- | --- |
| value | Class<?>[] | Array of classes (types of beans) to check. If beans of these types exist, the condition matches. |
| type | String[] | Fully qualified class names to check by type, used when the actual class may not be on the classpath at compile time. |
| name | String[] | Array of bean names to check for. If these bean names exist, the condition matches. |
| annotation | Class<? extends Annotation> | Specifies that a bean annotated with this annotation must exist. |
| ignored | Class<?>[] | Beans of these types are ignored during the match. |
| ignoredType | String[] | Fully qualified class names of types to ignore (when using type). |

# **4.** @ConditionalOnClass

@ConditionalOnClass is a **Spring Boot conditional annotation** used to **conditionally enable a bean or configuration class only if a specific class is present on the classpath**.

**✅ Purpose:**

It allows Spring to include a bean/config **only if a given class is available**, helping in auto-configuration and modularity.  
This is **very useful in libraries** or situations where some classes (like a JDBC driver or a Kafka client) may or may not be present.

**📌 Syntax:**

@ConditionalOnClass(name = "com.example.SomeClass")

OR

@ConditionalOnClass(SomeClass.class)

**🔧 Example:**

@Configuration

@ConditionalOnClass(name = "com.mysql.cj.jdbc.Driver")

public class MySQLAutoConfiguration {

@Bean

public DataSource mysqlDataSource() {

// configure and return MySQL DataSource

}

}

This configuration will **only be loaded if the MySQL JDBC driver class is on the classpath**.

**📋 Parameters:**

| **Parameter** | **Type** | **Description** |
| --- | --- | --- |
| value | Class<?>[] | One or more class references to check. If **all** are found, the condition matches. |
| name | String[] | Fully-qualified class names as strings. Use when the class might not be on the classpath at compile time. |

**✅ Use Cases:**

* Spring Boot auto-configurations (spring-boot-autoconfigure module) use this heavily.
* For example, only configure Redis support if org.springframework.data.redis.core.RedisTemplate is present.

# 5. @ConditionalOnExpression

@ConditionalOnExpression is a **Spring Boot conditional annotation** used to **control bean registration based on the result of a SpEL (Spring Expression Language) expression**.

**✅ Purpose:**

It allows for **flexible, expression-based conditions**—beyond simple property checks—when deciding whether a bean or configuration should be created.

**📌 Syntax:**

@ConditionalOnExpression("${feature.enabled:true}")

**📋 Parameters:**

| **Parameter** | **Type** | **Description** |
| --- | --- | --- |
| value | String | The SpEL expression to evaluate. If it resolves to true, the bean will be created. |

**🔧 Example 1: Based on Property**

@Bean

@ConditionalOnExpression("${my.feature.enabled:true}")

public MyService myService() {

return new MyService();

}

* This bean will be registered **only if** my.feature.enabled=true (or if the property is missing, it defaults to true).

**🔧 Example 2: Using Mathematical/Logical Expression**

@Bean

@ConditionalOnExpression("#{${app.version} > 2}")

public MyNewFeature myNewFeature() {

return new MyNewFeature();

}

✅ This bean will be created only if app.version > 2.

@ConditionalOnExpression("#{${app.version:1} > 2}")

➡️ Uses 1 as defaults if app.version is not defined.

**🔧 Example 3: Combining Properties**

@Bean

@ConditionalOnExpression("${feature.x.enabled:true} && ${feature.y.enabled:false}")

public MyFeature feature() {

return new MyFeature();

}

**⚠️ Caveats:**

* The expression **must resolve to a boolean**. Means if feature.x.enabled = parul is prop file then the application will fail at startup as parul can not be changes to Boolean. **Solution Use a comparison in the expression:**

@ConditionalOnExpression("'${feature.alpha}' == 'parul'")

Or provide the default value.

* If the expression fails (e.g., due to missing property and no default), the context may fail to start unless handled with defaults or matchIfMissing.

# 6. @ConditionalOnJava

@ConditionalOnJava is a **Spring Boot conditional annotation** used to **conditionally register a bean or configuration based on the Java version** used to run the application.

**✅ Purpose:**

It helps ensure that a bean is created only if the **JVM version** meets a specific requirement — useful when certain features or libraries only work with specific Java versions.

**📌 Syntax:**

@ConditionalOnJava(

value = JavaVersion.ELEVEN,

range = ConditionalOnJava.Range.EQUAL\_OR\_NEWER

)

**📋 Parameters:**

| **Parameter** | **Type** | **Description** |
| --- | --- | --- |
| value | JavaVersion | The target Java version (e.g., JavaVersion.EIGHT, JavaVersion.SEVENTEEN) |
| range | ConditionalOnJava.Range | Whether to match EQUAL, OLDER\_THAN, or EQUAL\_OR\_NEWER (default: EQUAL\_OR\_NEWER) |

**🔧 Example 1: Enable Bean for Java 11 or higher**

@Configuration

@ConditionalOnJava(value = JavaVersion.ELEVEN)

public class Java11Config {

@Bean

public MyService myService() {

return new MyService();

}

}

➡️ This bean will be created if the application is running on **Java 11 or newer** (because the default range is EQUAL\_OR\_NEWER).

**🔧 Example 2: Enable only for Java 8**

@ConditionalOnJava(value = JavaVersion.EIGHT, range = ConditionalOnJava.Range.EQUAL)

➡️ Bean will be created **only** if Java 8 is being used.

**🔧 Example 3: Enable only for Java older than 17**

@ConditionalOnJava(value = JavaVersion.SEVENTEEN, range = ConditionalOnJava.Range.OLDER\_THAN)

**💡 Use Case:**

You’re writing an auto-configuration that uses a feature only available in Java 11+ (like HttpClient), so you restrict the config to run only on Java 11 or newer.

# 7. @Conditional

@Conditional is a **core Spring annotation** (available in Spring Framework, not just Spring Boot) that allows you to conditionally **register beans or configuration classes** based on **custom logic** defined in a condition class.

**✅ Purpose:**

Unlike other @ConditionalOnX annotations (which are specific and prebuilt), @Conditional is **generic and powerful** — it lets you plug in **your own logic** to decide whether a bean should be created.

**📌 Syntax:**

@Conditional(MyCustomCondition.class)

**👷 Custom Condition Class:**

You must implement the Condition interface and override the matches() method.

public class MyCustomCondition implements Condition {

@Override

public boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata) {

// Custom logic here (e.g., check env properties, Java version, etc.)

String env = context.getEnvironment().getProperty("app.env");

return "prod".equalsIgnoreCase(env);

}

}

**🔧 Example: Conditional Bean for "prod" Environment**

@Configuration

public class MyConfig {

@Bean

@Conditional(MyCustomCondition.class)

public MyService myService() {

return new MyService();

}

}

In this example, MyService will be registered **only if app.env=prod**.

**🧠 Components You Get in matches():**

| **Component** | **Description** |
| --- | --- |
| ConditionContext | Gives access to Environment, BeanFactory, ClassLoader, Registry, etc. |
| AnnotatedTypeMetadata | Gives metadata about the annotated class/method, including attributes. |