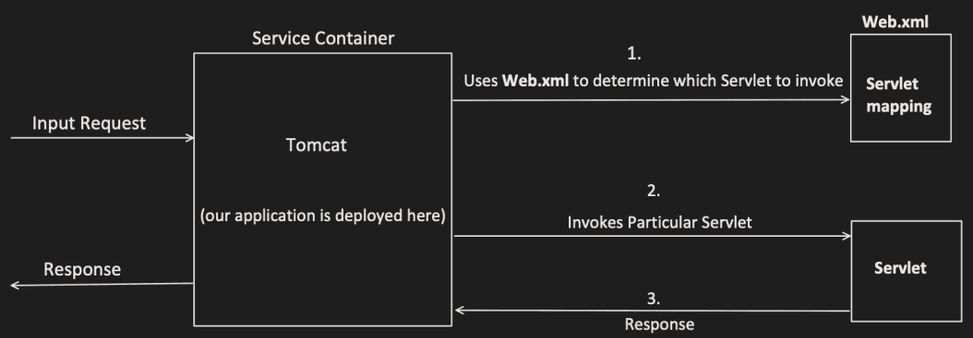
Before we talk about Spring or Spring Boot, first we need to understand about "**SERVLET**" and "**Servlet Container**"

- Provide foundation for building web applications.
- Servlet is a Java Class, which handles client request, process it and return the response.
- And Servlet Container are the ones which manages the Servlets.



Servlet1:

```java
@WebServlet("/demoservletone/*")
public class DemoServlet1 extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response) {

        String requestPathInfo = request.getPathInfo();

        if(requestPathInfo.equals("/")) {
            //do something
        }
        else if(requestPathInfo.equals("/firstendpoint")) {
            //do something
        }
        else if(requestPathInfo.equals("/secondendpoint")) {
            //do something
        }
    }

    @Override
    protected void doPut(HttpServletRequest request,
                         HttpServletResponse response) {
        //do something
    }
}
```

Web.xml

```xml
<!-- my first servlet configuration below-->
<servlet>
    <servlet-name>DemoServlet1</servlet-name>
    <servlet-class>DemoServlet1</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>DemoServlet1</servlet-name>
    <url-pattern>/demoservletone</url-pattern>
    <url-pattern>/demoservletone/firstendpoint</url-pattern>
    <url-pattern>/demoservletone/secondendpoint</url-pattern>
</servlet-mapping>

<!-- my second servlet configuration below-->
<servlet>
    <servlet-name>DemoServlet2</servlet-name>
    <servlet-class>DemoServlet2</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>DemoServlet2</servlet-name>
    <url-pattern>/demoservlettwo</url-pattern>
</servlet-mapping>
```

Servlet2:

```java
@WebServlet("/demoservlettwo/*")
public class DemoServlet2 extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response) {

        //do something
    }

    @Override
    protected void doPut(HttpServletRequest request,
                         HttpServletResponse response) {
        //do something
    }
}
```

Spring Framework, solve challenges which exists with Servlets.

- Removal of web.xml
  - this web.xml over the time becomes too big and becomes very difficult to manage and understand.
  - Spring framework introduced Annotations based configuration.

- Inversion of Control (IoC)
  - Servlets depends on Servlet container to create object and maintain its lifecycle.
  - IoC is more flexible way to manage object dependencies and its lifecycle (though Dependency injection)

- Unit Testing is much harder
  - As the object creation depends on Servlet container, mocking is not easy. Which makes Unit testing process harder.
  - Spring dependency injection facility makes the Unit testing very easy.

- Difficult to manage REST APIs
  - Handling different HTTP methods, request parameters, path mapping make code little difficult to understand.
  - Spring MVC provides an organised approach to handle the requests and its easy to build RESTful APIs.

There are many other areas where Spring framework makes developer life easy such as : integration with other technology like hibernate, adding security etc...

The most important feature of Spring framework is **DEPENDENCY INJECTION** or **Inversion of Control (IoC)**

Lets see an example **without** Dependency Injection:

```
public class Payment {

    User sender = new User();

    void getSenderDetails(String userID){
        sender.getUserDetails(userID);
    }
}
```

```
public class User {

    public void getUserDetails(String id) {
        //do something
    }
}
```

Payment class is creating an instance of User class, and there is one Major problems with this and i.e.

Tight coupling: Now payment class is tightly coupled with User class.

How?
-> Suppose I want to write Unit test cases for Payment "getSenderDetails()" method, but now I can not easily MOCK "User" object, as Payment class is creating new object of User, so it will invoke the method of User class too.

-> Suppose in future, we have different types of User like "admin", "Member" etc., then with this logic, I can not change the user dynamically.

Now, Lets see an example **with** Dependency Injection:

```java
@Component
public class Payment {

    @Autowired
    User sender;

    void getSenderDetails(String userID){
        sender.getUserDetails(userID);
    }
}
```

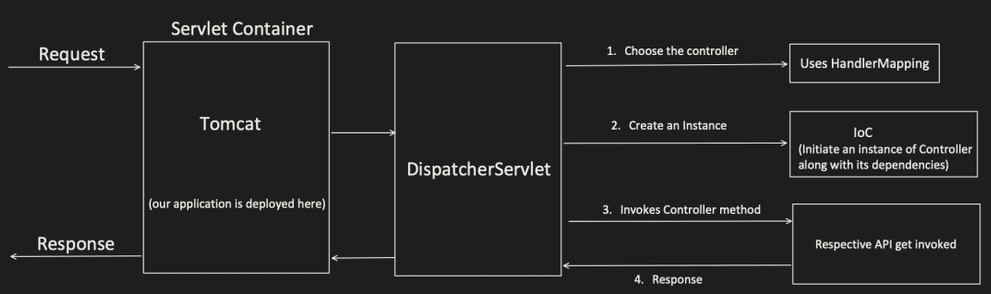
```java
@Component
public class User {

    public void getUserDetails(String id) {
        //do something
    }
}
```

@Component: tells Spring that, you have to manage this class or bean.
@Autowired: tells Spring to resolve and add this object dependency.

The another important feature of Spring framework is lot of **INTEGRATION** available with other frameworks.

This allow Developers to choose different combination of technologies and framework which best fits their requirements like:

- Integration with Unit testing framework like Junit or Mockito.
- Integration with Data Access framework like Hibernate, JDBC, JPA etc.
- Integration with Asynchronous programming.
- Similar way, it has different integration available for:
    - Caching
    - Messaging
    - Security etc.



pom.xml

```xml
<modelVersion>4.0.0</modelVersion>
<groupId>com.conceptandcoding</groupId>
<artifactId>learningspringboot</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>springboot application</name>
<description>project for learning springboot</description>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>6.1.4</version>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
            <version>2.5</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

Controller class

```java
@Controller
@RequestMapping("/paymentapi")
public class PaymentController {

    @Autowired
    PaymentDAO paymentService;

    @GetMapping("/payment")
    public String getPaymentDetails() {
        return paymentService.getDetails();
    }
}
```

Config class

```java
@Configuration
@EnableWebMvc
@ComponentScan(basePackages = "com.conceptandcoding")
public class AppConfig {
    // add configuration here if required
}
```

Dispatcher Servlet class

```java
public class MyApplicationInitializer extends
        AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return null;
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[]{AppConfig.class};
    }

    @Override
    protected String[] getServletMappings() {
        return new String[]{"/"};
    }
}
```

Spring Boot, solve challenges which exists with Spring MVC.

1. Dependency Management: No need for adding different dependencies separately and also their compatible version headache.

```xml
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.3</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.conceptandcoding</groupId>
<artifactId>learningspringboot</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>springboot application</name>
<description>project for learning springboot</description>
<properties>
    <java.version>17</java.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

2. Auto Configuration : No need for separately configuring "DispatcherServlet", "AppConfig" , "EnableWebMvc", "ComponentScan". Spring boot add internally by-default.

```java
@SpringBootApplication
public class SpringbootApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringbootApplication.class, args);
    }
}
```

3. Embedded Server:

In traditional Spring MVC application, we need to build a WAR file, which is a packaged file containing your application's classes, JSP pages, configuration files, and dependencies.
Then we need to deploy this WAR file to a servlet container like Tomcat.

But in Spring boot, Servlet container is already embedded, we don't have to do all this stuff. Just run the application, that's all.

So, what is Spring boot?

- It provides a quick way to create a production ready application.

- It is based on Spring framework.

- It support **"Convention over Configuration".**
  Use default values for configuration, and if developer don't want to go with convention(the way something is done), they can override it.

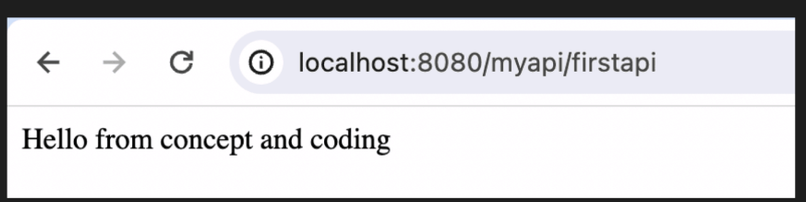
- It also help to run an application as quick as possible.

pom.xml

```xml
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.3</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.conceptandcoding</groupId>
<artifactId>learningspringboot</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>springboot application</name>
<description>project for learning springboot</description>
<properties>
    <java.version>17</java.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

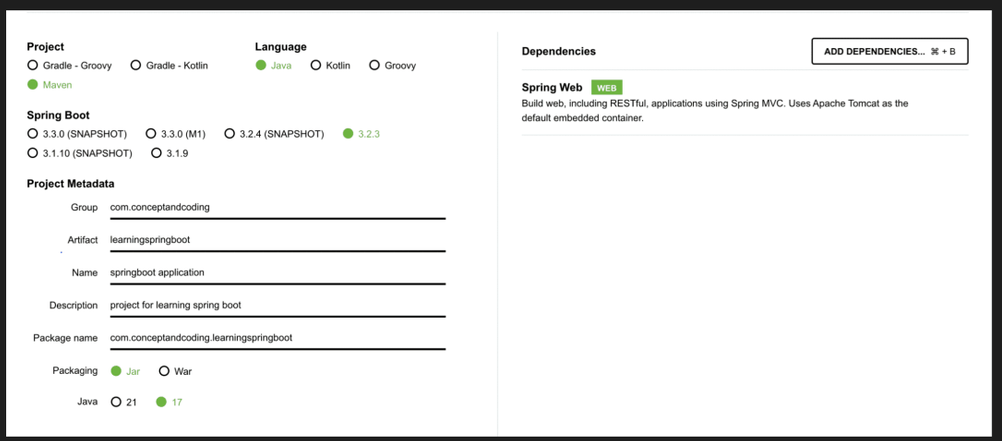
```java
@SpringBootApplication
public class SpringbootApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringbootApplication.class, args);
    }
}
```

```java
@RestController
@RequestMapping("/myapi")
public class MyController {

    @GetMapping("/firstapi")
    public String getData() {
        return "Hello from concept and coding";
    }
}
```

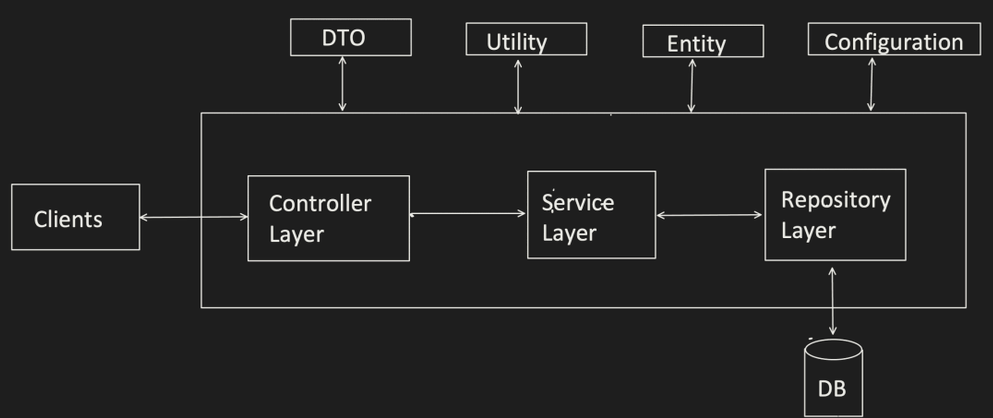


Setting up the Project

1. Go to Spring Initializr i.e. "start.spring.io"



LAYERED ARCHITECTURE




```java
@RestController
@RequestMapping("/payments")
public class PaymentController {

    @Autowired
    PaymentService paymentService;

    @GetMapping("/{id}")
    public ResponseEntity<PaymentResponse> getPaymentById(@PathVariable Long id) {

        //map incoming data to internal request DTO
        PaymentRequest internalRequestObj = new PaymentRequest();
        internalRequestObj.setPaymentId(id);

        //pass this internalRequestObj to further layer for processing
        PaymentResponse payment = paymentService.getPaymentDetailsById(internalRequestObj);

        //return the Response DTO
        return ResponseEntity.ok(payment);

    }
}
```

```java
@Service
public class PaymentService {

    @Autowired
    PaymentRepository paymentRepository;

    public PaymentResponse getPaymentDetailsById(PaymentRequest internalRequestObj) {

        PaymentEntity paymentModel = paymentRepository.getPaymentById(internalRequestObj);

        //map it to response obj
        PaymentResponse paymentResponse = mapModelToResponseDTO(paymentModel);
        return paymentResponse;
    }

    private PaymentResponse mapModelToResponseDTO(PaymentEntity paymentEntity){

        PaymentResponse response = new PaymentResponse();
        response.setPaymentId(paymentEntity.getId());
        response.setAmount(paymentEntity.getPaymentAmount());
        response.setCurrency(paymentEntity.getPaymentCurrency());
        return response;
    }
}
```

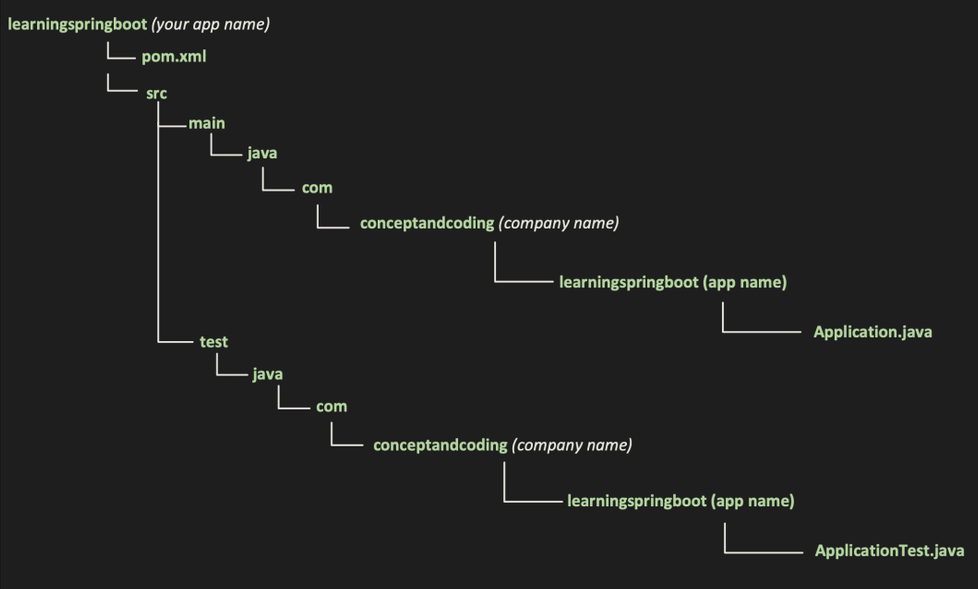
```java
@Repository
public class PaymentRepository {

    public PaymentEntity getPaymentById(PaymentRequest request){
        PaymentEntity paymentModel = executeQuery(request);
        return paymentModel;
    }

    private PaymentEntity executeQuery(PaymentRequest request){
        //connect with DB and fetch the data
        PaymentEntity payment = new PaymentEntity();
        payment.setId(request.getPaymentId());
        payment.setPaymentCurrency("INR");
        payment.setPaymentAmount(100.00);
        return payment;
    }
}
```

What is Maven:

- It's a project management tool. Helps developers with:
  - Build generation
  - Dependency resolution
  - Documentation etc.

- Maven uses POM (Project Object Model) to achieve this.

- When "maven" command is given, it looks for "pom.xml" in the current directory & get needed configuration.




```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.3</version>
        <relativePath/>
    </parent>
    <groupId>com.conceptandcoding</groupId>
    <artifactId>learningspringboot</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>springboot application</name>
    <properties>
        <java.version>17</java.version>
    </properties>
    <repositories>
        <repository>
            <id>central</id>
            <url>https://repo.maven.apache.org/maven2</url>
        </repository>
    </repositories>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

It specifies the XML schema, also make sure that our XML adheres to correct structure and version defined by maven.

Used to define the Parent project. Current project inherit the configurations from the specified Parent project. Which in turn might get inherited from the Super Pom.

If this <parent> field is not specified, maven by-default inherit the configurations from "Super Pom".
This is the link of maven Super POM:
https://maven.apache.org/ref/3.6.3/maven-model-builder/super-pom.html

Unique identifier of your project

Define Key-Value pair for configuration. Can be referenced through out the pom file.

This is from where Maven look for project dependencies and download the artifacts (jars)

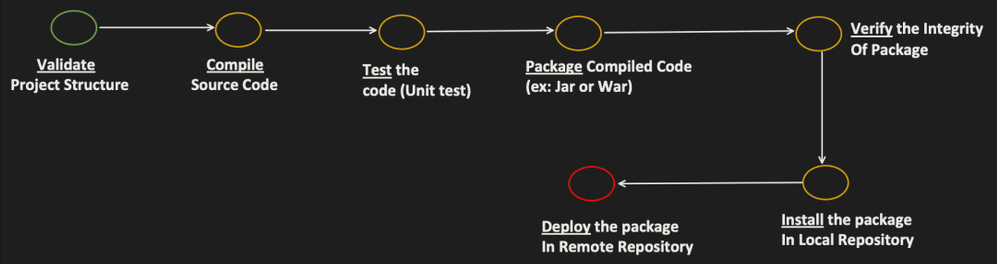
This is where we declare the dependencies that our project relies on.

Lets first understand the BUILD LIFECYCLE

Maven Build lifecycle phases:

- If you want to run "package" phase, all its previous phase will get executed first.

- And if you want to run specific goal of a particular phase, then all the goals of previous phases + current phase goals before the one you defined will get run.

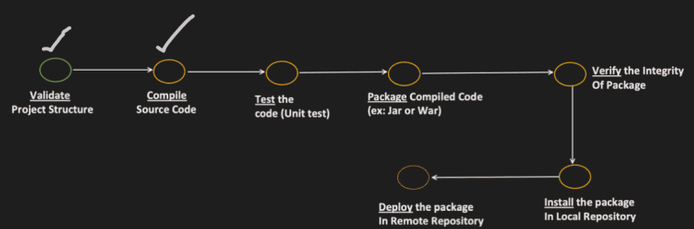


Maven already has Validate phase defined and its goal, but if we want to add more goals or task, then we can use <build> element. And add the goal to specific phase.

Validate : `mvn validate`

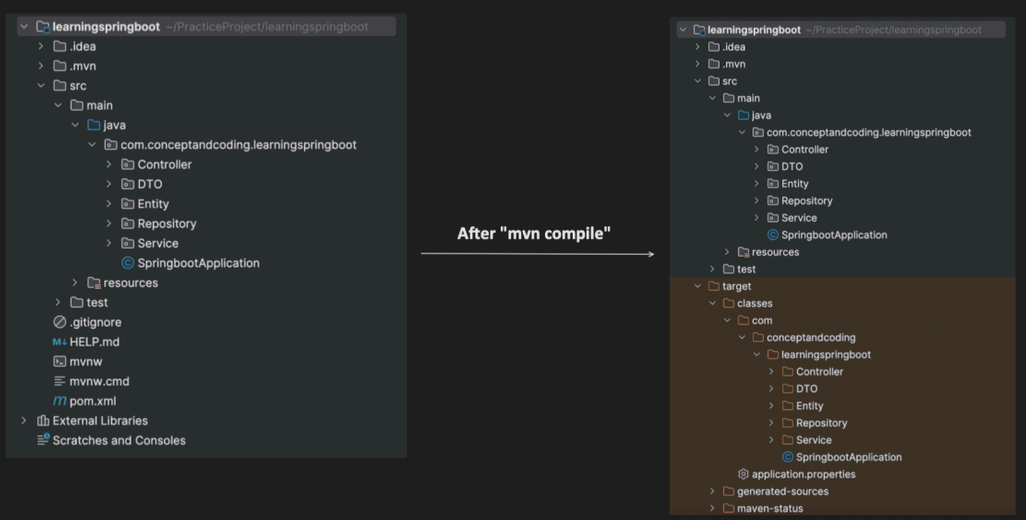
```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-checkstyle-plugin</artifactId>
    <version>3.1.2</version>
    <executions>
        <execution>
            <id>validate-checkstyle</id>
            <phase>validate</phase>
            <goals>
                <goal>check</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <configLocation>myCodeStyle.xml</configLocation>
    </configuration>
</plugin>
```

Compile: **Run the command:** `mvn compile`



It will validate and compile your code and put it under ${project.basedir}/target/classes

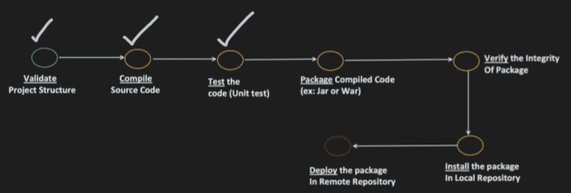
```
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------< com.conceptandcoding:learningspringboot >----------------
[INFO] Building springboot application 0.0.1-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ learningspringboot ---
[INFO] Copying 1 resource from src/main/resources to target/classes
[INFO] Copying 0 resource from src/main/resources to target/classes
[INFO]
[INFO] --- compiler:3.11.0:compile (default-compile) @ learningspringboot ---
[INFO] Changes detected - recompiling the module! :source
[INFO] Compiling 7 source files with javac [debug release 17] to target/classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.812 s
[INFO] Finished at: 2024-03-09T16:35:24+05:30
[INFO] ------------------------------------------------------------------------
```



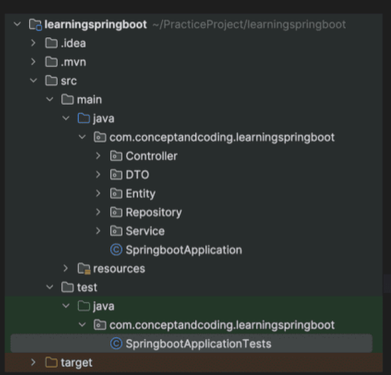
Previously, "Ant" was popular, there we have to tell what to do and also how to do.

```
<project default="compile">
<target name="compile">
//some other properties here
<javac destdir="{provide your destination directory}">
<src>
<pathelement location="src/main/java"/>
</src>
//some other properties here
</javac>
</target>
</project>
```

Test: mvn test



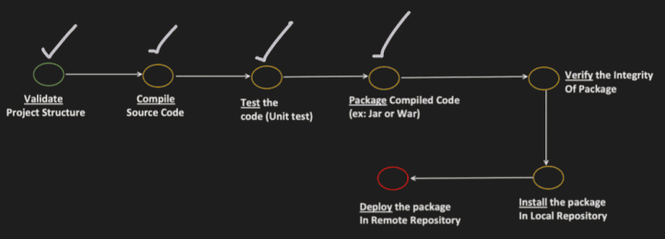
It will validate, compile and then run the TEST cases in your project.




```java
package com.conceptandcoding.learningspringboot;

import ...

@SpringBootTest
class SpringbootApplicationTests {

    @Test
    void contextLoads() {

        System.out.println("CONCEPT && CODING : TEST CASE RUNNING");
    }

}
```

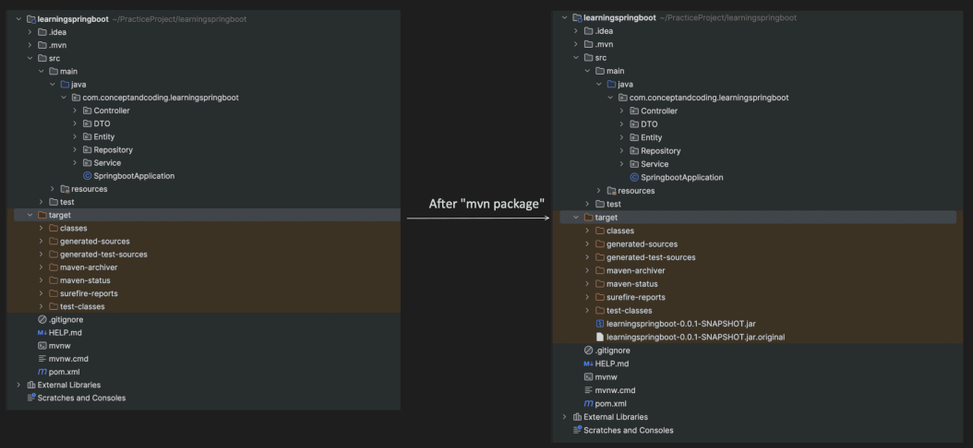
```
[INFO] Scanning for projects...
[INFO]
[INFO] --------------< com.conceptandcoding:learningspringboot >---------------
[INFO] Building springboot application 0.0.1-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ learningspringboot ---
[INFO] Copying 1 resource from src/main/resources to target/classes
[INFO] Copying 0 resource from src/main/resources to target/classes
[INFO]
[INFO] --- compiler:3.11.0:compile (default-compile) @ learningspringboot ---
[INFO] Nothing to compile - all classes are up to date
[INFO] --- surefire:3.1.2:test (default-test) @ learningspringboot ---
[INFO] Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformProvider
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.conceptandcoding.learningspringboot.SpringbootApplicationTests
08:50:22.606 [main] INFO org.springframework.test.context.support.AnnotationConfigContextLoaderUtils --
tApplicationTests does not declare any static, non-private, non-final, nested classes annotated with @Co
08:50:22.655 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper -- Four
ot.SpringbootApplicationTests

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.2.3)

2024-03-10T08:50:22.806+05:30  INFO 37695 --- [           main] c.c.l.SpringbootApplicationTests
gboot)
2024-03-10T08:50:22.806+05:30  INFO 37695 --- [           main] c.c.l.SpringbootApplicationTests
2024-03-10T08:50:23.292+05:30  INFO 37695 --- [           main] c.c.l.SpringbootApplicationTests
WARNING: If a serviceability tool is not in use, please run with -Djdk.instrument.traceUsage for more information
WARNING: Dynamic loading of agents will be disallowed by default in a future release
CONCEPT && CODING : TEST CASE RUNNING
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.235 s -- in com.conceptandcoding.learningspringboot.SpringbootApplicationTests
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.824 s
[INFO] Finished at: 2024-03-10T08:50:23+05:30
[INFO] ------------------------------------------------------------------------
```

Package: mvn package

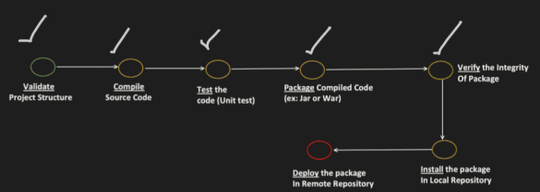


First complete Validate, Compile, Test phase and then run Package phase in which it Generates .jar or .war file.

```
[INFO] Scanning for projects...
[INFO]
[INFO] --------------< com.conceptandcoding:learningspringboot >---------------
[INFO] Building springboot application 0.0.1-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ learningspringboot ---
[INFO] Copying 1 resource from src/main/resources to target/classes
[INFO] Copying 0 resource from src/main/resources to target/classes
[INFO]
[INFO] --- compiler:3.11.0:compile (default-compile) @ learningspringboot ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ learningspringboot ---
[INFO] --- compiler:3.11.0:testCompile (default-testCompile) @ learningspringboot ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- surefire:3.1.2:test (default-test) @ learningspringboot ---
[INFO] Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformPr
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.2.3)

2024-03-10T09:05:27.378+05:30  INFO 38242 --- [           main] c.c.l.SpringbootApplicationTests
gboot)
2024-03-10T09:05:27.378+05:30  INFO 38242 --- [           main] c.c.l.SpringbootApplicationTests
2024-03-10T09:05:27.847+05:30  INFO 38242 --- [           main] c.c.l.SpringbootApplicationTests
OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstra
WARNING: If a serviceability tool is in use, please run with -XX:+EnableDynamicAgentLoading to hide this warning
WARNING: If a serviceability tool is not in use, please run with -Djdk.instrument.traceUsage for more information
WARNING: Dynamic loading of agents will be disallowed by default in a future release
CONCEPT && CODING : TEST CASE RUNNING
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.177 s -- in com.conceptandcoding.learningspringboot.
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- jar:3.3.0:jar (default-jar) @ learningspringboot ---
[INFO] Building jar: /Users/PracticeProject/learningspringboot/target/learningspringboot-0.0.1-SNAPSHOT.jar
```



Verify: mvn verify



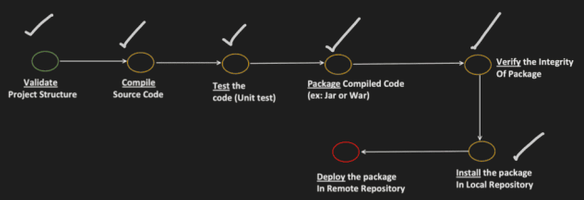
It can perform some additional checks apart from unit test cases like:
- **STATIC CODE ANALYSIS**
- **CHECKSUM VERFICATION etc...**

STATIC CODE ANALYSIS:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-pmd-plugin</artifactId>
    <version>3.21.2</version>
    <executions>
        <execution>
            <id>pmd-analysis</id>
            <phase>verify</phase>
            <goals>
                <goal>pmd</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

PMD is a source code analyzer:
- **Finds unused variable**
- **Finds unused imports**
- **Empty Catch block**
- **No usage of object**
- **Finds duplicate code etc...**

Install: mvn install

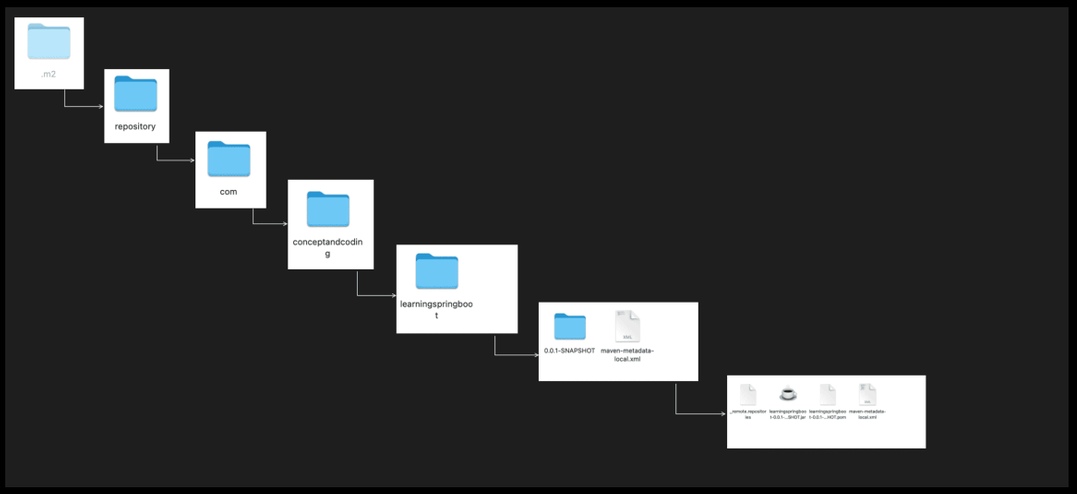


It will install the .jar package in local Maven Repository.
which is typically located in your user home directory (~/.m2/repository)

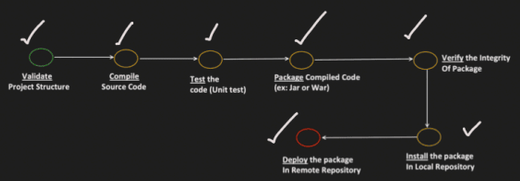
settings.xml (in .m2 folder)

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                              http://maven.apache.org/xsd/settings-1.0.0.xsd">

    <!-- Local Repository: Location where Maven stores downloaded artifacts -->
    <localRepository>${user.home}/.m2/repository</localRepository>


  <!— some other configurations goes here —>

</settings>
```

```
[INFO] Scanning for projects...
[INFO]
[INFO] -------------< com.conceptandcoding:learningspringboot >--------------
[INFO] Building springboot application 0.0.1-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ learningspringboot ---
[INFO] Copying 1 resource from src/main/resources to target/classes
[INFO] Copying 0 resource from src/main/resources to target/classes
[INFO]
[INFO] --- compiler:3.11.0:compile (default-compile) @ learningspringboot ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ learningspringboot ---
[INFO] --- compiler:3.11.0:testCompile (default-testCompile) @ learningspringboot ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- surefire:3.1.2:test (default-test) @ learningspringboot ---
[INFO] Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformPr
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.2.3)

2024-03-10T09:05:27.378+05:30  INFO 38242 --- [           main] c.c.l.SpringbootApplicationTests
gboot]
2024-03-10T09:05:27.378+05:30  INFO 38242 --- [           main] c.c.l.SpringbootApplicationTests
2024-03-10T09:05:27.847+05:30  INFO 38242 --- [           main] c.c.l.SpringbootApplicationTests
OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstra
WARNING: If a serviceability tool is in use, please run with -XX:+EnableDynamicAgentLoading to hide this warning
WARNING: If a serviceability tool is not in use, please run with -Djdk.instrument.traceUsage for more information
WARNING: Dynamic loading of agents will be disallowed by default in a future release
CONCEPT && CODING : TEST CASE RUNNING
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.177 s -- in com.conceptandcoding.learningspringboot.
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- jar:3.3.0:jar (default-jar) @ learningspringboot ---
[INFO] Building jar: /Users/PracticeProject/learningspringboot/target/learningspringboot-0.0.1-SNAPSHOT.jar
[INFO] --- install:3.1.1:install (default-install) @ learningspringboot ---
[INFO] Installing /Users/       /PracticeProject/learningspringboot/pom.xml to      /Users/.m2/repository/com/conceptandcoding/learningspringboot/0.0.1-SNAPSHOT/learningspringboot-0.0.1-SNAPSHOT.pom
[INFO] Installing /Users/       /PracticeProject/learningspringboot/target/learningspringboot-0.0.1-SNAPSHOT.jar to      /Users/.m2/repository/com/conceptandcoding/learningspringboot/0.0.1-SNAPSHOT/learningspringboot-0.0.1-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  4.314 s
[INFO] Finished at: 2024-03-10T10:21:50+05:30
[INFO] ------------------------------------------------------------------------
```



Deploy: mvn deploy



It will deploy the .jar to REMOTE Repository

pom.xml

```
<project>
    <!-- ... other project configurations ... -->

    <distributionManagement>
        <repository>
            <id>remote repository id</id>
            <url>https://remote-repository-url</url>
        </repository>
    </distributionManagement>

    <!-- ... other pom.xml configurations ... -->
</project>
```

settings.xml

```
<settings>
    <!--  other settings configurations  -->

    <servers>
        <server>
            <id>remote repository id</id>
            <username>remote-repository-username</username>
            <password>remote-repository-password</password>
        </server>
    </servers>

    <!--  other settings configurations -->
</settings>
```

If we do not define the remote repository, then MAVEN during "mvn deploy" command we will face below error

```
[INFO] --- deploy:3.1.1:deploy (default-deploy) @ learningspringboot ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  4.587 s
[INFO] Finished at: 2024-03-10T13:00:11+05:30
[INFO] ------------------------------------------------------------------------
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-deploy-plugin:3.1.1:deploy (default-deploy) on project learningspringboot: Deployment failed: repository element was not specified in the POM inside distributionManagement element or
in -DaltDeploymentRepository=id::url parameter -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoExecutionException
```

If we want to deploy the manifest to Maven central repository : https://repo.maven.apache.org/maven2.
Since its public, we do not need username and password in settings.xml

1. @Controller : It indicates that the class is responsible for handling incoming HTTP requests.

```java
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Component
public @interface Controller {
    @AliasFor(
            annotation = Component.class
    )
    String value() default "";
}
```

2. @RestController :

   RestController = Controller + ResponseBody

3. @ResponseBody:
   - Denotes that return value of the controller method should be serialized to HTTP response body.
   - If we do not provide ResponseBody, Spring will consider response as name for the view and tries to resolve and render it (in case we are using the @Controller annotation)

4. @RequestMapping
   1) Value, path (both are same)
   2) Method
   3) Consumes, produces
   4) @Mapping
   5) @Reflective({ControllerMappingReflectiveProcessor.class})

@RequestMapping (path = "/fetchUser", method = RequestMethod.GET, consumes = "application/json", produces = "application/json")

```java
@Target({ElementType.TYPE, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Mapping
@Reflective({ControllerMappingReflectiveProcessor.class})
public @interface RequestMapping {
    String name() default "";

    @AliasFor("path")
    String[] value() default {};

    @AliasFor("value")
    String[] path() default {};

    RequestMethod[] method() default {};

    String[] params() default {};

    String[] headers() default {};

    String[] consumes() default {};

    String[] produces() default {};
}
```

5. @RequestParam: Used to bind, request parameter to controller method parameter.

http://localhost:8080/api/fetchUser?firstName=SHRAYANSH&lastName=JAIN&age=32

```java
@RestController
@RequestMapping(value = "/api/")
public class SampleController {

    @GetMapping(path = "/fetchUser")
    public String getUserDetails(@RequestParam (name = "firstName") String firstName,
                                 @RequestParam (name = "lastName", required = false) String lastName,
                                 @RequestParam(name = "age") int age) {
        return "fetching and returning user details based on first name = " + firstName + " , lastName = " + lastName + " and age is = " + age;
    }
}
```

The framework automatically performs type conversion from the request parameter's string representation to the specified type.

1. Primitive types: Such as int, long, float, double, boolean, etc.
2. Wrapper classes: Such as Integer, Long, Float, Double, Boolean, etc.
3. String: Request parameters are inherently treated as strings only.
4. Enums: You can bind request parameters to enum types.
5. Custom object types: We can do it using a registered PropertyEditor.

How to used PropertyEditor?

```java
@RestController
@RequestMapping(value = "/api/")
public class SampleController {

    @InitBinder
    protected void initBinder(DataBinder binder) {

        binder.registerCustomEditor(String.class, field: "firstName", new FirstNamePropertyEditor());
    }

    @GetMapping(path = "/fetchUser")
    public String getUserDetails(@RequestParam (name = "firstName") String firstName,
                                 @RequestParam (name = "lastName", required = false) String lastName,
                                 @RequestParam(name = "age") int age) {
        return "fetching and returning user details based on first name = " + firstName + " , lastName = " + lastName + " and age is = " + age;
    }
}
```

```java
public class FirstNamePropertyEditor extends PropertyEditorSupport {
    @Override
    public void setAsText(String text) throws IllegalArgumentException {
        setValue(text.trim().toLowerCase());
    }
}
```

6. @PathVariable: Used to extract values from the path of the URL and help to bind it to controller method parameter.

```java
@RestController
@RequestMapping(value = "/api/")
public class SampleController {

    @GetMapping(path = "/fetchUser/{firstName}")
    public String getUserDetails(@PathVariable(value = "firstName") String firstName) {
        return "fetching and returning user details based on first name = " + firstName;
    }
}
```

7. @RequestBody: Bind the body of HTTP request (typically JSON) to controller method parameter (java object).

```java
@RestController
@RequestMapping(value = "/api/")
public class SampleController {


    @PostMapping(path = "/saveUser")
    public String getUserDetails(@RequestBody User user) {
        return "User created " + user.username + ":" + user.email;
    }
}
```

```java
public class User {

    @JsonProperty ("user_name")
    String username;
    String email;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

```
curl --location --request POST 'http://localhost:8080/api/saveUser' \
--header 'Content-Type: application/json' \
--data-raw '{
   "user_name": "Shrayansh",
   "email": "sjxyztest@gmail.com"
}'
```

8. ResponseEntity : It represents the entire HTTP response.

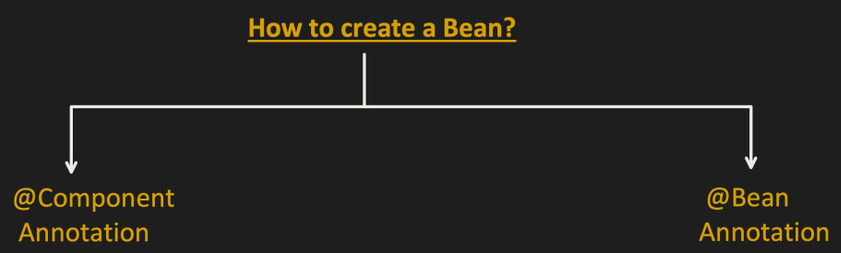
Header, status, response body etc.

```java
@RestController
@RequestMapping(value = "/api/")
public class SampleController {


    @GetMapping(path = "/fetchUser")
    public ResponseEntity<String> getUserDetails(@RequestParam(value = "firstName") String firstName) {
        String output = "fetched User details of " + firstName;
        return ResponseEntity.status(HttpStatus.OK).body(output);
    }
}
```

What is bean?

In a Simple term, Bean is a Java Object, which is managed by Spring container (also known as IOC Container).

IOC container -> contains all the beans which get created and also manage them.



1. Using ***@Component*** Annotation:

- ***@Component*** annotation follows "convention over configuration" approach.
- Means Spring boot will try to auto configure based on conventions reducing the need for explicit configuration.
- @Controller, @Service etc. all are internally tells Spring to create bean and manage it.

So here, Spring boot will internally call **new User()** to create an instance of this class.

```java
@Component
public class User {

    String username;
    String email;

    public String getUsername() { return username; }

    public void setUsername(String username) { this.username = username; }

    public String getEmail() { return email; }

    public void setEmail(String email) { this.email = email; }
}
```

But what will happen to this now:

```java
@Component
public class User {
    String username;
    String email;

    public User(String username, String email){
        this.username =username;
        this.email = email;
    }

    public String getUsername() { return username; }

    public void setUsername(String username) { this.username = username; }

    public String getEmail() { return email; }

    public void setEmail(String email) { this.email = email; }
}
```

```
***************************
APPLICATION FAILED TO START
***************************
```

Why? Because Spring boot does not know what to pass in these Constructor parameters.

So what to do?

@Bean comes into the picture, where we provide the configuration details and tells Spring boot to use it while creating a Bean.

```java
public class User {
    String username;
    String email;

    public User(String username, String email){
        this.username =username;
        this.email = email;
    }

    public String getUsername() { return username; }

    public void setUsername(String username) { this.username = username; }

    public String getEmail() { return email; }

    public void setEmail(String email) { this.email = email; }
}
```

```java
@Configuration
public class AppConfig {

    @Bean
    public User createUserBean(){
        return new User( username: "defaultusername",  email: "defaultemail");
    }
}
```

If we add 2 times in Configuration file, Spring will create 2 beans of it.

```java
@Configuration
public class AppConfig {

    @Bean
    public User createUserBean(){
        return new User( username: "defaultusername",  email: "defaultemail");
    }

    @Bean
    public User createAnotherUserBean(){
        return new User( username: "anotherUsername",  email: "anotheremail");
    }
}
```

Now, we know what is bean and how its getting created. But few Questions comes to our minds:

-> How Spring boot find these Beans?
-> At what time, these beans get created?

How Spring boot find these Beans?

1. Using @ComponentScan annotation, it will scan the specified package and sub-package for classes annotated with @Component, @Service etc.

```java
@SpringBootApplication
@ComponentScan(basePackages = "com.conceptandcoding.learningspringboot")
public class SpringbootApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringbootApplication.class, args);
    }
}
```

2. Through Explicit defining of bean via @Bean annotation in @Configuration class.

```java
@Configuration
public class AppConfig {

    @Bean
    public User createUserBean(){
        return new User( username: "defaultusername",  email: "defaultemail");
    }
}
```

At what time, these beans get created

Eagerness

-> Some Beans get created, When we start up an application.

-> For ex: Beans with Singleton Scope are Eagerly initialized.

Lazy

-> Some Beans get created Lazily, means when they actually needed.

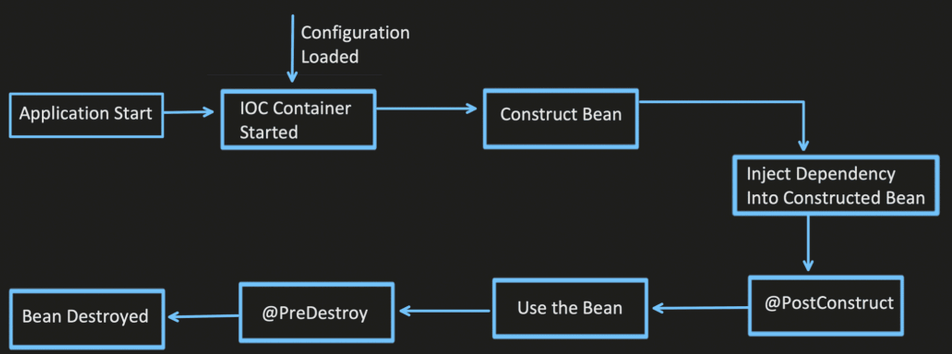
-> For ex: Beans with Scope like Prototype etc. are Lazily initialized.

Or

Beans with @Lazy annotation.

```
@Lazy
@Component
public class Order {

    public Order(){
        System.out.println("initializing Order");
    }
}
```

Lifecycle of Bean:



Step1:

-> During Application Startup, Spring boot invokes IOC Container
(ApplicationContext provides the implementation of IOC container)
-> IOC Container, make use of Configuration and @ComponentScan to look out for the Classes for which beans need to be created.

Invoking IOC Container

```
main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 419 ms
main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 0.771 seconds (process running for 0.946)
```

Step2: Construct the Beans

```java
@Component
public class User {

    public User(){
        System.out.println("initializing user");
    }
}
```

```
2024-04-09T00:47:43.681+05:30  INFO 60692 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2024-04-09T00:47:43.687+05:30  INFO 60692 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-04-09T00:47:43.687+05:30  INFO 60692 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-04-09T00:47:43.711+05:30  INFO 60692 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-04-09T00:47:43.711+05:30  INFO 60692 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 412 ms
initializing user
2024-04-09T00:47:43.859+05:30  INFO 60692 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-04-09T00:47:43.864+05:30  INFO 60692 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 0.756 seconds (process running for 0.927)
```

Step3:

-> Inject the Dependency into the Constructed Bean.
-> @Autowired, first look for a bean of the required type.
-> If bean found, Spring will inject it. Different ways of Injection:

- Constructor Injection
- Setter Injection
- Field Injection

*******We will see each injection and which one to use in next part *********

-> If bean is not found, Spring will create one and then inject it.

```java
@Component
public class User {

    @Autowired
    Order order;

    public User(){
        System.out.println("initializing user");
    }
}
```

```java
@Lazy
@Component
public class Order {

    public Order(){
        System.out.println("Lazy: initializing Order");
    }
}
```

```
2024-04-09T00:53:11.751+05:30  INFO 60872 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2024-04-09T00:53:11.757+05:30  INFO 60872 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-04-09T00:53:11.758+05:30  INFO 60872 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-04-09T00:53:11.782+05:30  INFO 60872 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-04-09T00:53:11.782+05:30  INFO 60872 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 416 ms
initializing user
Lazy: initializing Order
2024-04-09T00:53:11.931+05:30  INFO 60872 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-04-09T00:53:11.936+05:30  INFO 60872 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 0.76 seconds (process running for 0.929)
```

Step4: Perform any task before Bean to be used in application.

```java
@Component
public class User {

    @Autowired
    Order order;

    @PostConstruct
    public void initialize(){
        System.out.println("Bean has been constructed and dependencies have been injected");
    }

    public User(){
        System.out.println("initializing user");
    }
}
```

```
2024-04-09T01:01:14.147+05:30  INFO 61161 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2024-04-09T01:01:14.152+05:30  INFO 61161 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-04-09T01:01:14.153+05:30  INFO 61161 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-04-09T01:01:14.175+05:30  INFO 61161 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-04-09T01:01:14.176+05:30  INFO 61161 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 417 ms
initializing user
Lazy: initializing Order
Bean has been constructed and dependencies have been injected
2024-04-09T01:01:14.318+05:30  INFO 61161 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-04-09T01:01:14.322+05:30  INFO 61161 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 0.754 seconds (process running for 0.923)
```

Step5: Use the Bean in your application.

Means, we are using the bean (or Object) to invoke some methods to perform some business logic

Step6: Perform any task before Bean is getting destroyed.

```java
@SpringBootApplication
public class SpringbootApplication {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(SpringbootApplication.class, args);
        context.close();
    }
}
```

```java
@Component
public class User {

    @PostConstruct
    public void initialize() { System.out.println("Post Construct initiated"); }

    @PreDestroy
    public void preDestroy() { System.out.println("Bean is about to destory, in PreDestoryMethod"); }

    public User() { System.out.println("initializing user"); }
}
```

```
2024-04-09T18:07:37.600+05:30  INFO 16725 --- [           main] c.c.l.SpringbootApplication              : No active profile set, falling back to 1 default profile: "default"
2024-04-09T18:07:38.016+05:30  INFO 16725 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2024-04-09T18:07:38.024+05:30  INFO 16725 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-04-09T18:07:38.024+05:30  INFO 16725 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-04-09T18:07:38.048+05:30  INFO 16725 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-04-09T18:07:38.048+05:30  INFO 16725 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 426 ms
initializing user
Post Construct initiated
2024-04-09T18:07:38.197+05:30  INFO 16725 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-04-09T18:07:38.201+05:30  INFO 16725 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 0.764 seconds (process running for 0.931)
2024-04-09T18:07:49.215+05:30  WARN 16725 --- [           main] org.apache.tomcat.util.net.Acceptor      : The acceptor thread [http-nio-8080-Acceptor] did not stop cleanly
Bean is about to destory, in PreDestoryMethod
```

What is Dependency Injection

- Using Dependency Injection, we can makes our class independent of its dependencies.
- It helps to remove the dependency on concrete implementation and inject the dependencies from external source.

Let's see the Problem first:

```java
public class User {

    Order order = new Order();

    public User(){
        System.out.println("initializing user");
    }
}
```

```java
public class Order {

    public Order(){
        System.out.println("initializing Order");
    }
}
```

Issues with above class structure:

1. Both User and Order class are **Tightly coupled**.
   - Suppose, Order object creation logic gets changed *(lets say in future Object becomes an Interface and it has many concrete class)*, then USER class has to be changed too.

```java
public class User {

    Order order = new Order();

    public User(){
        System.out.println("initializing user");
    }
}
```

new OnlineOrder();

```java
public interface Order {
}
```

```java
public class OnlineOrder implements Order{
}
```

```java
public class OfflineOrder implements Order{
}
```

2. It breaks **Dependency Inversion** rule of S.O.L.I.D principle
   i. This principle says that DO NOT depend on concrete implementation, rather depends on abstraction.

Breaks Dependency Inversion Principle (DIP)

```java
public class User {

    Order order = new OnlineOrder();

    public User(){
        System.out.println("initializing user");
    }
}
```

Follows Dependency Inversion Principle (DIP)

```java
public class User {

    Order order;

    public User(Order orderObj) {
        this.order = orderObj;
    }
}
```

Now in Spring boot how to achieve Dependency Inversion Principle?

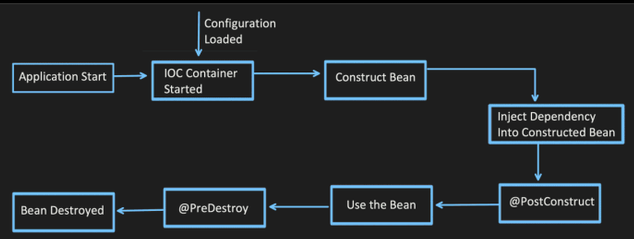
Through Dependency Injection

- Using Dependency Injection, we can makes our class independent of its dependencies.
- It helps to remove the dependency on concrete implementation and inject the dependencies from external source.

```java
@Component
public class User {

    @Autowired
    Order order;

}
```

```java
@Component
public class Order {
}
```

@Autowired, first look for a bean of the required type.
-> If bean found, Spring will inject it.



Different ways of Injection and which one is better?

- Field Injection
- Setter Injection
- Constructor Injection

Field Injection

- Dependency is set into the fields of the class directly.
- Spring uses reflection, it iterates over the fields and resolve the dependency.

```java
@Component
public class User {

    @Autowired
    Order order;

    public User() {
        System.out.println("User initialized");
    }
}
```

```java
@Component
@Lazy
public class Order {

    public Order(){
        System.out.println("order initialized");
    }
}
```

```
2024-04-13T21:33:30.390+05:30  INFO 20786 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2024-04-13T21:33:30.397+05:30  INFO 20786 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-04-13T21:33:30.397+05:30  INFO 20786 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-04-13T21:33:30.425+05:30  INFO 20786 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-04-13T21:33:30.425+05:30  INFO 20786 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 445 ms
User initialized
order initialized
2024-04-13T21:33:30.593+05:30  INFO 20786 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-04-13T21:33:30.598+05:30  INFO 20786 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 0.811 seconds (process running for 0.983)
```

Advantage:

- Very simple and easy to use.

Disadvantage:

- Can not be used with Immutable fields.

```java
@Component
public class User {

    @Autowired
    public final Order order;

    public User() {
        System.out.println("User initialized");
    }
}
```

- Chances of NPE

```java
@Component
public class User {

    @Autowired
    public Order order;

    public User() {
        System.out.println("User initialized");
    }

    public void process(){
        order.process();
    }
}
```

```java
User userObj = new User();
userObj.process();
```

```
Exception in thread "main" java.lang.NullPointerException Create breakpoint :
    at com.conceptandcoding.learningspringboot.User.process(User.java:18)
```

- During Unit Testing, setting MOCK dependency to this field becomes difficult.

```java
@Component
public class User {

    @Autowired
    private Order order;

    public User() {
        System.out.println("User initialized");
    }

    public void process(){
        order.process();
    }
}
```

```java
class UserTest {

private Order orderMockObj;

private User user;

    @BeforeEach
    public void setup(){
        this.orderMockObj = Mockito.mock(Order.class);
        this.user = new User();
    }
}
```

How to set this MOCK, we have to use reflection like @InjectMock annotation internally uses

```java
class UserTest {

@Mock
private Order orderMockObj;

@InjectMocks
private User user;

    @BeforeEach
    public void setup() {
        MockitoAnnotations.initMocks(this);
    }
}
```

Setter Injection

- Dependency is set into the fields using the setter method.
- We have to annotate the method using @Autowired

```java
@Component
public class User {

    public Order order;

    public User() {
        System.out.println("User initialized");
    }

    @Autowired
    public void setOrderDependency(Order order){
        this.order = order;
    }
}
```

```java
@Component
@Lazy
public class Order {

    public Order(){
        System.out.println("order initialized");
    }
}
```

Advantage:

- Dependency can be changed any time after the object creation (as object can not be marked as final).
- Ease of testing, as we can pass mock object in the dependency easily.

Disadvantage:

- Field Can not be marked as final. (We can not make it immutable).

```java
@Component
public class User {

    public final Order order;

    public User() {
        System.out.println("User initialized");
    }

    @Autowired
    public void setOrderDependency(Order order){
        this.order = order;
    }
}
```

- Difficult to read and maintained, as per standard, object should be initialized during object creation, so this might create code readability issue.

Constructor Injection

- Dependency get resolved at the time of initialization of the Object itself.
- Its recommended to use

```java
@Component
public class User {

    Order order;

    @Autowired
    public User(Order order) {
        this.order = order;
        System.out.println("User initialized");
    }

}
```

```java
@Component
@Lazy
public class Order {

    public Order(){
        System.out.println("order initialized");
    }
}
```

```
2024-04-13T21:08:45.923+05:30  INFO 19992 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2024-04-13T21:08:45.929+05:30  INFO 19992 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-04-13T21:08:45.929+05:30  INFO 19992 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-04-13T21:08:45.955+05:30  INFO 19992 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-04-13T21:08:45.955+05:30  INFO 19992 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 422 ms
order initialized
User initialized
2024-04-13T21:08:46.101+05:30  INFO 19992 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-04-13T21:08:46.105+05:30  INFO 19992 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 0.759 seconds (process running for 0.932)
```

When only 1 constructor is present, then using @Autowired on constructor is not mandatory. (from Spring version 4.3)

```java
@Component
public class User {

    Order order;

    public User(Order order ) {
        this.order = order;
        System.out.println("User initialized");
    }
}
```

```java
@Component
@Lazy
public class Order {

    public Order(){
        System.out.println("order initialized");
    }
}
```

When more than 1 constructor is present, then using @Autowired on constructor is mandatory.

```java
@Component
public class User {

    Order order;
    Invoice invoice;

    public User(Order order ) {
        this.order = order;
        System.out.println("User initialized with only Order");
    }

    public User(Invoice invoice) {
        this.invoice = invoice;
        System.out.println("User initialized with only Invoice");
    }
}
```

```java
@Component
@Lazy
public class Invoice {

    public Invoice() { System.out.println("invoice initialized"); }
}
```

```java
@Component
@Lazy
public class Order {

    public Order(){
        System.out.println("order initialized");
    }
}
```

```
Caused by: org.springframework.beans.BeanInstantiationException Create breakpoint : Failed to instantiate [com.conceptandcoding.learningspringboot.User]: No default constructor found
```

```java
@Component
public class User {

    Order order;
    Invoice invoice;

    public User(Order order ) {
        this.order = order;
        System.out.println("User initialized with only Order");
    }

    @Autowired
    public User(Invoice invoice) {
        this.invoice = invoice;
        System.out.println("User initialized with only Invoice");
    }
}
```

```java
@Component
@Lazy
public class Invoice {

    public Invoice() { System.out.println("invoice initialized"); }
}
```

```java
@Component
@Lazy
public class Order {

    public Order(){
        System.out.println("order initialized");
    }
}
```

```
2024-04-13T21:16:01.654+05:30  INFO 20242 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-04-13T21:16:01.654+05:30  INFO 20242 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-04-13T21:16:01.676+05:30  INFO 20242 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-04-13T21:16:01.677+05:30  INFO 20242 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 430 ms
invoice initialized
User initialized with only Invoice
2024-04-13T21:16:01.827+05:30  INFO 20242 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-04-13T21:16:01.832+05:30  INFO 20242 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 0.775 seconds (process running for 0.94)
```

Why Constructor Injection is Recommended (Advantages):

1. All mandatory dependencies are created at the time of initialization itself. Makes 100% sure that our object is fully initialized with mandatory dependency
   i. avoid NPE during runtime
   ii. Unnecessary null checks can be avoided too.

2. We can create immutable object using Constructor injection.

```java
@Component
public class User {

    public final Order order ;

    @Autowired
    public User(Order order) {
        this.order = order;
        System.out.println("User initialized");
    }
}
```

```java
@Component
public class User {

    @Autowired
    public final Order order ;

    public User() {
        System.out.println("User initialized");
    }
}
```

3. Fail Fast: If there is any missing dependency, it will fail during compilation itself, rather than failing during run Time.

```java
@Component
public class User {

    public Order order;

    public User() {
        System.out.println("User initialized");
    }

    @PostConstruct
    public void init(){
        System.out.println(order == null);
    }
}
```

```java
@Component
public class Order {

    public Order(){
        System.out.println("order initialized");
    }
}
```

Using Constructor Injection, even if we missed @Autowired

```java
@Component
public class User {

    public Order order;

    public User(Order order) {
        this.order = order;
        System.out.println("User initialized");
    }

    @PostConstruct
    public void init(){
        System.out.println(order == null);
    }
}
```

OR

(it will fail fast, if Order bean is missing)

```java
@Component
public class User {

    public Order order;

    public User(Order order) {
        this.order = order;
        System.out.println("User initialized");
    }

    @PostConstruct
    public void init(){
        System.out.println(order == null);
    }
}
```

```java
public class Order {

    public Order(){
        System.out.println("order initialized");
    }
}
```

```
***************************
APPLICATION FAILED TO START
***************************

Description:

Parameter 0 of constructor in com.conceptandcoding.learningspringboot.User required a bean of type 'com.conceptandcoding.learningspringboot.Order' that could not be found.
```

4. Unit testing is easy.

```java
@Component
public class User {

    private Order order;

    @Autowired
    public User(Order order) {
        this.order = order;
        System.out.println("User initialized");
    }

    public void process(){
        order.process();
    }
}
```

```java
class UserTest {

private Order orderMockObj;

private User user;

    @BeforeEach
    public void setup(){
        this.orderMockObj = Mockito.mock(Order.class);
        this.user = new User(orderMockObj);
    }
}
```

Common Issues when dealing with Dependency Injection:

1. CIRCULAR DEPENDENCY

```java
@Component
public class Order {

    @Autowired
    Invoice invoice;

    public Order() {
        System.out.println("order initialized");
    }
}
```

```java
@Component
public class Invoice {

    @Autowired
    Order order;

    public Invoice() {
        System.out.println("invoice initialized");
    }
}
```

```
***************************
APPLICATION FAILED TO START
***************************

Description:

The dependencies of some of the beans in the application context form a cycle:

┌─────┐
|  invoice
↑     ↓
|  order
└─────┘
```

Solutions:

1. First and foremost, can we refactor the code and remove this cycle dependency:

For example, common code in which both are dependent, can be taken out to separate class. This way we can break the circular dependency.

2. Using @Lazy on @Autowired annotation .

Spring will create proxy bean instead of creating the bean instance immediately during application startup.

@Lazy on field Injection

Let's first consider this

```java
@Component
@Lazy
public class Order {

    public Order() {
        System.out.println("Order initialized");
    }
}
```

```java
@Component
public class Invoice {

    @Autowired
    public Order order;

    public Invoice() {
        System.out.println("Invoice initialized");
    }
}
```

```
2024-04-14T18:21:55.967+05:30  INFO 48155 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-04-14T18:21:55.967+05:30  INFO 48155 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-04-14T18:21:55.993+05:30  INFO 48155 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-04-14T18:21:55.993+05:30  INFO 48155 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 438 ms
Invoice initialized
Order initialized
2024-04-14T18:21:56.141+05:30  INFO 48155 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-04-14T18:21:56.146+05:30  INFO 48155 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 0.778 seconds (process running for 0.967)
```

Now, lets see this:

```java
@Component
@Lazy
public class Order {

    public Order() {
        System.out.println("Order initialized");
    }
}
```

```java
@Component
public class Invoice {

    @Lazy
    @Autowired
    public Order order;

    public Invoice() {
        System.out.println("Invoice initialized");
    }
}
```

```
2024-04-14T18:24:03.474+05:30  INFO 48250 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2024-04-14T18:24:03.482+05:30  INFO 48250 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-04-14T18:24:03.482+05:30  INFO 48250 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-04-14T18:24:03.507+05:30  INFO 48250 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-04-14T18:24:03.507+05:30  INFO 48250 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 426 ms
Invoice initialized
2024-04-14T18:24:03.677+05:30  INFO 48250 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-04-14T18:24:03.683+05:30  INFO 48250 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 0.792 seconds (process running for 0.96)
```

Now, We can use this @Lazy to resolve the circular dependency

```java
@Component
public class Order {

    @Autowired
    Invoice invoice;

    public Order() {
        System.out.println("order initialized");
    }
}
```

```java
@Component
public class Invoice {

    @Lazy
    @Autowired
    public Order order;

    public Invoice() {
        System.out.println("Invoice initialized");
    }
}
```

```
2024-04-14T18:27:04.567+05:30  INFO 48425 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-04-14T18:27:04.568+05:30  INFO 48425 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-04-14T18:27:04.591+05:30  INFO 48425 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-04-14T18:27:04.592+05:30  INFO 48425 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 432 ms
Invoice initialized
Order initialized
2024-04-14T18:27:04.745+05:30  INFO 48425 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-04-14T18:27:04.750+05:30  INFO 48425 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 0.784 seconds (process running for 0.958)
```

3. Using @PostConstruct

```java
@Component
public class Order {

    @Autowired
    Invoice invoice;

    public Order() {
        System.out.println("Order initialized");
    }

    @PostConstruct
    public void initialize(){
        invoice.setOrder(this);
    }
}
```

```java
@Component
public class Invoice {

    public Order order;

    public Invoice() {
        System.out.println("Invoice initialized");
    }

    public void setOrder(Order order) {
        this.order = order;
    }
}
```

```
2024-04-14T18:27:04.567+05:30  INFO 48425 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-04-14T18:27:04.568+05:30  INFO 48425 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-04-14T18:27:04.591+05:30  INFO 48425 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-04-14T18:27:04.592+05:30  INFO 48425 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 432 ms
Invoice initialized
Order initialized
2024-04-14T18:27:04.745+05:30  INFO 48425 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-04-14T18:27:04.750+05:30  INFO 48425 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 0.784 seconds (process running for 0.958)
```

UNSATISFIED DEPENDENCY

Problem:

```java
@Component
public class User {

    @Autowired
    Order order;

    public User() {
        System.out.println("User initialized");
    }
}
```

```java
public interface Order {
}
```

```java
@Component
public class OnlineOrder implements Order {

    public OnlineOrder() {
        System.out.println("Online order initialized");
    }
}
```

```java
@Component
public class OfflineOrder implements Order{

    public OfflineOrder() {
        System.out.println("Offline order initialized");
    }
}
```

```
***************************
APPLICATION FAILED TO START
***************************
```

```
UnsatisfiedDependencyException: Error creating bean with name 'user'
```

Solution:

1. @Primary annotation

```java
@Component
public class User {

    @Autowired
    Order order;

    public User() {
        System.out.println("User initialized");
    }
}
```

```java
public interface Order {
}
```

```java
@Primary
@Component
public class OnlineOrder implements Order {

    public OnlineOrder() {
        System.out.println("Online order initialized");
    }
}
```

```java
@Component
public class OfflineOrder implements Order{

    public OfflineOrder() {
        System.out.println("Offline order initialized");
    }
}
```

```
2024-04-14T19:09:38.705+05:30  INFO 51207 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-04-14T19:09:38.729+05:30  INFO 51207 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-04-14T19:09:38.729+05:30  INFO 51207 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 417 ms
Offline order initialized
Online order initialized
User initialized
2024-04-14T19:09:38.882+05:30  INFO 51207 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-04-14T19:09:38.887+05:30  INFO 51207 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 0.763 seconds (process running for 0.927)
```

2. @Qualifier annotation

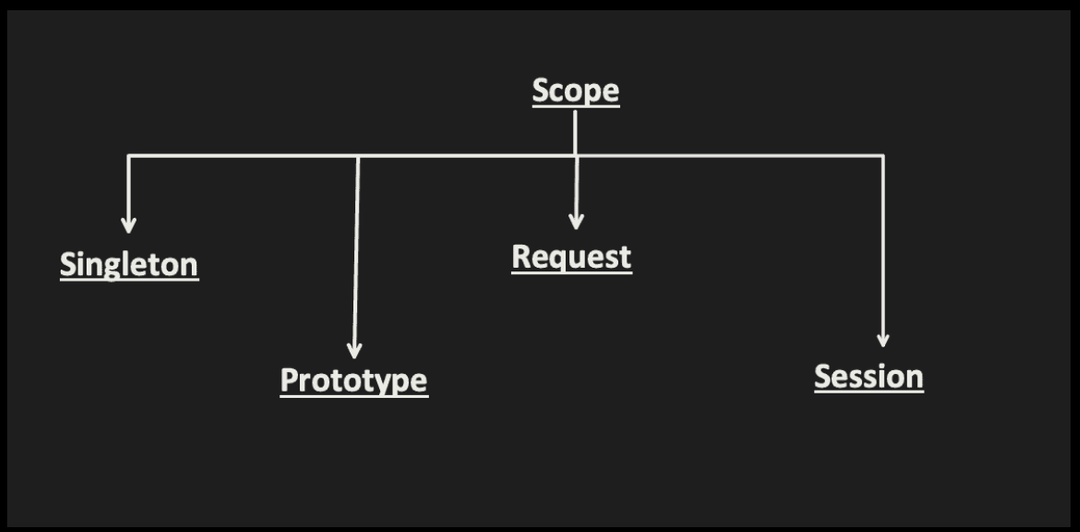
```java
@Component
public class User {

    @Qualifier("offlineOrderName")
    @Autowired
    Order order;

    public User() {
        System.out.println("User initialized");
    }
}
```

```java
public interface Order {
}
```

```java
@Component
@Qualifier("onlineOrderName")
public class OnlineOrder implements Order {

    public OnlineOrder() {
        System.out.println("Online order initialized");
    }
}
```

```java
@Component
@Qualifier("offlineOrderName")
public class OfflineOrder implements Order{

    public OfflineOrder() {
        System.out.println("Offline order initialized");
    }
}
```

```
2024-04-14T19:16:15.633+05:30  INFO 51489 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-04-14T19:16:15.633+05:30  INFO 51489 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-04-14T19:16:15.657+05:30  INFO 51489 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-04-14T19:16:15.657+05:30  INFO 51489 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 419 ms
Offline order initialized
Online order initialized
User initialized
2024-04-14T19:16:15.807+05:30  INFO 51489 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-04-14T19:16:15.813+05:30  INFO 51489 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 0.763 seconds (process running for 0.932)
```



Singleton:

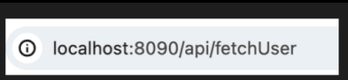
- Default scope
- Only 1 instance created per IOC.
- Eagerly initialized by IOC (means at the time of application startup, object get created)

```java
@RestController
@RequestMapping(value = "/api/")
public class TestController1 {

    @Autowired
    User user;

    public TestController1(){
        System.out.println("TestController1 instance initialization" );
    }

    @PostConstruct
    public void init(){
        System.out.println("TestController1 object hashCode: " + this.hashCode() +
                " User object hashCode: " + user.hashCode());
    }

    @GetMapping(path = "/fetchUser")
    public ResponseEntity<String> getUserDetails() {
        System.out.println("fetchUser api invoked");
        return ResponseEntity.status(HttpStatus.OK).body("");
    }
}
```

```java
@RestController
@RequestMapping(value = "/api/")
@Scope(value = ConfigurableBeanFactory.SCOPE_SINGLETON)
public class TestController2 {

    @Autowired
    User user;

    public TestController2() {
        System.out.println("TestController2 instance initialization");
    }

    @PostConstruct
    public void init(){
        System.out.println("TestController2 object hashCode: " + this.hashCode() +
                " User object hashCode: " + user.hashCode());
    }

    @GetMapping(path = "/fetchUser2")
    public ResponseEntity<String> getUserDetails() {
        System.out.println("fetchUser2 api invoked");
        return ResponseEntity.status(HttpStatus.OK).body("");
    }
}
```

```java
@Component
@Scope("singleton")
public class User {

    public User() {
        System.out.println("User initialization");
    }

    @PostConstruct
    public void init(){
        System.out.println("User object hashCode: " + this.hashCode());
    }
}
```

```
2024-05-01T13:12:47.754+05:30  INFO 36566 --- [           main] c.c.l.SpringbootApplication
2024-05-01T13:12:47.755+05:30  INFO 36566 --- [           main] c.c.l.SpringbootApplication
2024-05-01T13:12:48.182+05:30  INFO 36566 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-01T13:12:48.188+05:30  INFO 36566 --- [           main] o.apache.catalina.core.StandardService
2024-05-01T13:12:48.188+05:30  INFO 36566 --- [           main] o.apache.catalina.core.StandardEngine
2024-05-01T13:12:48.213+05:30  INFO 36566 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-01T13:12:48.213+05:30  INFO 36566 --- [           main] w.s.c.ServletWebServerApplicationContext
TestController1 instance initialization
User initialization
User object hashCode: 1140202235
TestController1 object hashCode: 1046302571 User object hashCode: 1140202235
TestController2 instance initialization
TestController2 object hashCode: 1525241607 User object hashCode: 1140202235
2024-05-01T13:12:48.365+05:30  INFO 36566 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-01T13:12:48.369+05:30  INFO 36566 --- [           main] c.c.l.SpringbootApplication
2024-05-01T13:13:50.207+05:30  INFO 36566 --- [nio-8090-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-01T13:13:50.207+05:30  INFO 36566 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet
2024-05-01T13:13:50.208+05:30  INFO 36566 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet
```




```
2024-05-01T13:12:47.754+05:30  INFO 36566 --- [           main] c.c.l.SpringbootApplication
2024-05-01T13:12:47.755+05:30  INFO 36566 --- [           main] c.c.l.SpringbootApplication
2024-05-01T13:12:48.182+05:30  INFO 36566 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-01T13:12:48.188+05:30  INFO 36566 --- [           main] o.apache.catalina.core.StandardService
2024-05-01T13:12:48.188+05:30  INFO 36566 --- [           main] o.apache.catalina.core.StandardEngine
2024-05-01T13:12:48.213+05:30  INFO 36566 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-01T13:12:48.213+05:30  INFO 36566 --- [           main] w.s.c.ServletWebServerApplicationContext
TestController1 instance initialization
User initialization
User object hashCode: 1140202235
TestController1 object hashCode: 1046302571 User object hashCode: 1140202235
TestController2 instance initialization
TestController2 object hashCode: 1525241607 User object hashCode: 1140202235
2024-05-01T13:12:48.365+05:30  INFO 36566 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-01T13:12:48.369+05:30  INFO 36566 --- [           main] c.c.l.SpringbootApplication
2024-05-01T13:13:50.207+05:30  INFO 36566 --- [nio-8090-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-01T13:13:50.207+05:30  INFO 36566 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet
2024-05-01T13:13:50.208+05:30  INFO 36566 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet
fetchUser api invoked
```

Prototype:

- Each time new Object is created.
- Its Lazily Initialized, means when object is created only when its required.

```java
@RestController
@Scope(ConfigurableBeanFactory.SCOPE_PROTOTYPE)
@RequestMapping(value = "/api/")
public class TestController1 {

    @Autowired
    User user;

    @Autowired
    Student student;

    public TestController1() {
        System.out.println("TestController1 instance initialization");
    }

    @PostConstruct
    public void init() {
        System.out.println("TestController1 object hashCode: " + this.hashCode() +
                " User object hashCode: " + user.hashCode()
                + " Student object hashCode: " + student.hashCode());
    }

    @GetMapping(path = "/fetchUser")
    public ResponseEntity<String> getUserDetails() {
        System.out.println("fetchUser api invoked");
        return ResponseEntity.status(HttpStatus.OK).body("");
    }
}
```

```java
@Component
public class Student {

    @Autowired
    User user;

    public Student() {
        System.out.println("Student instance initialization");
    }

    @PostConstruct
    public void init(){
        System.out.println("Student object hashCode: " + this.hashCode() +
                " User object hashCode: " + user.hashCode());
    }

}
```

```java
@Component
@Scope("prototype")
public class User {

    public User() {
        System.out.println("User initialization");
    }

    @PostConstruct
    public void init(){
        System.out.println("User object hashCode: " + this.hashCode());
    }
}
```

```
2024-05-01T13:32:00.689+05:30  INFO 37722 --- [           main] c.c.l.SpringbootApplication
2024-05-01T13:32:00.691+05:30  INFO 37722 --- [           main] c.c.l.SpringbootApplication
2024-05-01T13:32:01.130+05:30  INFO 37722 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-01T13:32:01.136+05:30  INFO 37722 --- [           main] o.apache.catalina.core.StandardService
2024-05-01T13:32:01.136+05:30  INFO 37722 --- [           main] o.apache.catalina.core.StandardEngine
2024-05-01T13:32:01.160+05:30  INFO 37722 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-01T13:32:01.160+05:30  INFO 37722 --- [           main] w.s.c.ServletWebServerApplicationContext
Student instance initialization
User initialization
User object hashCode: 1510009630
Student object hashCode: 2092450685 User object hashCode: 1510009630
2024-05-01T13:32:01.328+05:30  INFO 37722 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-01T13:32:01.333+05:30  INFO 37722 --- [           main] c.c.l.SpringbootApplication
```

localhost:8090/api/fetchUser

```
2024-05-01T13:32:00.689+05:30  INFO 37722 --- [           main] c.c.l.SpringbootApplication              : Starting
2024-05-01T13:32:00.691+05:30  INFO 37722 --- [           main] c.c.l.SpringbootApplication              : No activ
2024-05-01T13:32:01.130+05:30  INFO 37722 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat i
2024-05-01T13:32:01.136+05:30  INFO 37722 --- [           main] o.apache.catalina.core.StandardService   : Starting
2024-05-01T13:32:01.136+05:30  INFO 37722 --- [           main] o.apache.catalina.core.StandardEngine    : Starting
2024-05-01T13:32:01.160+05:30  INFO 37722 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initiali
2024-05-01T13:32:01.160+05:30  INFO 37722 --- [           main] w.s.c.ServletWebServerApplicationContext : Root Web
Student instance initialization
User initialization
User object hashCode: 1510009630
Student object hashCode: 2092450685 User object hashCode: 1510009630
2024-05-01T13:32:01.328+05:30  INFO 37722 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat s
2024-05-01T13:32:01.333+05:30  INFO 37722 --- [           main] c.c.l.SpringbootApplication              : Started
2024-05-01T13:32:50.142+05:30  INFO 37722 --- [nio-8090-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initiali
2024-05-01T13:32:50.142+05:30  INFO 37722 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet        : Initiali
2024-05-01T13:32:50.143+05:30  INFO 37722 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet        : Complete
TestController1 instance initialization
User initialization
User object hashCode: 1984730322
TestController1 object hashCode: 1786739287 User object hashCode: 1984730322 Student object hashCode: 2092450685
fetchUser api invoked
```

Request:

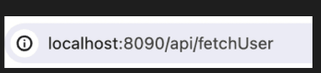
- New Object is created for each HTTP request.
- Lazily initialized.

```java
@RestController
@Scope("request")
@RequestMapping(value = "/api/")
public class TestController1 {

    @Autowired
    User user;

    @Autowired
    Student student;

    public TestController1() {
        System.out.println("TestController1 instance initialization");
    }

    @PostConstruct
    public void init() {
        System.out.println("TestController1 object hashCode: " + this.hashCode() +
                " User object hashCode: " + user.hashCode()
                + " Student object hashCode: " + student.hashCode());
    }

    @GetMapping(path = "/fetchUser")
    public ResponseEntity<String> getUserDetails() {
        System.out.println("fetchUser api invoked");
        return ResponseEntity.status(HttpStatus.OK).body("");
    }
}
```

```java
@Component
@Scope("prototype")
public class Student {

    @Autowired
    User user;

    public Student() {
        System.out.println("Student instance initialization");
    }

    @PostConstruct
    public void init(){
        System.out.println("Student object hashCode: " + this.hashCode() +
                " User object hashCode: " + user.hashCode());
    }

}
```

```java
@Component
@Scope("request")
public class User {

    public User() {
        System.out.println("User initialization");
    }

    @PostConstruct
    public void init(){
        System.out.println("User object hashCode: " + this.hashCode());
    }
}
```

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.2.3)

2024-05-01T14:13:02.583+05:30  INFO 40344 --- [           main] c.c.l.SpringbootApplication              : Starting SpringbootApplication using Java 21.0.
2024-05-01T14:13:02.584+05:30  INFO 40344 --- [           main] c.c.l.SpringbootApplication              : No active profile set, falling back to 1 defaul
2024-05-01T14:13:03.026+05:30  INFO 40344 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8090 (http)
2024-05-01T14:13:03.033+05:30  INFO 40344 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-05-01T14:13:03.033+05:30  INFO 40344 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19
2024-05-01T14:13:03.059+05:30  INFO 40344 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationCont
2024-05-01T14:13:03.059+05:30  INFO 40344 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization comp
2024-05-01T14:13:03.205+05:30  INFO 40344 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8090 (http) with context
2024-05-01T14:13:03.209+05:30  INFO 40344 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 0.795 seconds
```




```
2024-05-01T14:13:03.033+05:30  INFO 40344 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine:
2024-05-01T14:13:03.059+05:30  INFO 40344 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embed
2024-05-01T14:13:03.059+05:30  INFO 40344 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContex
2024-05-01T14:13:03.205+05:30  INFO 40344 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 80
2024-05-01T14:13:03.209+05:30  INFO 40344 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplica
2024-05-01T14:14:05.931+05:30  INFO 40344 --- [nio-8090-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring Dispa
2024-05-01T14:14:05.931+05:30  INFO 40344 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dis
2024-05-01T14:14:05.932+05:30  INFO 40344 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization
TestController1 instance initialization
User initialization
User object hashCode: 39793904
Student instance initialization
Student object hashCode: 275139209 User object hashCode: 39793904
TestController1 object hashCode: 898967761 User object hashCode: 39793904 Student object hashCode: 275139209
fetchUser api invoked
```

localhost:8090/api/fetchUser

```
2024-05-01T14:13:03.205+05:30  INFO 40344 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started o
2024-05-01T14:13:03.209+05:30  INFO 40344 --- [           main] c.c.l.SpringbootApplication              : Started Springbo
2024-05-01T14:14:05.931+05:30  INFO 40344 --- [nio-8090-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spr
2024-05-01T14:14:05.931+05:30  INFO 40344 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Ser
2024-05-01T14:14:05.932+05:30  INFO 40344 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initia
TestController1 instance initialization
User initialization
User object hashCode: 39793904
Student instance initialization
Student object hashCode: 275139209 User object hashCode: 39793904
TestController1 object hashCode: 898967761 User object hashCode: 39793904 Student object hashCode: 275139209
fetchUser api invoked
TestController1 instance initialization
User initialization
User object hashCode: 1227388929
Student instance initialization
Student object hashCode: 1206886228 User object hashCode: 1227388929
TestController1 object hashCode: 1137709937 User object hashCode: 1227388929 Student object hashCode: 1206886228
fetchUser api invoked
```

Consider below, What will happen?

```java
@RestController
@Scope("singleton")
@RequestMapping(value = "/api/")
public class TestController1 {

    @Autowired
    User user;

    public TestController1() {
        System.out.println("TestController1 instance initialization");
    }

    @PostConstruct
    public void init() {
        System.out.println("TestController1 object hashCode: " + this.hashCode() +
                " User object hashCode: " + user.hashCode());
    }

    @GetMapping(path = "/fetchUser")
    public ResponseEntity<String> getUserDetails() {
        System.out.println("fetchUser api invoked");
        return ResponseEntity.status(HttpStatus.OK).body("");
    }
}
```



```java
@Component
@Scope("request")
public class User {

    public User() {
        System.out.println("User initialization");
    }

    @PostConstruct
    public void init(){
        System.out.println("User object hashCode: " + this.hashCode());
    }
}
```

```
2024-05-01T14:19:25.563+05:30  INFO 40699 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-05-01T14:19:25.563+05:30  INFO 40699 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-05-01T14:19:25.590+05:30  INFO 40699 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-05-01T14:19:25.591+05:30  INFO 40699 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 44
TestController1 instance initialization
2024-05-01T14:19:25.617+05:30  WARN 40699 --- [           main] ConfigServletWebServerApplicationContext : Exception encountered during context initialization - canc
2024-05-01T14:19:25.619+05:30  INFO 40699 --- [           main] o.apache.catalina.core.StandardService   : Stopping service [Tomcat]
2024-05-01T14:19:25.626+05:30  INFO 40699 --- [           main] .s.b.a.l.ConditionEvaluationReportLogger :

Error starting ApplicationContext. To display the condition evaluation report re-run your application with 'debug' enabled.
2024-05-01T14:19:25.635+05:30 ERROR 40699 --- [           main] o.s.boot.SpringApplication               : Application run failed

org.springframework.beans.factory.UnsatisfiedDependencyException Create breakpoint : Error creating bean with name 'testController1': Unsatisfied dependency expressed thr
```

Spring creates Request scope bean only when there is active HTTP request present. Since Singleton, is eagerly initialized, there is no active HTTP request present in current thread. So it wont create a bean for User.

```java
@RestController
@Scope("singleton")
@RequestMapping(value = "/api/")
public class TestController1 {

    @Autowired
    User user;

    public TestController1() {
        System.out.println("TestController1 instance initialization");
    }

    @PostConstruct
    public void init() {
        System.out.println("TestController1 object hashCode: " + this.hashCode() +
                " User object hashCode: " + user.hashCode());
    }

    @GetMapping(path = "/fetchUser")
    public ResponseEntity<String> getUserDetails() {
        System.out.println("fetchUser api invoked");
        return ResponseEntity.status(HttpStatus.OK).body("");
    }
}
```

```java
@Component
@Scope(value = "request", proxyMode = ScopedProxyMode.TARGET_CLASS)
public class User {

    public User() {
        System.out.println("User initialization");
    }

    @PostConstruct
    public void init(){
        System.out.println("User object hashCode: " + this.hashCode());
    }

    public void dummyMethod(){

    }
}
```

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.2.3)

2024-05-01T15:16:24.864+05:30  INFO 43839 --- [           main] c.c.l.SpringbootApplication
2024-05-01T15:16:24.865+05:30  INFO 43839 --- [           main] c.c.l.SpringbootApplication
2024-05-01T15:16:25.318+05:30  INFO 43839 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-01T15:16:25.325+05:30  INFO 43839 --- [           main] o.apache.catalina.core.StandardService
2024-05-01T15:16:25.325+05:30  INFO 43839 --- [           main] o.apache.catalina.core.StandardEngine
2024-05-01T15:16:25.351+05:30  INFO 43839 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-01T15:16:25.352+05:30  INFO 43839 --- [           main] w.s.c.ServletWebServerApplicationContext
TestController1 instance initialization
TestController1 object hashCode: 1356419559 User object hashCode: 1159352444
2024-05-01T15:16:25.534+05:30  INFO 43839 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-01T15:16:25.539+05:30  INFO 43839 --- [           main] c.c.l.SpringbootApplication
```

localhost:8090/api/fetchUser

```
2024-05-01T15:16:24.864+05:30  INFO 43839 --- [           main] c.c.l.SpringbootApplication
2024-05-01T15:16:24.865+05:30  INFO 43839 --- [           main] c.c.l.SpringbootApplication
2024-05-01T15:16:25.318+05:30  INFO 43839 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-01T15:16:25.325+05:30  INFO 43839 --- [           main] o.apache.catalina.core.StandardService
2024-05-01T15:16:25.325+05:30  INFO 43839 --- [           main] o.apache.catalina.core.StandardEngine
2024-05-01T15:16:25.351+05:30  INFO 43839 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-01T15:16:25.352+05:30  INFO 43839 --- [           main] w.s.c.ServletWebServerApplicationContext
TestController1 instance initialization
TestController1 object hashCode: 1356419559 User object hashCode: 1159352444
2024-05-01T15:16:25.534+05:30  INFO 43839 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-01T15:16:25.539+05:30  INFO 43839 --- [           main] c.c.l.SpringbootApplication
2024-05-01T15:16:50.198+05:30  INFO 43839 --- [nio-8090-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-01T15:16:50.198+05:30  INFO 43839 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet
2024-05-01T15:16:50.199+05:30  INFO 43839 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet
fetchUser api invoked
User initialization
User object hashCode: 1078757370
```

Session:

- New Object is created for each HTTP session.
- Lazily initialized.
- When user accesses any endpoint, session is created.
- Remains active, till it does not expires.

```java
@RestController
@Scope(value = "session")
@RequestMapping(value = "/api/")
public class TestController1 {

    @Autowired
    User user;

    public TestController1() {
        System.out.println("TestController1 instance initialization");
    }

    @PostConstruct
    public void init() {
        System.out.println("TestController1 object hashCode: " + this.hashCode() +
                " User object hashCode: " + user.hashCode());
    }

    @GetMapping(path = "/fetchUser")
    public ResponseEntity<String> getUserDetails() {
        System.out.println("fetchUser api invoked");
        return ResponseEntity.status(HttpStatus.OK).body("");
    }

    @GetMapping(path = "/logout")
    public ResponseEntity<String> getUserDetails(HttpServletRequest request) {
        System.out.println("end the session");
        HttpSession session = request.getSession();
        session.invalidate();
        return ResponseEntity.status(HttpStatus.OK).body("");
    }
}
```

```java
@Component
public class User {

    public User() {
        System.out.println("User initialization");
    }

    @PostConstruct
    public void init(){
        System.out.println("User object hashCode: " + this.hashCode());
    }

    public void dummyMethod(){

    }
}
```

```
2024-05-01T15:37:03.271+05:30  INFO 46071 --- [           main] c.c.l.SpringbootApplication
2024-05-01T15:37:03.272+05:30  INFO 46071 --- [           main] c.c.l.SpringbootApplication
2024-05-01T15:37:03.710+05:30  INFO 46071 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-01T15:37:03.716+05:30  INFO 46071 --- [           main] o.apache.catalina.core.StandardService
2024-05-01T15:37:03.716+05:30  INFO 46071 --- [           main] o.apache.catalina.core.StandardEngine
2024-05-01T15:37:03.739+05:30  INFO 46071 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-01T15:37:03.740+05:30  INFO 46071 --- [           main] w.s.c.ServletWebServerApplicationContext
User initialization
User object hashCode: 254812619
2024-05-01T15:37:03.920+05:30  INFO 46071 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-01T15:37:03.926+05:30  INFO 46071 --- [           main] c.c.l.SpringbootApplication
```

1. localhost:8090/api/fetchUser

```
2024-05-01T15:37:03.271+05:30  INFO 46071 --- [           main] c.c.l.SpringbootApplication
2024-05-01T15:37:03.272+05:30  INFO 46071 --- [           main] c.c.l.SpringbootApplication
2024-05-01T15:37:03.710+05:30  INFO 46071 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-01T15:37:03.716+05:30  INFO 46071 --- [           main] o.apache.catalina.core.StandardService
2024-05-01T15:37:03.716+05:30  INFO 46071 --- [           main] o.apache.catalina.core.StandardEngine
2024-05-01T15:37:03.739+05:30  INFO 46071 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-01T15:37:03.740+05:30  INFO 46071 --- [           main] w.s.c.ServletWebServerApplicationContext
User initialization
User object hashCode: 254812619
2024-05-01T15:37:03.920+05:30  INFO 46071 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-01T15:37:03.926+05:30  INFO 46071 --- [           main] c.c.l.SpringbootApplication
2024-05-01T15:37:26.050+05:30  INFO 46071 --- [nio-8090-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-01T15:37:26.051+05:30  INFO 46071 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet
2024-05-01T15:37:26.051+05:30  INFO 46071 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet
TestController1 instance initialization
TestController1 object hashCode: 1476370807 User object hashCode: 254812619
fetchUser api invoked
```

2. localhost:8090/api/fetchUser

```
2024-05-01T15:37:03.271+05:30  INFO 46071 --- [           main] c.c.l.SpringbootApplication
2024-05-01T15:37:03.272+05:30  INFO 46071 --- [           main] c.c.l.SpringbootApplication
2024-05-01T15:37:03.710+05:30  INFO 46071 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-01T15:37:03.716+05:30  INFO 46071 --- [           main] o.apache.catalina.core.StandardService
2024-05-01T15:37:03.716+05:30  INFO 46071 --- [           main] o.apache.catalina.core.StandardEngine
2024-05-01T15:37:03.739+05:30  INFO 46071 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-01T15:37:03.740+05:30  INFO 46071 --- [           main] w.s.c.ServletWebServerApplicationContext
User initialization
User object hashCode: 254812619
2024-05-01T15:37:03.920+05:30  INFO 46071 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-01T15:37:03.926+05:30  INFO 46071 --- [           main] c.c.l.SpringbootApplication
2024-05-01T15:37:26.050+05:30  INFO 46071 --- [nio-8090-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-01T15:37:26.051+05:30  INFO 46071 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet
2024-05-01T15:37:26.051+05:30  INFO 46071 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet
TestController1 instance initialization
TestController1 object hashCode: 1476370807 User object hashCode: 254812619
fetchUser api invoked
fetchUser api invoked
```

3. localhost:8090/api/logout

```
2024-05-01T17:46:42.430+05:30  INFO 52913 --- [           main] w.s.c.ServletWebServerApplicationContext
User initialization
User object hashCode: 254812619
2024-05-01T17:46:42.585+05:30  INFO 52913 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-01T17:46:42.590+05:30  INFO 52913 --- [           main] c.c.l.SpringbootApplication
2024-05-01T17:46:47.076+05:30  INFO 52913 --- [nio-8090-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-01T17:46:47.076+05:30  INFO 52913 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet
2024-05-01T17:46:47.077+05:30  INFO 52913 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet
TestController1 instance initialization
TestController1 object hashCode: 406833946 User object hashCode: 254812619
fetchUser api invoked
fetchUser api invoked
end the session
```

4. localhost:8090/api/fetchUser

```
2024-05-01T17:46:42.402+05:30  INFO 52913 --- [           main] o.apache.catalina.core.StandardEngine
2024-05-01T17:46:42.429+05:30  INFO 52913 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-01T17:46:42.430+05:30  INFO 52913 --- [           main] w.s.c.ServletWebServerApplicationContext
User initialization
User object hashCode: 254812619
2024-05-01T17:46:42.585+05:30  INFO 52913 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-01T17:46:42.590+05:30  INFO 52913 --- [           main] c.c.l.SpringbootApplication
2024-05-01T17:46:47.076+05:30  INFO 52913 --- [nio-8090-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-01T17:46:47.076+05:30  INFO 52913 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet
2024-05-01T17:46:47.077+05:30  INFO 52913 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet
TestController1 instance initialization
TestController1 object hashCode: 406833946 User object hashCode: 254812619
fetchUser api invoked
fetchUser api invoked
end the session
TestController1 instance initialization
TestController1 object hashCode: 754846954 User object hashCode: 254812619
fetchUser api invoked
```

Unsatisfied Dependency problem

```java
@RestController
@RequestMapping(value = "/api")
public class User {

    @Autowired
    Order order;

    @PostMapping("/createOrder")
    public ResponseEntity<String> createOrder() {

        order.createOrder();
        return ResponseEntity.ok( body: "");
    }
}
```

```java
public interface Order {

    public void createOrder();
}
```

```java
@Component
public class OnlineOrder implements Order{

    public OnlineOrder(){
        System.out.println("Online Order Initialized");
    }

    public void createOrder(){
        System.out.println("created Online Order");
    }
}
```

```java
@Component
public class OfflineOrder implements Order{

    public OfflineOrder(){
        System.out.println("Offline Order initialized");
    }

    public void createOrder(){
        System.out.println("created Offline Order");
    }
}
```

```
**************************
APPLICATION FAILED TO START
**************************
```

```
UnsatisfiedDependencyException: Error creating bean with name 'user'
```

@Qualifier

```java
@RestController
@RequestMapping(value = "/api")
public class User {

    @Qualifier("onlineOrderObject")
    @Autowired
    Order order;


    @PostMapping("/createOrder")
    public ResponseEntity<String> createOrder() {

        order.createOrder();
        return ResponseEntity.ok( body: "");
    }
}
```

```java
public interface Order {

    public void createOrder();
}
```

```java
@Qualifier("onlineOrderObject")
@Component
public class OnlineOrder implements Order{

    public OnlineOrder(){
        System.out.println("Online Order Initialized");
    }

    public void createOrder(){
        System.out.println("created Online Order");
    }
}
```

```java
@Qualifier("offlineOrderObject")
@Component
public class OfflineOrder implements Order{

    public OfflineOrder(){
        System.out.println("Offline Order initialized");
    }

    public void createOrder(){
        System.out.println("created Offline Order");
    }
}
```

```
2024-05-07T16:41:14.520+05:30  INFO 98826 --- [           main] c.c.l.SpringbootApplication
2024-05-07T16:41:14.521+05:30  INFO 98826 --- [           main] c.c.l.SpringbootApplication
2024-05-07T16:41:14.953+05:30  INFO 98826 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-07T16:41:14.959+05:30  INFO 98826 --- [           main] o.apache.catalina.core.StandardService
2024-05-07T16:41:14.959+05:30  INFO 98826 --- [           main] o.apache.catalina.core.StandardEngine
2024-05-07T16:41:14.985+05:30  INFO 98826 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T16:41:14.985+05:30  INFO 98826 --- [           main] w.s.c.ServletWebServerApplicationContext
Offline Order initialized
Online Order Initialized
2024-05-07T16:41:15.149+05:30  INFO 98826 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-07T16:41:15.154+05:30  INFO 98826 --- [           main] c.c.l.SpringbootApplication
```

```
POST    http://localhost:8090/api/createOrder
```

```
2024-05-07T16:41:14.959+05:30  INFO 98826 --- [           main] o.apache.catalina.core.StandardEngine
2024-05-07T16:41:14.985+05:30  INFO 98826 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T16:41:14.985+05:30  INFO 98826 --- [           main] w.s.c.ServletWebServerApplicationContext
Offline Order initialized
Online Order Initialized
2024-05-07T16:41:15.149+05:30  INFO 98826 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-07T16:41:15.154+05:30  INFO 98826 --- [           main] c.c.l.SpringbootApplication
2024-05-07T16:41:54.579+05:30  INFO 98826 --- [nio-8090-exec-2] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T16:41:54.579+05:30  INFO 98826 --- [nio-8090-exec-2] o.s.web.servlet.DispatcherServlet
2024-05-07T16:41:54.580+05:30  INFO 98826 --- [nio-8090-exec-2] o.s.web.servlet.DispatcherServlet
created Online Order
```

1st Solution:

```java
@RestController
@RequestMapping(value = "/api")
public class User {

    @Qualifier("onlineOrderObject")
    @Autowired
    Order onLineOrderObj;

    @Qualifier("offlineOrderObject")
    @Autowired
    Order offlineOrderObj;

    @PostMapping("/createOrder")
    public ResponseEntity<String> createOrder(@RequestParam boolean isOnlineOrder) {

        if (isOnlineOrder) {
            onlineOrderObj.createOrder();
        } else {
            offlineOrderObj.createOrder();
        }

        return ResponseEntity.ok( body: "");
    }
}
```

```java
public interface Order {

    public void createOrder();
}
```

```java
@Qualifier("onlineOrderObject")
@Component
public class OnlineOrder implements Order{

    public OnlineOrder(){
        System.out.println("Online Order Initialized");
    }

    public void createOrder(){
        System.out.println("created Online Order");
    }
}
```

```java
@Qualifier("offlineOrderObject")
@Component
public class OfflineOrder implements Order{

    public OfflineOrder(){
        System.out.println("Offline Order initialized");
    }

    public void createOrder(){
        System.out.println("created Offline Order");
    }
}
```

2nd Solution:

```java
@RestController
@RequestMapping(value = "/api")
public class User {

    @Autowired
    Order order;

    @PostMapping("/createOrder")
    public ResponseEntity<String> createOrder() {

        order.createOrder();

        return ResponseEntity.ok( body: "");
    }
}
```

```java
public interface Order {

    public void createOrder();
}
```

```java
public class OnlineOrder implements Order{

    public OnlineOrder(){
        System.out.println("Online Order Initialized");
    }

    public void createOrder(){
        System.out.println("created Online Order");
    }
}
```

```java
public class OfflineOrder implements Order{

    public OfflineOrder(){
        System.out.println("Offline Order initialized");
    }

    public void createOrder(){
        System.out.println("created Offline Order");
    }
}
```

```java
@Configuration
public class AppConfig {

    @Bean
    public Order createOrderBean(@Value("${isOnlineOrder}") boolean isOnlineOrder){
        if(isOnlineOrder) {
            return new OnlineOrder();
        } else {
            return new OfflineOrder();
        }
    }
}
```

application.properties

```
isOnlineOrder=false
```

```
2024-05-08T19:30:38.605+05:30  INFO 45167 --- [           main] o.apache.catalina.core.StandardService   :
2024-05-08T19:30:38.606+05:30  INFO 45167 --- [           main] o.apache.catalina.core.StandardEngine    :
2024-05-08T19:30:38.631+05:30  INFO 45167 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       :
2024-05-08T19:30:38.632+05:30  INFO 45167 --- [           main] w.s.c.ServletWebServerApplicationContext :
Offline Order initialized
2024-05-08T19:30:38.793+05:30  INFO 45167 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  :
2024-05-08T19:30:38.798+05:30  INFO 45167 --- [           main] c.c.l.SpringbootApplication              :
```

POST http://localhost:8090/api/createOrder

```
2024-05-08T19:30:38.606+05:30  INFO 45167 --- [           main] o.apache.catalina.core.StandardEngine
2024-05-08T19:30:38.631+05:30  INFO 45167 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-08T19:30:38.632+05:30  INFO 45167 --- [           main] w.s.c.ServletWebServerApplicationContext
Offline Order initialized
2024-05-08T19:30:38.793+05:30  INFO 45167 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-08T19:30:38.798+05:30  INFO 45167 --- [           main] c.c.l.SpringbootApplication
2024-05-08T19:32:33.610+05:30  INFO 45167 --- [nio-8090-exec-2] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-08T19:32:33.610+05:30  INFO 45167 --- [nio-8090-exec-2] o.s.web.servlet.DispatcherServlet
2024-05-08T19:32:33.611+05:30  INFO 45167 --- [nio-8090-exec-2] o.s.web.servlet.DispatcherServlet
created Offline Order
```

@Value :

it is used to inject values from various sources like property file, environment variables or inline literals.

Inline Literal Example

```
@Bean
public Order createOrderBean(@Value("false") boolean isOnlineOrder){
    if(isOnlineOrder) {
        return new OnlineOrder();
    } else {
        return new OfflineOrder();
    }
}
```

@ConditionalOnProperty :

Bean is created Conditionally (mean Bean can be created or Not).

We have already know the behavior of below program:

```java
@Component
public class DBConnection {

    @Autowired
    MySQLConnection mySQLConnection;

    @Autowired
    NoSQLConnection noSQLConnection;

    @PostConstruct
    public void init(){
        System.out.println("DB Connection Bean Created with dependencies below:");
        System.out.println("is MySQLConnection object Null: " + Objects.isNull(mySQLConnection));
        System.out.println("is NoSQLConnection object Null: " + Objects.isNull(noSQLConnection));
    }
}
```

```java
@Component
public class NoSQLConnection {

    NoSQLConnection() {
        System.out.println("initialization of NoSQLConnection Bean");
    }
}
```

```java
@Component
public class MySQLConnection {

    MySQLConnection(){
        System.out.println("initialization of MySQLConnection Bean");
    }
}
```

```
2024-05-25T10:55:57.207+05:30  INFO 6700 --- [           main] w.s.c.ServletWebServerApplicationContext
initialization of MySQLConnection Bean
initialization of NoSQLConnection Bean
DB Connection Bean Created with dependencies below:
is MySQLConnection object Null: false
is NoSQLConnection object Null: false
2024-05-25T10:55:57.381+05:30  INFO 6700 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-25T10:55:57.387+05:30  INFO 6700 --- [           main] c.c.l.SpringbootApplication
```

But how you will handle below Use Cases?

Use case 1:

We want to create only 1 Bean, either *MySQLConnection* or *NoSQLConnection*.

Use case 2:

We have 2 components, sharing same codebase, But 1 component need *MYSQLConnection* and other needs *NoSQLConnection*

Solution is @ConditionalOnProperty Annotation

```java
@Component
public class DBConnection {

    @Autowired(required = false)
    MySQLConnection mySQLConnection;

    @Autowired(required = false)
    NoSQLConnection noSQLConnection;

    @PostConstruct
    public void init(){
        System.out.println("DB Connection Bean Created with dependencies below:");
        System.out.println("is MySQLConnection object Null: " + Objects.isNull(mySQLConnection));
        System.out.println("is NoSQLConnection object Null: " + Objects.isNull(noSQLConnection));
    }
}
```

```java
@Component
@ConditionalOnProperty(prefix = "sqlconnection", value = "enabled", havingValue = "true", matchIfMissing = false)
public class MySQLConnection {

    MySQLConnection(){
        System.out.println("initialization of MySQLConnection Bean");
    }
}
```

```java
@Component
@ConditionalOnProperty(prefix = "nosqlconnection", value = "enabled", havingValue = "true", matchIfMissing = false)
public class NoSQLConnection {

    NoSQLConnection() {
        System.out.println("initialization of NoSQLConnection Bean");
    }
}
```

Application.properties

```
sqlconnection.enabled=true
```

```
2024-05-25T11:32:03.950+05:30  INFO 8352 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-25T11:32:03.951+05:30  INFO 8352 --- [           main] w.s.c.ServletWebServerApplicationContext
initialization of MySQLConnection Bean
DB Connection Bean Created with dependencies below:
is MySQLConnection object Null: false
is NoSQLConnection object Null: true
2024-05-25T11:32:04.102+05:30  INFO 8352 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-05-25T11:32:04.106+05:30  INFO 8352 --- [           main] c.c.l.SpringbootApplication
```

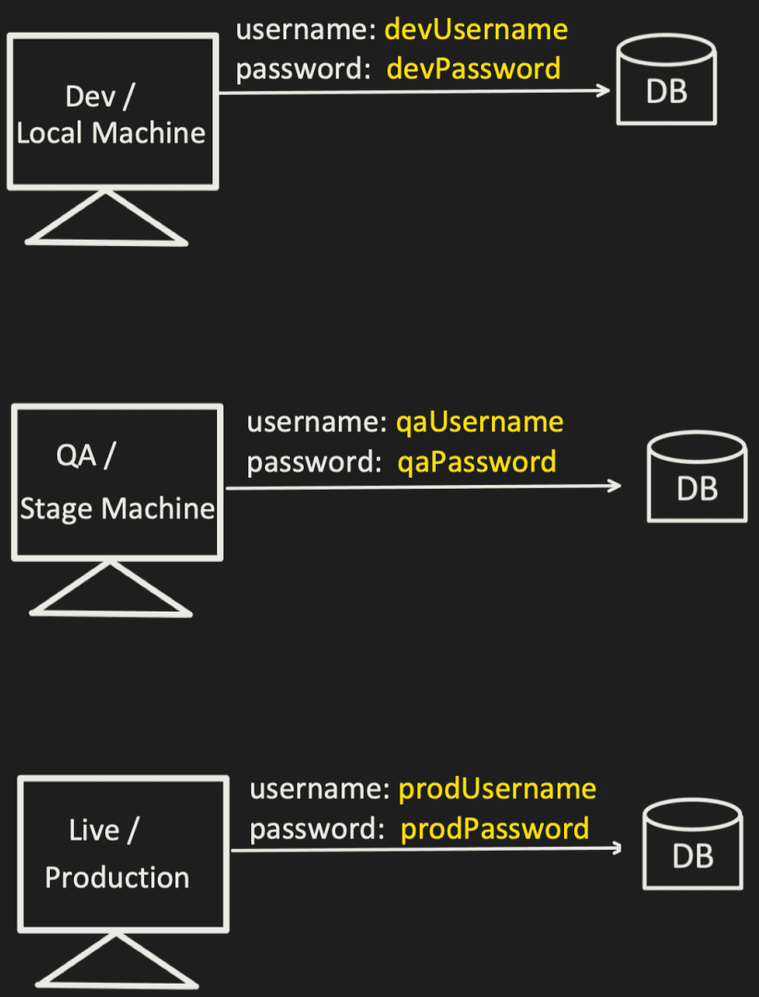
Advantages:

1. Toggling of Feature
2. Avoid cluttering Application context with un-necessary beans.
3. Save Memory
4. Reduce application Startup time

Disadvantages:

1. Misconfiguration can happen.
2. Code Complexity when over used.
3. Multiple bean creation with same Configuration, brings confusion.
4. Complexity in managing.

Let's deep dive into **@Profile** annotation to understand it better



This "username", "password" is just one example.
There are so many other configuration, which are different for different environments, like:

- URL and Port number
- Connection timeout values
- Request timeout values
- Throttle values
- Retry values etc.

How to do it?
We put the configurations in "application.properties" file.
But how to handle, different environment configurations?

That's where Profiling comes into the picture


```java
@Component
public class MySQLConnection {

    @Value("${username}")
    String username;

    @Value("${password}")
    String password;

    @PostConstruct
    public void init(){
        System.out.println("username: " + username + " passowrd: " + password);
    }
}
```

application.properties

```
username=defaultUsername
password=defaultPassword
```

application-dev.properties

```
username=devUsername
password=devPassword
```

application-qa.properties

```
username=qaUsername
password=qaPassword
```

application-prod.properties

```
username=prodUsername
password=prodPassword
```

```
2024-06-01T22:55:24.548+05:30  INFO 48119 --- [           main] c.c.l.SpringbootApplication
2024-06-01T22:55:24.549+05:30  INFO 48119 --- [           main] c.c.l.SpringbootApplication
2024-06-01T22:55:24.982+05:30  INFO 48119 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-06-01T22:55:24.990+05:30  INFO 48119 --- [           main] o.apache.catalina.core.StandardService
2024-06-01T22:55:24.990+05:30  INFO 48119 --- [           main] o.apache.catalina.core.StandardEngine
2024-06-01T22:55:25.016+05:30  INFO 48119 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-06-01T22:55:25.016+05:30  INFO 48119 --- [           main] w.s.c.ServletWebServerApplicationContext
username: defaultUsername passowrd: defaultPassword
2024-06-01T22:55:25.175+05:30  INFO 48119 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2024-06-01T22:55:25.179+05:30  INFO 48119 --- [           main] c.c.l.SpringbootApplication
```

And during application startup, we can tell spring boot to pick specific "application properties" file,
Using "spring.profiles.active" configuration.

```java
@Component
public class MySQLConnection {

    @Value("${username}")
    String username;

    @Value("${password}")
    String password;

    @PostConstruct
    public void init(){
        System.out.println("username: " + username + " passowrd: " + password);
    }
}
```

application.properties

```
username=defaultUsername
password=defaultPassword
spring.profiles.active=qa
```

application-dev.properties

```
username=devUsername
password=devPassword
```

application-qa.properties

```
username=qaUsername
password=qaPassword
```

application-prod.properties

```
username=prodUsername
password=prodPassword
```

```
2024-06-02T00:04:23.599+05:30  INFO 52175 --- [           main] c.c.l.SpringbootApplication              : The following 1 profile is active: "qa"
2024-06-02T00:04:24.002+05:30  INFO 52175 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2024-06-02T00:04:24.008+05:30  INFO 52175 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-06-02T00:04:24.008+05:30  INFO 52175 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-06-02T00:04:24.033+05:30  INFO 52175 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-06-02T00:04:24.033+05:30  INFO 52175 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 413 ms
username: qaUsername passowrd: qaPassword
2024-06-02T00:04:24.192+05:30  INFO 52175 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-06-02T00:04:24.198+05:30  INFO 52175 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 0.78 seconds (process running for 0.959)
```

We can pass the value to this configuration "spring.profiles.active" during application startup itself.

```
mvn spring-boot:run -Dspring-boot.run.profiles=prod
```

or

Add this in **Pom.xml**

```xml
<profiles>
    <profile>
        <id>local</id>
        <properties>
            <spring-boot.run.profiles>dev</spring-boot.run.profiles>
        </properties>
    </profile>
    <profile>
        <id>production</id>
        <properties>
            <spring-boot.run.profiles>prod</spring-boot.run.profiles>
        </properties>
    </profile>
    <profile>
        <id>stage</id>
        <properties>
            <spring-boot.run.profiles>qa</spring-boot.run.profiles>
        </properties>
    </profile>
</profiles>
```

```
mvn spring-boot:run -Pproduction
```

```
2024-06-02T00:21:57.336+05:30  INFO 53096 --- [           main] c.c.l.SpringbootApplication              : The following 1 profile is active: "prod"
2024-06-02T00:21:57.647+05:30  INFO 53096 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2024-06-02T00:21:57.652+05:30  INFO 53096 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-06-02T00:21:57.652+05:30  INFO 53096 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-06-02T00:21:57.673+05:30  INFO 53096 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-06-02T00:21:57.673+05:30  INFO 53096 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 318 ms
username: prodUsername passowrd: prodPassword
2024-06-02T00:21:57.804+05:30  INFO 53096 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
```

So, now we know, what's Profiling is. Lets see what is @Profile annotation.

Using @Profile annotation, we can tell spring boot, to create bean only when particular profile is set.

```java
@Component
@Profile("prod")
public class MySQLConnection {

    @Value("${username}")
    String username;

    @Value("${password}")
    String password;

    @PostConstruct
    public void init(){
        System.out.println("MySql username: " + username + " passowrd: " + password);
    }
}
```

```java
@Component
@Profile("dev")
public class NoSQLConnection {

    @Value("${username}")
    String username;

    @Value("${password}")
    String password;

    @PostConstruct
    public void init(){
        System.out.println("NoSQL username: " + username + " passowrd: " + password);
    }
}
```

application.properties

```
username=defaultUsername
password=defaultPassword
spring.profiles.active=qa
```

application-dev.properties

```
username=devUsername
password=devPassword
```

application-qa.properties

```
username=qaUsername
password=qaPassword
```

application-prod.properties

```
username=prodUsername
password=prodPassword
```

mvn spring-boot:run

```
2024-06-02T00:38:46.935+05:30  INFO 55048 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2024-06-02T00:38:46.940+05:30  INFO 55048 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-06-02T00:38:46.941+05:30  INFO 55048 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-06-02T00:38:46.964+05:30  INFO 55048 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-06-02T00:38:46.964+05:30  INFO 55048 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 410 ms
2024-06-02T00:38:47.109+05:30  INFO 55048 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
```

mvn spring-boot:run -Dspring-boot.run.profiles=prod

```
2024-06-02T00:42:30.245+05:30  INFO 55216 --- [           main] c.c.l.SpringbootApplication              : The following 1 profile is active: "prod"
2024-06-02T00:42:30.563+05:30  INFO 55216 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2024-06-02T00:42:30.568+05:30  INFO 55216 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-06-02T00:42:30.568+05:30  INFO 55216 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-06-02T00:42:30.591+05:30  INFO 55216 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-06-02T00:42:30.591+05:30  INFO 55216 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 327 ms
MySql username: prodUsername passowrd: prodPassword
2024-06-02T00:42:30.731+05:30  INFO 55216 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
```

We can also set multiple profiles at a time.

```java
@Component
@Profile("prod")
public class MySQLConnection {

    @Value("${username}")
    String username;

    @Value("${password}")
    String password;

    @PostConstruct
    public void init(){
        System.out.println("MySql username: " + username + " passowrd: " + password);
    }
}
```

```java
@Component
@Profile("qa")
public class NoSQLConnection {

    @Value("${username}")
    String username;

    @Value("${password}")
    String password;

    @PostConstruct
    public void init(){
        System.out.println("NoSQL username: " + username + " passowrd: " + password);
    }
}
```

application.properties

```
username=defaultUsername
password=defaultPassword
spring.profiles.active=prod,qa
```

application-dev.properties

```
username=devUsername
password=devPassword
```

application-qa.properties

```
username=qaUsername
password=qaPassword
```

application-prod.properties

```
username=prodUsername
password=prodPassword
```

```
2024-06-02T00:45:36.744+05:30  INFO 56836 --- [           main] c.c.l.SpringbootApplication              : The following 2 profiles are active: "prod", "qa"
2024-06-02T00:45:37.149+05:30  INFO 56836 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2024-06-02T00:45:37.156+05:30  INFO 56836 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-06-02T00:45:37.156+05:30  INFO 56836 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-06-02T00:45:37.179+05:30  INFO 56836 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-06-02T00:45:37.179+05:30  INFO 56836 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 413 ms
MySql username: qaUsername passowrd: qaPassword
NoSQL username: qaUsername passowrd: qaPassword
2024-06-02T00:45:37.327+05:30  INFO 56836 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
```

Now lets come back to the previous question, which I asked:

You have 2 Application and 1 common code base, how you will make sure that, BEAN is only created for 1 Application, not for other?

Common Codebase

```java
@Component
@Profile("app1")
public class NoSQLConnection {

    @Value("${username}")
    String username;

    @Value("${password}")
    String password;

    @PostConstruct
    public void init(){
        System.out.println("NoSQL username: " + username + " passowrd: " + password);
    }
}
```

Application1
(application.properties)

```
spring.profiles.active=app1
```

Application2
(application.properties)

```
spring.profiles.active=app2
```

AOP (Aspect Oriented Programming)

- In simple term, It helps to Intercept the method invocation. And we can perform some task before and after the method.
- AOP allow us to focus on business logic by handling boilerplate and repetitive code like logging, transaction management etc.
- So, Aspect is a module which handle this repetitive or boilerplate code.
- Helps in achieving reusability, maintainability of the code.

Used during:
- Logging
- Transaction Management
- Security etc..

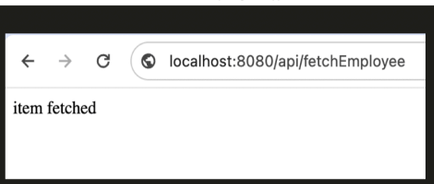
Dependency you need to add in pom.xml:

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```

```java
@RestController
@RequestMapping(value = "/api/")
public class Employee {

    @GetMapping (path = "/fetchEmployee")
    public String fetchEmployee(){
        return "item fetched";
    }
}
```

```java
@Component
@Aspect
public class LoggingAspect {

    @Before("execution(public String com.conceptandcoding.learningspringboot.Employee.fetchEmployee())")
    public void beforeMethod(){
        System.out.println("inside beforeMethod Aspect");
    }
}
```

```
2024-06-16T23:03:38.117+05:30  INFO 18766 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 6.4.4.Final
2024-06-16T23:03:38.137+05:30  INFO 18766 --- [           main] o.h.c.internal.RegionFactoryInitiator    : HHH000026: Second-level cache disabled
2024-06-16T23:03:38.237+05:30  INFO 18766 --- [           main] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: ignoring JPA class transformer
2024-06-16T23:03:38.398+05:30  INFO 18766 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform'
2024-06-16T23:03:38.400+05:30  INFO 18766 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2024-06-16T23:03:38.440+05:30  WARN 18766 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may
2024-06-16T23:03:38.632+05:30  INFO 18766 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-06-16T23:03:38.637+05:30  INFO 18766 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 1.517 seconds (process running for 1.727)
```




```
2024-06-16T23:03:38.237+05:30  INFO 18766 --- [           main] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: ignoring JPA class transformer
2024-06-16T23:03:38.398+05:30  INFO 18766 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000489: No JTA platform available (set 'hibernate.tra
2024-06-16T23:03:38.400+05:30  INFO 18766 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence uni
2024-06-16T23:03:38.440+05:30  WARN 18766 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore
2024-06-16T23:03:38.632+05:30  INFO 18766 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-06-16T23:03:38.637+05:30  INFO 18766 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 1.517 seconds (process
2024-06-16T23:05:49.486+05:30  INFO 18766 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-06-16T23:05:49.486+05:30  INFO 18766 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2024-06-16T23:05:49.487+05:30  INFO 18766 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
inside beforeMethod Aspect
```

Some important AOP concepts :

```
@Component
@Aspect
public class LoggingAspect {

    @Before("execution(public String com.conceptandcoding.learningspringboot.Employee.fetchEmployee())")
    public void beforeMethod(){
        System.out.println("inside beforeMethod Aspect");
    }
}
```

Access-modifiers: optional and can be omitted

Return type: optional but can not be omitted

This is a pointcut

This @Before and method together is called Advice

Pointcut:

Its an Expression, which tells where an ADVICE should be applied.

Types of Pointcut:

1. Execution: matches a particular method in a particular class.
   `@Before("execution(public String com.conceptandcoding.learningspringboot.Employee.fetchEmployee())")`

   → (*) wildcard : matches any single item

   Matches any return type

```
@Before("execution(* com.conceptandcoding.learningspringboot.Employee.fetchEmployee())")
public void beforeMethod(){
    System.out.println("inside beforeMethod Aspect");
}
```

   Matches any method with single parameter String

```
@Before("execution(* com.conceptandcoding.learningspringboot.Employee.*(String))")
public void beforeMethod(){
    System.out.println("inside beforeMethod Aspect");
}
```

   Matches fetchEmployee method that take any single parameter

```
@Before("execution(String com.conceptandcoding.learningspringboot.Employee.fetchEmployee(*))")
public void beforeMethod(){
    System.out.println("inside beforeMethod Aspect");
}
```

(..) wildcard : matches 0 or More item

Matches fetchEmployee method that take any 0 or More parameters

```
@Before("execution(String com.conceptandcoding.learningspringboot.Employee.fetchEmployee(..))")
public void beforeMethod(){
    System.out.println("inside beforeMethod Aspect");
}
```

Matches fetchEmployee method in *'com.conceptandcoding'* package and subpackage classes

```
@Before("execution(String com.conceptandcoding..fetchEmployee())")
public void beforeMethod(){
    System.out.println("inside beforeMethod Aspect");
}
```

Matches any method in *'com.conceptandcoding'* package and subpackage classes

```
@Before("execution(String com.conceptandcoding..*())")
public void beforeMethod(){
    System.out.println("inside beforeMethod Aspect");
}
```

2. Within: matches all method within any class or package.

This pointcut will run for each method in the class Employee

```
@Before("within(com.conceptandcoding.learningspringboot.Employee)")
```

This pointcut will run for each method in this package and subpackage

```
@Before("within(com.conceptandcoding.learningspringboot..*)")
```

3. @within: matches any method in a class which has this annotation.

```java
@RestController
@RequestMapping(value = "/api/")
public class Employee {

    @Autowired
    EmployeeUtil employeeUtil;

    @GetMapping (path = "/fetchEmployee")
    public String fetchEmployee(){
        employeeUtil.employeeHelperMethod();
        return "item fetched";
    }
}
```

```java
@Aspect
@Component
public class LoggingAspect {

    @Before("@within(org.springframework.stereotype.Service)")
    public void beforeMethod() {
        System.out.println("inside beforeMethod aspect");
    }
}
```

```java
@Service
public class EmployeeUtil {

    public void employeeHelperMethod() {
        System.out.println("employee helper method called");
    }
}
```

```
2024-06-22T11:54:02.459+05:30  INFO 66918 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 1.646 s
2024-06-22T11:54:04.070+05:30  INFO 66918 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet '
2024-06-22T11:54:04.070+05:30  INFO 66918 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2024-06-22T11:54:04.070+05:30  INFO 66918 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 0 ms
inside beforeMethod aspect
employee helper method called
```

4. @annotation: matches any method that is annotated with given annotation

```java
@RestController
@RequestMapping(value = "/api/")
public class Employee {

    @GetMapping (path = "/fetchEmployee")
    public String fetchEmployee(){
        return "item fetched";
    }
}
```

```java
@Component
@Aspect
public class LoggingAspect {

    @Before("@annotation(org.springframework.web.bind.annotation.GetMapping)")
    public void beforeMethod(){
        System.out.println("inside beforeMethod Aspect");
    }
}
```

5. Args: matches any method with particular arguments (or parameters)

@Before("args(String,int)")

```java
@RestController
@RequestMapping(value = "/api/")
public class Employee {

    @Autowired
    EmployeeUtil employeeUtil;

    @GetMapping (path = "/fetchEmployee")
    public String fetchEmployee(){
        employeeUtil.employeeHelperMethod( str: "xyz",  val: 123);
        return "item fetched";
    }
}
```

```java
@Aspect
@Component
public class LoggingAspect {

    @Before("args(String, int)")
    public void beforeMethod() {
        System.out.println("inside beforeMethod aspect");
    }
}
```

```java
@Service
public class EmployeeUtil {

    public void employeeHelperMethod(String str, int val) {
        System.out.println("employee helper method called");
    }
}
```

```
2024-06-22T11:13:27.258+05:30  INFO 64094 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing Persist
2024-06-22T11:13:27.290+05:30  INFO 64094 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core
2024-06-22T11:13:27.309+05:30  INFO 64094 --- [           main] o.h.c.internal.RegionFactoryInitiator    : HHH000026: Second-level cache
2024-06-22T11:13:27.414+05:30  INFO 64094 --- [           main] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: igno
2024-06-22T11:13:27.583+05:30  INFO 64094 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000489: No JTA platform av
2024-06-22T11:13:27.585+05:30  INFO 64094 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManager
2024-06-22T11:13:27.636+05:30  WARN 64094 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is en
2024-06-22T11:13:27.868+05:30  INFO 64094 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (
2024-06-22T11:13:27.874+05:30  INFO 64094 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication
2024-06-22T11:13:30.408+05:30  INFO 64094 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring Dispatche
2024-06-22T11:13:30.408+05:30  INFO 64094 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatc
2024-06-22T11:13:30.409+05:30  INFO 64094 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1
inside beforeMethod aspect
employee helper method called
```

If instead of primitive type, we need object, then we can give like this

@Before("args(com.conceptandcoding.learningspringboot.Employee)")

6. @args: matches any method with particular parameters and that parameter class is annotated with particular annotation.

@Before("@args(org.springframework.stereotype.Service)")

```java
@Service
public class EmployeeUtil {

    public void employeeHelperMethod(EmployeeDAO employeeDAO) {
        System.out.println("employee helper method called");
    }
}
```

```java
@Aspect
@Component
public class LoggingAspect {

    @Before("@args(org.springframework.stereotype.Service)")
    public void beforeMethod() {
        System.out.println("inside beforeMethod aspect");
    }
}
```

```java
@Service
public class EmployeeDAO {

}
```

7. target: matches any method on a particular instance of a class.

```java
@RestController
@RequestMapping(value = "/api/")
public class Employee {

    @Autowired
    EmployeeUtil employeeUtil;

    @GetMapping (path = "/fetchEmployee")
    public String fetchEmployee(){
        employeeUtil.employeeHelperMethod();
        return "item fetched";
    }
}
```

```java
@Aspect
@Component
public class LoggingAspect {

    @Before("target(com.conceptandcoding.learningspringboot.EmployeeUtil)")
    public void beforeMethod() {
        System.out.println("inside beforeMethod aspect");
    }
}
```

```java
@Component
public class EmployeeUtil {

    public void employeeHelperMethod() {
        System.out.println("employee helper method called");
    }
}
```

Interface

@Before("target(com.conceptandcoding.learningspringboot.IEmployee)")

```java
@RestController
@RequestMapping(value = "/api/")
public class EmployeeController {

    @Autowired
    @Qualifier("tempEmployee")
    IEmployee employeeObj;

    @GetMapping (path = "/fetchEmployee")
    public String fetchEmployee(){
        employeeObj.fetchEmployeeMethod();
        return "item fetched";
    }
}
```

```java
@Aspect
@Component
public class LoggingAspect {

    @Before("target(com.conceptandcoding.learningspringboot.IEmployee)")
    public void beforeMethod() {
        System.out.println("inside beforeMethod aspect");
    }
}
```

```java
public interface IEmployee {

    public void fetchEmployeeMethod();
}
```

```java
@Component
@Qualifier("tempEmployee")
public class TempEmployee implements IEmployee{
    @Override
    public void fetchEmployeeMethod() {
        System.out.println("in temp Employee fetch method");
    }
}
```

```java
@Component
@Qualifier("permaEmployee")
public class PermanentEmployee implements IEmployee{
    @Override
    public void fetchEmployeeMethod() {
        System.out.println("inside permanent fetch employee method");
    }
}
```

```
2024-06-22T12:12:42.485+05:30  INFO 67695 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication
2024-06-22T12:12:48.192+05:30  INFO 67695 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring Dispatcher
2024-06-22T12:12:48.192+05:30  INFO 67695 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatch
2024-06-22T12:12:48.193+05:30  INFO 67695 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1
inside beforeMethod aspect
in temp Employee fetch method
```

Combining two pointcuts using:

&& (boolean and)
|| (boolean or)

```java
@RestController
@RequestMapping(value = "/api/")
public class EmployeeController {

    @Autowired
    EmployeeUtil employeeUtil;

    @GetMapping (path = "/fetchEmployee")
    public String fetchEmployee(){
        return "item fetched";
    }
}
```

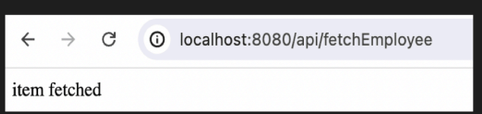
```java
@Aspect
@Component
public class LoggingAspect {

    @Before("execution(* com.conceptandcoding.learningspringboot.EmployeeController.*())"
            +
            "&& @within(org.springframework.web.bind.annotation.RestController)")
    public void beforeAndMethod() {
        System.out.println("inside beforeAndMethod aspect");
    }

    @Before("execution(* com.conceptandcoding.learningspringboot.EmployeeController.*())"
            +
            "|| @within(org.springframework.stereotype.Component)")
    public void beforeOrMethod() {
        System.out.println("inside beforeOrMethod aspect");
    }

}
```

```
2024-06-22T13:02:39.698+05:30  INFO 70500 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispat
2024-06-22T13:02:39.698+05:30  INFO 70500 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2024-06-22T13:02:39.699+05:30  INFO 70500 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
inside beforeAndMethod aspect
inside beforeOrMethod aspect
```

Named Pointcuts

```java
@RestController
@RequestMapping(value = "/api/")
public class EmployeeController {

    @Autowired
    EmployeeUtil employeeUtil;

    @GetMapping (path = "/fetchEmployee")
    public String fetchEmployee(){
        return "item fetched";
    }
}
```

```java
@Aspect
@Component
public class LoggingAspect {

    @Pointcut("execution(* com.conceptandcoding.learningspringboot.EmployeeController.*())")
    public void customPointcutName() {
        //always stays empty
    }

    @Before("customPointcutName()")
    public void beforeMethod() {
        System.out.println("inside beforeMethod aspect");
    }

}
```




```
2024-06-22T13:07:37.055+05:30  WARN 70647 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Th
2024-06-22T13:07:37.254+05:30  INFO 70647 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context p
2024-06-22T13:07:37.260+05:30  INFO 70647 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 1.526 seconds (p
2024-06-22T13:07:41.084+05:30  INFO 70647 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcher
2024-06-22T13:07:41.085+05:30  INFO 70647 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2024-06-22T13:07:41.085+05:30  INFO 70647 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 0 ms
inside beforeMethod aspect
```

Advice:

Its an action, which is taken @Before or @After or @Around the method execution.

```
@Component
@Aspect
public class LoggingAspect {

    @Before("execution(public String com.conceptandcoding.learningspringboot.Employee.fetchEmployee())")
    public void beforeMethod(){
        System.out.println("inside beforeMethod Aspect");
    }
}
```

Advice

@Before or @After: its simple and we have already seen

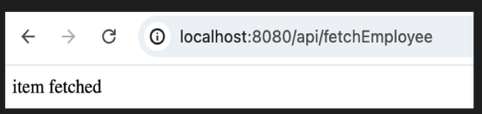
@Around:

As the name says, it surrounds the method execution (before and after both).

```
@RestController
@RequestMapping(value = "/api/")
public class EmployeeController {

    @Autowired
    EmployeeUtil employeeUtil;

    @GetMapping (path = "/fetchEmployee")
    public String fetchEmployee(){
        return "item fetched";
    }
}
```

```
@Aspect
@Component
public class LoggingAspect {

    @Around("execution(* com.conceptandcoding.learningspringboot.EmployeeUtil.*())")
    public void aroundMethod(ProceedingJoinPoint joinPoint) throws Throwable {
        System.out.println("inside before Method aspect");
        joinPoint.proceed();
        System.out.println("inside after Method aspect");
    }

}
```

```
@Component
public class EmployeeUtil {

    public void employeeHelperMethod() {
        System.out.println("employee helper method called");
    }
}
```



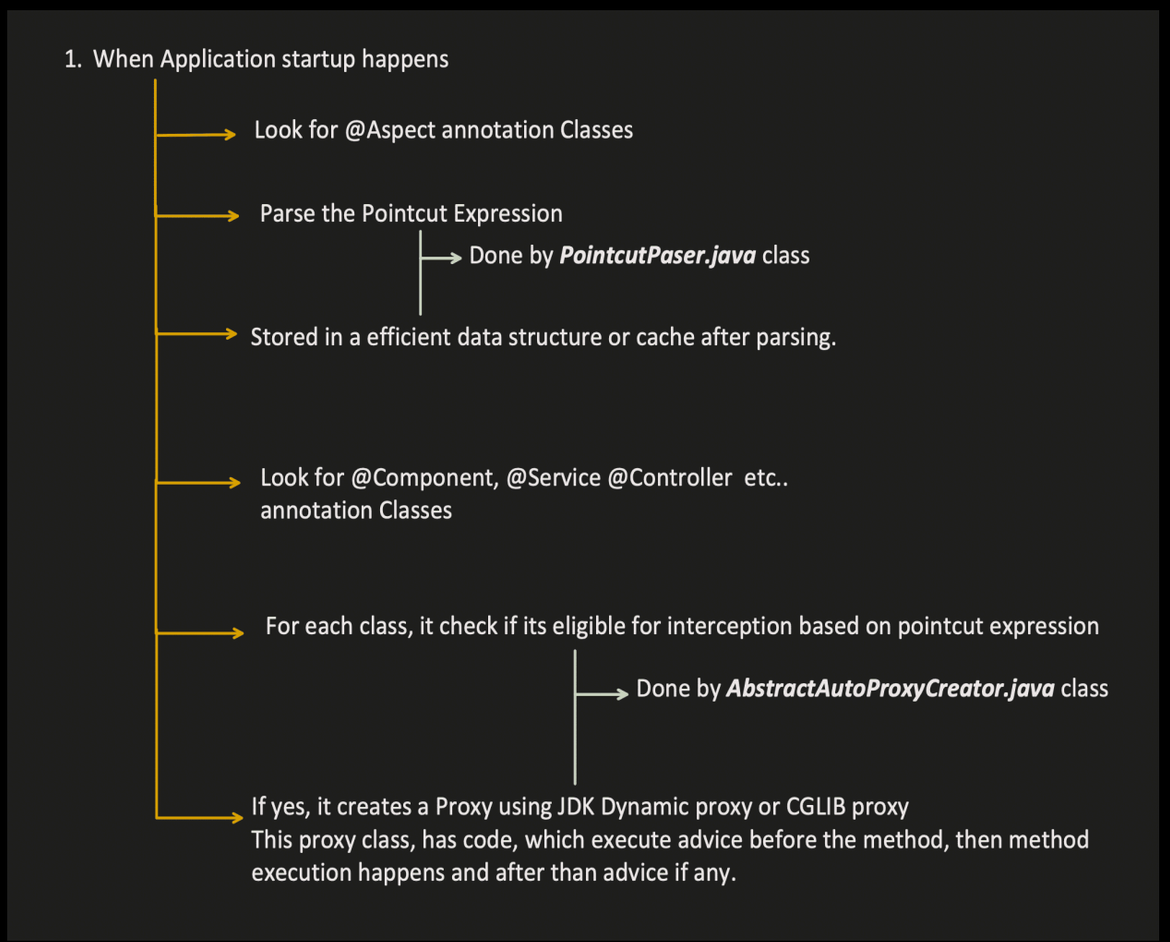

```
2024-06-22T16:25:39.014+05:30  INFO 79523 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication i
2024-06-22T16:25:41.031+05:30  INFO 79523 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherS
2024-06-22T16:25:41.031+05:30  INFO 79523 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatche
2024-06-22T16:25:41.032+05:30  INFO 79523 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 m
inside before Method aspect
fetching employee details
inside after Method aspect
```

Join Point: Its generally considered a point, where actual method invocation happens.

By now, few questions we all should have:

1. How this interception works?
2. What if we have 1000s of pointcut, so whenever I invoke a method, does matching happens with 1000s of pointcuts?

Let's understand the AOP flow, to get an answer of above doubts:


1. When Application startup happens
Look for @Aspect annotation Classes
Parse the Pointcut Expression
Done by PointcutPaser.java class
Stored in a efficient data structure or cache after parsing.
Look for @Component, @Service @Controller etc..
annotation Classes
For each class, it check if its eligible for interception based on pointcut expression
Done by AbstractAutoProxyCreator.java class
If yes, it creates a Proxy using JDK Dynamic proxy or CGLIB proxy
This proxy class, has code, which execute advice before the method, then method
execution happens and after than advice if any.

Now, when we initiate the Request after application startup, this Proxy class is actually invoked.

1. Parse the Pointcut Expression

PointcutParser.class

Decompiled .class file, bytecode version: 52.0 (Java 8) Download Sources

Search 0 results

```
171
172     public PointcutExpression parsePointcutExpression(String expression, Class<?> inScope, PointcutParameter[] formalParameters)
173         PointcutExpressionImpl pcExpr = null;
174
175         try {
176             Pointcut pc = this.resolvePointcutExpression(expression, inScope, formalParameters);
177             pc = this.concretizePointcutExpression(pc, inScope, formalParameters);
178             this.validateAgainstSupportedPrimitives(pc, expression);
179             pcExpr = new PointcutExpressionImpl(pc, expression, formalParameters, this.getWorld());
180             return pcExpr;
181         } catch (ParserException var6) {
182             throw new IllegalArgumentException(this.buildUserMessageFromParserException(expression, var6));
183         } catch (ReflectionWorld.ReflectionWorldException var7) {
184             throw new IllegalArgumentException(var7.getMessage());
185         }
186     }
187
```

2. Create Proxy class (either JDK Dynamic proxy or CGLIB proxy) if required for the class

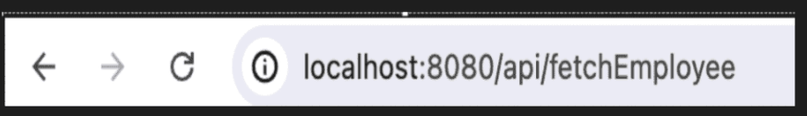
AbstractAutoProxyCreator.class

Decompiled .class file, bytecode version: 61.0 (Java 17) Download Sources Choose Sources...

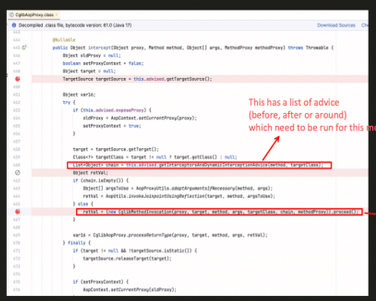
```
191         }
192     }
193
194     protected Object wrapIfNecessary(Object bean, String beanName, Object cacheKey) {  bean: EmployeeUtil@7593   beanName: "employeeUtil"
195         if (StringUtils.hasLength(beanName) && this.targetSourcedBeans.contains(beanName)) {  targetSourcedBeans:  size = 0
196             return bean;
197         } else if (Boolean.FALSE.equals(this.advisedBeans.get(cacheKey))) {
198             return bean;
199         } else if (!this.isInfrastructureClass(bean.getClass()) && !this.shouldSkip(bean.getClass(), beanName)) {
200             Object[] specificInterceptors = this.getAdvicesAndAdvisorsForBean(bean.getClass(), beanName, (TargetSource) null);  specificInterce
201             if (specificInterceptors != DO_NOT_PROXY) {
202                 this.advisedBeans.put(cacheKey, Boolean.TRUE);  cacheKey: "employeeUtil"   advisedBeans:  size = 56
203                 Object proxy = this.createProxy(bean.getClass(), beanName, specificInterceptors, new SingletonTargetSource(bean));  specificInt
204                 this.proxyTypes.put(cacheKey, proxy.getClass());
205                 return proxy;
206             } else {
207                 this.advisedBeans.put(cacheKey, Boolean.FALSE);
208                 return bean;
209             }
210         } else {
211             this.advisedBeans.put(cacheKey, Boolean.FALSE);
212             return bean;
213         }
214     }
215
```

Application startup success

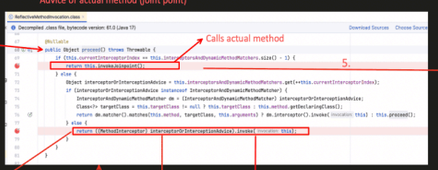
3. Invoke the request, which need Advice to be run.



4. Now, request actually tries to call Proxy Class, so it invokes either CGLIB proxy class or JDK proxy class

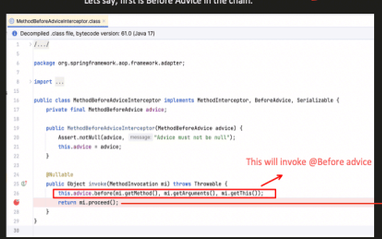


Invokes this, which decide either invoke specific Advice or actual method (joint point)



1.

Lets say, first is Before Advice in the chain.



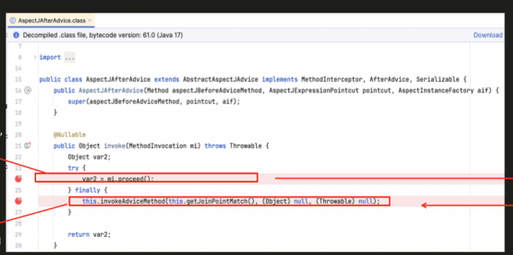
2.

3.

4.

6.

Lets say, second is After Advice in the chain (as counter ++ happens in above method right)



First proceed with other calls, either Before advice or Method execution

Then calls After Advice method