Spring boot : Introduction

"Concept & Coding" YT Video Notes

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.

Report Abuse

Servlet:

```
@WebServlet("/demoservlets")
public class DemoServlet extends HttpServlet {

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

        String requestPathInfo = request.getPathInfo();
        if(requestPathInfo.equals("/")) {
            //do something
        }
        else if(requestPathInfo.equals("/firstdemoapi")) {
            //do something
        }
        else if(requestPathInfo.equals("/secondmoapi")) {
            //do something
        }
    }

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

Web.xml

```
<!-- Map URL patterns to servlet classes -->
<servlet>
    <servlet-name>demoServlet</servlet-name>
    <servlet-class>com.example.DemoServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>demoServlet</servlet-name>
    <url-pattern>/demoservlets/*</url-pattern>
    <url-pattern>/demoservlets/firstdemoapi/*</url-pattern>
    <url-pattern>/demoservlets/secondmoapi/*</url-pattern>
</servlet-mapping>

<!-- Optional: Welcome file -->
<servlet>
    <servlet-name>demoServlet</servlet-name>
    <servlet-class>com.example.DemoServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>demoServlet</servlet-name>
    <url-pattern>/demoservlets/*</url-pattern>
</servlet-mapping>
```

Servlet:

```
@WebServlet("/demoservlets/*")
public class DemoServlet extends HttpServlet {

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

        //do something
    }

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

Spring Framework, solve challenges which exists with Servlets.

- Removal of web.xml
  - Web.xml was 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 development 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 user2Id){
        sender.getUserDetails(user2Id);
    }
}
```

```
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:

```
@Component
public class Payment {

    @Autowired
    User sender;

    void getSenderDetails(String user2Id){
        sender.getUserDetails(user2Id);
    }
}
```

```
@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 bit 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.
- Spring Boot has different integration available for:
  - Caching
  - Messaging
  - Security etc.

pom.xml

```
<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
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.conceptandcoding</groupId>
<artifactId>springprojectforlearning</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>
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.3.3</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
</project>
```

Controller class

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

    @Autowired
    PaymentSAO paymentService;

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

Config class

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

Dispatcher Servlet class.

```
public class MyWebAppInitializer
        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.

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.3</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.conceptandcoding</groupId>
<artifactId>springprojectforlearning</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.

```
@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 server 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

```
<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
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.conceptandcoding</groupId>
<artifactId>springprojectforlearning</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>
        <version>2.7.18</version>
        <exclusions>
            <exclusion>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-tomcat</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <version>2.7.18</version>
        </plugin>
    </plugins>
</build>
</project>
```

```
@SpringBootApplication
public class SpringbootApplication {

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

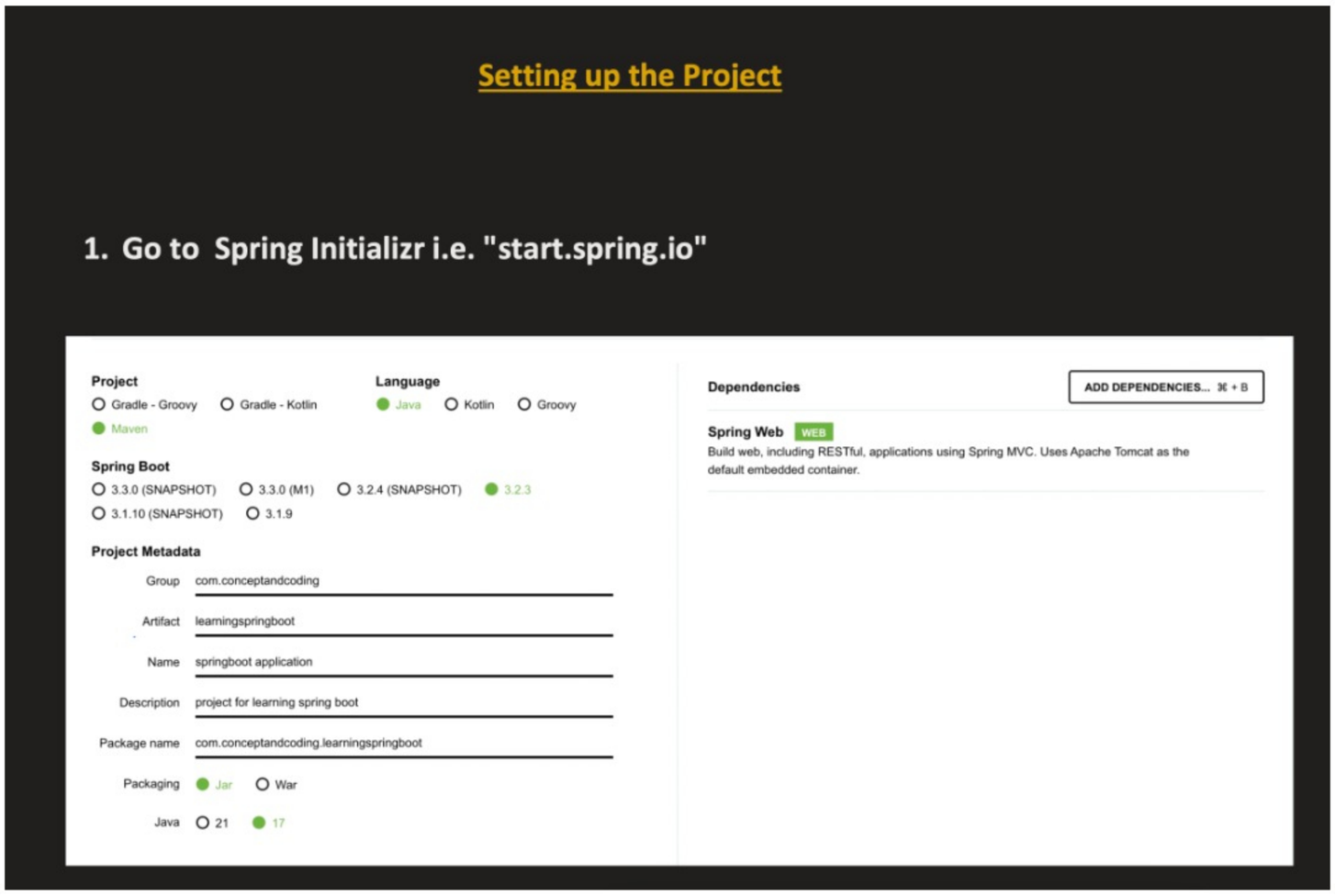
```
@RestController
@RequestMapping("/myapi")
public class MyController {

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

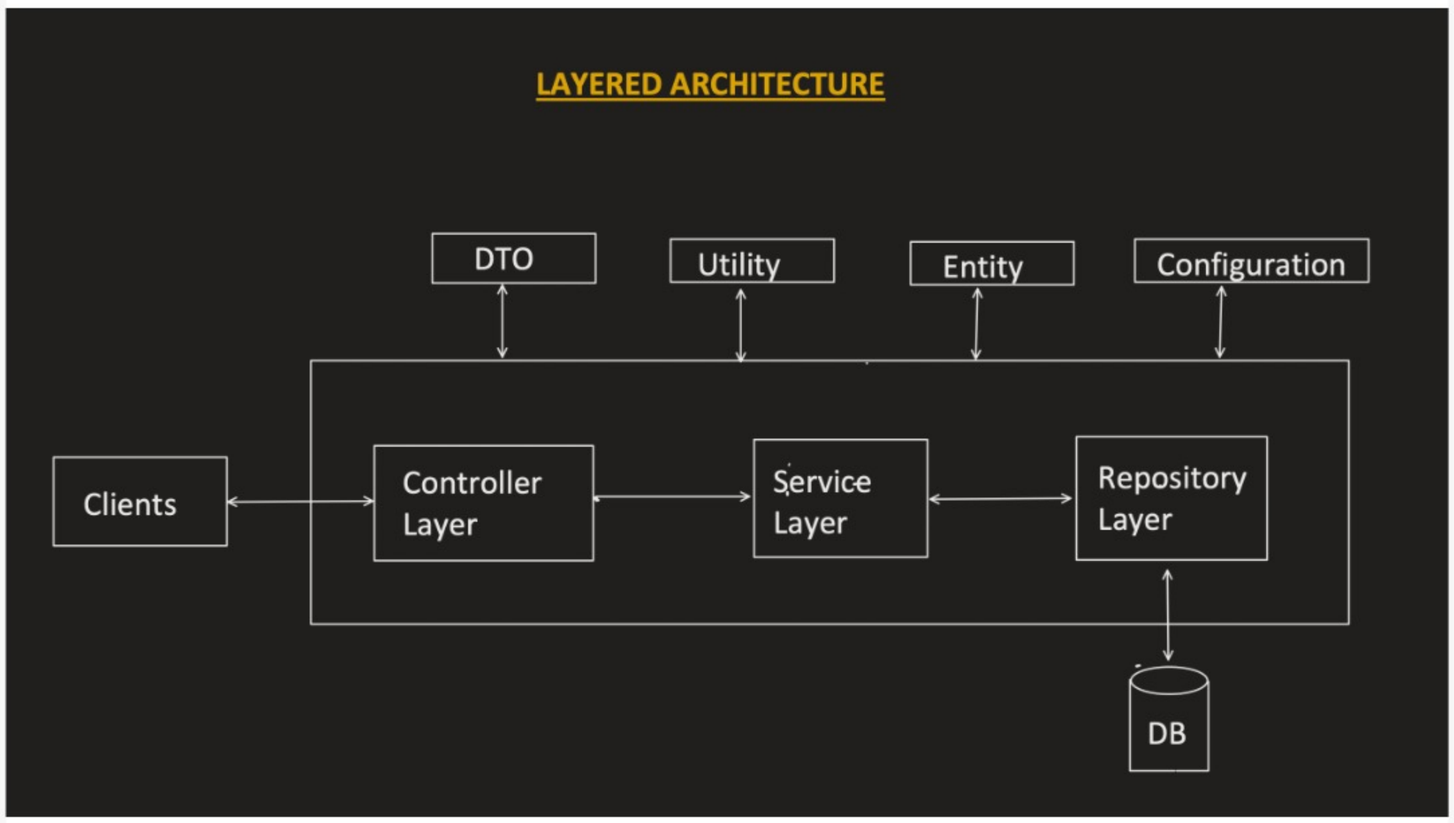
localhost:8080/myapi/firstapi

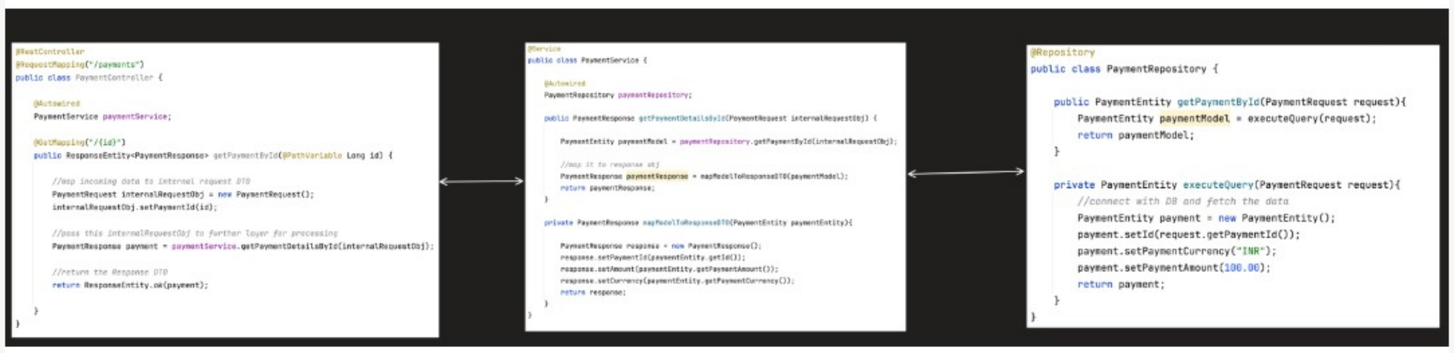
Hello from concept and coding

Spring boot: Project Setup and Layered Architecture (Concept …

"Concept && Coding" YT Video Notes







```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);

    }
}
```

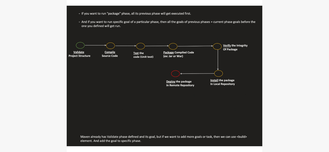
```java
@Service
public class PaymentService {

    @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;
    }
}
```

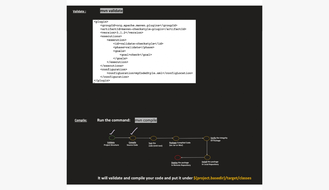
```java
@Repository
public class PaymentRepository {

    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;
    }
}
```

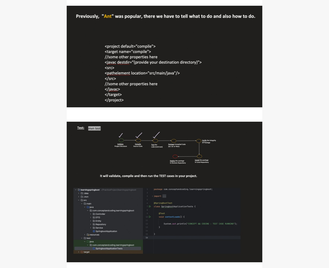
Report Abuse

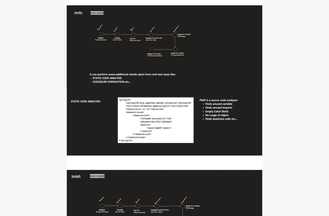
SpringBoot: Maven

"Concept && Coding" YT Video Notes









SpringBoot - Annotations (Controller Layer) Part1

"Concept && Coding" YT Video Notes

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

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

2. @RestController :

   RestController = Controller + ResponseBody

Report Abuse

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")

```
@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

```
@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?

```
@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;
    }
}
```

```
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.

```
@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).

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

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

```
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": "sjxytest@gmail.com"
}'
```

8. ResponseEntity : It represents the entire HTTP response.

   Header, status, response body etc.

```
@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);
    }
}
```

Spring boot : Bean and its Lifecycle

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.

How to create a Bean?

- @Component Annotation
- @Bean Annotation

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.

```
@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:

```
@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.

```
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; }
}
```

```
@Configuration
public class AppConfig {

    @Bean
    public User getDefaultUser(){
        return new User("defaultUsername", "defaultEmail");
    }
}
```

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

```
@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.

```
@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.

```
@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
-> Spring boot reads configuration metadata from @Configuration
-> IOC Container, make use of Configuration and @ComponentScan to look out for the Classes for which beans need to be created.

Starting IOC Container

```
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] c.c.c.m.ConfigurationClassPostProcessor : Starting enhancement of bean definitions
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] o.s.c.a.ConfigurationClassEnhancer : Enhancing [illegible] CGLIB
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] o.s.boot.StartupInfoLogger : Started SpringbootApplication in 0.[illegible] seconds (process running for 0.[illegible])
```

Step2: Construct the Beans

```
@Component
public class User {

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

```
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] c.c.c.m.ConfigurationClassPostProcessor : Starting enhancement of bean definitions
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] o.s.c.a.ConfigurationClassEnhancer : Enhancing [illegible] CGLIB
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] o.s.boot.StartupInfoLogger : Started SpringbootApplication in 0.[illegible] seconds (process running for 0.[illegible])
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] c.c.c.m.ConfigurationClassPostProcessor : Starting enhancement of bean definitions
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] o.s.c.a.ConfigurationClassEnhancer : Enhancing [illegible] CGLIB
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] o.s.boot.StartupInfoLogger : Started SpringbootApplication in 0.[illegible] seconds (process running for 0.[illegible])
```

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:
    i) Constructor Injection.
    ii) Setter Injection
    iii) Field Injection

*********you 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.

```
@Component
public class User {

    @Autowired
    Order order;

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

```
@Lazy
@Component
public class Order {

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

```
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] c.c.c.m.ConfigurationClassPostProcessor : Starting enhancement of bean definitions
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] o.s.c.a.ConfigurationClassEnhancer : Enhancing [illegible] CGLIB
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] o.s.boot.StartupInfoLogger : Started SpringbootApplication in 0.[illegible] seconds (process running for 0.[illegible])
Initializing user
Lazy: Initializing Order
```

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

```
@Component
public class User {

    @Autowired
    Order order;

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

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

```
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] c.c.c.m.ConfigurationClassPostProcessor : Starting enhancement of bean definitions
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] o.s.c.a.ConfigurationClassEnhancer : Enhancing [illegible] CGLIB
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] o.s.boot.StartupInfoLogger : Started SpringbootApplication in 0.[illegible] seconds (process running for 0.[illegible])
Initializing user
Task has been constructed and dependencies have been injected
Lazy: Initializing Order
```

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.

```
@SpringBootApplication
public class SpringbootApplication {

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

```
@Component
public class User {

    @Autowired
    Order order;

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

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

    @PreDestroy
    public void destroyUserBean(){ System.out.println("Destroying user"); }
}
```

```
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] c.c.c.m.ConfigurationClassPostProcessor : Starting enhancement of bean definitions
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] o.s.c.a.ConfigurationClassEnhancer : Enhancing [illegible] CGLIB
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] o.s.boot.StartupInfoLogger : Started SpringbootApplication in 0.[illegible] seconds (process running for 0.[illegible])
Initializing user
Task has been constructed and dependencies have been injected
Lazy: Initializing Order
Destroying user
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] o.s.c.a.AnnotationConfigApplicationContext : Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@[illegible], started on Wed Jun 08 00:07:20.[illegible] seconds (process running for 0.[illegible])
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] o.s.c.a.ConfigurationClassPostProcessor : Starting enhancement of bean definitions
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] o.s.c.a.ConfigurationClassEnhancer : Enhancing [illegible] CGLIB
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] o.s.boot.StartupInfoLogger : Started SpringbootApplication in 0.[illegible] seconds (process running for 0.[illegible])
Task has been constructed and dependencies have been injected
Initializing user
Lazy: Initializing Order
Destroying user
2022-06-08T00:07:20.[illegible]  INFO [illegible] --- [main] o.s.c.a.AnnotationConfigApplicationContext : Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@[illegible], started on Wed Jun 08 00:07:20.[illegible] seconds (process running for 0.[illegible])
```

Springboot: Dependency Injection

What is Dependency Injection?

> Using Dependency Injection, we can make our class independent of its dependencies.
>
> It helps to remove the dependency or control implementation and inject the dependencies from external source.

Let's see the Problem first:

```
@Component
public class Order {
    private Item item = new Item();

    public void run()
        System.out.println("initializing order");
    }
}
```

```
@Component
public class Order {
    private Item item;

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

Issue with above class structure

1. Not just one instance, but two Items created!
   - Instance Order object creates Item again (even though they use a Factory Object)
   - Because Order and Item are both Spring-managed beans, Spring sees that they have no dependencies and
   - Spring creates two separate instances of Item!

```
@Component
public class Order {
    private Item item = new Item();

    public void run()
        System.out.println("initializing order");
    }
}
```

```
@Component
public class Order {
    private Item item;

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

Spring Container

Order Instance(Spring) | Item Instance(Spring)

2. Tightly coupled

Dependency Injection Principle (DIP)

```
public class Order {
    private Item item = new DeliveryItem();

    public void run()
        System.out.println("initializing order");
    }
}
```

Violates Dependency Inversion Principle (DIP)

```
public class Order {
    private Item item;

    public void run()
        System.out.println("initializing order");
    }
}
```

How to Spring Core Fixes to enforce Dependency Inversion Principle?

Through Dependency Injection

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

```
@Component
public class Order {
    private Item item;

    public void run()
        System.out.println("initializing order");
    }
}
```

```
@Component
@Primary
public class Item {
}
```

@Component, first look for a bean of the required type.
If it does found, Spring will inject it.

Different ways of Injection and which one is better?

1. Field Injection
2. Setter Injection
3. Constructor Injection

Field Injection

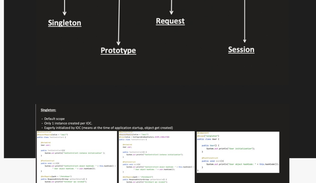
- Dependency is set into the fields of the class directly
- Spring uses reflection, it forcefully set the fields and resolve the dependency.

```
@Component
public class Order {
    private Item item;

    public void run()
        System.out.println("initializing order");
    }
}
```

```
@Component
@Primary
public class Item {

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

```
2025-06-03 18:47:01.887 INFO    com.i.springboot.DependencyInjectionApplication : Starting Application using Java 17.0.12 with PID 17
2025-06-03 18:47:01.889 DEBUG   com.i.springboot.DependencyInjectionApplication : Running with Spring Boot v3.4.5, Spring v6.2.6
2025-06-03 18:47:01.889 INFO    com.i.springboot.DependencyInjectionApplication : No active profile set, falling back to 1 default profile: "default"
2025-06-03 18:47:02.000 INFO    com.i.springboot.DependencyInjectionApplication : Started DependencyInjectionApplication in 0.112 seconds (process running for 0.127)
2025-06-03 18:47:02.000 DEBUG   com.i.springboot.Order : initializing order
2025-06-03 18:47:02.000 DEBUG   com.i.springboot.Order : initializing order
2025-06-03 18:47:02.000 DEBUG   com.i.springboot.Order : initializing order
2025-06-03 18:47:02.000 DEBUG   com.i.springboot.Item : order initialized
2025-06-03 18:47:02.000 DEBUG   com.i.springboot.Order : initializing order
```

Springboot: Bean Scopes by Concept && Coding



Singleton:

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

```
@Component
@Scope("singleton")
public class Student {
    private int id;
    private int age;
    private String name;
    private Address address;
    private Laptop laptop;
    private Scanner sc = new Scanner(System.in);
    public Student(int id, int age, String name, Address address, Laptop laptop) {
        this.id = id;
        this.age = age;
        this.name = name;
        this.address = address;
        this.laptop = laptop;
        System.out.println("Student instance being created");
    }
    public void setId(int id) {
        this.id = id;
        System.out.println("Student id: " + this.id);
    }
    public void setAge(int age) {
        this.age = age;
        System.out.println("Student age: " + this.age);
    }
    public void setName(String name) {
        this.name = name;
        System.out.println("Student name: " + this.name);
    }
    public void setAddress(Address address) {
        this.address = address;
        System.out.println("Student address: " + this.address);
    }
    public void setLaptop(Laptop laptop) {
        this.laptop = laptop;
        System.out.println("Student laptop: " + this.laptop);
    }
    public void getStudentInfo() {
        System.out.println("Student id: " + this.id);
        System.out.println("Student age: " + this.age);
        System.out.println("Student name: " + this.name);
        System.out.println("Student address: " + this.address);
        System.out.println("Student laptop: " + this.laptop);
    }
}
```

```
@Component
@Scope("singleton")
public class Address {
    private int id;
    private int houseNo;
    private int pincode;
    private Scanner sc = new Scanner(System.in);
    public Address(int id, int houseNo, int pincode) {
        this.id = id;
        this.houseNo = houseNo;
        this.pincode = pincode;
        System.out.println("Address being created");
    }
    public void setId(int id) {
        this.id = id;
        System.out.println("Address id: " + this.id);
    }
    public void setHouseNo(int houseNo) {
        this.houseNo = houseNo;
        System.out.println("Address house no: " + this.houseNo);
    }
    public void setPincode(int pincode) {
        this.pincode = pincode;
        System.out.println("Address pincode: " + this.pincode);
    }
    public void getAddressInfo() {
        System.out.println("Address id: " + this.id);
        System.out.println("Address house no: " + this.houseNo);
        System.out.println("Address pincode: " + this.pincode);
    }
}
```

```
2024-05-07T13:12:47.704+05:30  INFO 5204 --- [                  main] c.c.c.SpringbootApplication
2024-05-07T13:12:47.706+05:30  INFO 5204 --- [                  main] com.c.c.Student.Student
2024-05-07T13:12:47.707+05:30  INFO 5204 --- [                  main] com.c.c.Address.Address
2024-05-07T13:12:47.708+05:30  INFO 5204 --- [                  main] com.c.c.Laptop.Laptop
2024-05-07T13:12:47.710+05:30  INFO 5204 --- [                  main] o.apache.catalina.core.StandardService
2024-05-07T13:12:47.710+05:30  INFO 5204 --- [                  main] o.apache.catalina.core.StandardEngine
2024-05-07T13:12:47.720+05:30  INFO 5204 --- [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:47.721+05:30  INFO 5204 --- [                  main] w.s.c.ServletWebServerApplicationContext
Initialized Address
Initialized Laptop
Initialized Student
Student instance being created
Address being created
Laptop being created
Student id: 1
Student age: 21
Student name: Lucky
Student address: com.c.c.Address.Address@1
Student laptop: com.c.c.Laptop.Laptop@1
Address id: 1
Address house no: 1
Address pincode: 1
Laptop id: 1
Laptop name: Lenovo
Laptop price: 1
Student id: 1
Student age: 21
Student name: Lucky
Student address: com.c.c.Address.Address@1
Student laptop: com.c.c.Laptop.Laptop@1
TestController1 instance initialization
User instance
User object hashCode: 1140822370
Student object hashCode: 1944385571 User object hashCode: 1140822370
TestController2 instance initialization
TestController2 object hashCode: 1325214387 User object hashCode: 1140822370
2024-05-07T13:12:48.247+05:30  INFO 5204 --- [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.249+05:30  INFO 5204 --- [                  main] c.c.c.SpringbootApplication
2024-05-07T13:12:48.287+05:30  INFO 5204 --- [nio-8090-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.287+05:30  INFO 5204 --- [nio-8090-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.287+05:30  INFO 5204 --- [nio-8090-exec-1] w.s.c.ServletWebServerApplicationContext
2024-05-07T13:12:48.288+05:30  INFO 5204 --- [nio-8090-exec-1] w.s.c.ServletWebServerApplicationContext
Fetchuser api invoked
```

Prototype:

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

```
@Component
@Scope("prototype")
public class Student {
    private int id;
    private int age;
    private String name;
    private Address address;
    private Laptop laptop;
    private Scanner sc = new Scanner(System.in);
    public Student(int id, int age, String name, Address address, Laptop laptop) {
        this.id = id;
        this.age = age;
        this.name = name;
        this.address = address;
        this.laptop = laptop;
        System.out.println("Student being created");
    }
    public void setId(int id) {
        this.id = id;
        System.out.println("Student id: " + this.id);
    }
    public void setAge(int age) {
        this.age = age;
        System.out.println("Student age: " + this.age);
    }
    public void setName(String name) {
        this.name = name;
        System.out.println("Student name: " + this.name);
    }
    public void setAddress(Address address) {
        this.address = address;
        System.out.println("Student address: " + this.address);
    }
    public void setLaptop(Laptop laptop) {
        this.laptop = laptop;
        System.out.println("Student laptop: " + this.laptop);
    }
    public void getStudentInfo() {
        System.out.println("Student id: " + this.id);
        System.out.println("Student age: " + this.age);
        System.out.println("Student name: " + this.name);
        System.out.println("Student address: " + this.address);
        System.out.println("Student laptop: " + this.laptop);
    }
}
```

```
@Component
@Scope("prototype")
public class Address {
    private int id;
    private int houseNo;
    private int pincode;
    private Scanner sc = new Scanner(System.in);
    public Address(int id, int houseNo, int pincode) {
        this.id = id;
        this.houseNo = houseNo;
        this.pincode = pincode;
        System.out.println("Address being created");
    }
    public void setId(int id) {
        this.id = id;
        System.out.println("Address id: " + this.id);
    }
    public void setHouseNo(int houseNo) {
        this.houseNo = houseNo;
        System.out.println("Address house no: " + this.houseNo);
    }
    public void setPincode(int pincode) {
        this.pincode = pincode;
        System.out.println("Address pincode: " + this.pincode);
    }
    public void getAddressInfo() {
        System.out.println("Address id: " + this.id);
        System.out.println("Address house no: " + this.houseNo);
        System.out.println("Address pincode: " + this.pincode);
    }
}
```

```
2024-05-07T13:12:47.704+05:30  INFO 5204 --- [                  main] c.c.c.SpringbootApplication
2024-05-07T13:12:47.706+05:30  INFO 5204 --- [                  main] com.c.c.Student.Student
2024-05-07T13:12:47.707+05:30  INFO 5204 --- [                  main] com.c.c.Address.Address
2024-05-07T13:12:47.708+05:30  INFO 5204 --- [                  main] com.c.c.Laptop.Laptop
2024-05-07T13:12:47.710+05:30  INFO 5204 --- [                  main] o.apache.catalina.core.StandardService
2024-05-07T13:12:47.710+05:30  INFO 5204 --- [                  main] o.apache.catalina.core.StandardEngine
2024-05-07T13:12:47.720+05:30  INFO 5204 --- [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:47.721+05:30  INFO 5204 --- [                  main] w.s.c.ServletWebServerApplicationContext
TestController1 instance initialization
User instance
User object hashCode: 1140822370
Student instance initialization
Student being created
Student object hashCode: 1944385571 User object hashCode: 1140822370
TestController2 instance initialization
TestController2 object hashCode: 1325214387 User object hashCode: 1140822370
2024-05-07T13:12:48.247+05:30  INFO 5204 --- [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.249+05:30  INFO 5204 --- [                  main] c.c.c.SpringbootApplication
2024-05-07T13:12:48.287+05:30  INFO 5204 --- [nio-8090-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.287+05:30  INFO 5204 --- [nio-8090-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.287+05:30  INFO 5204 --- [nio-8090-exec-1] w.s.c.ServletWebServerApplicationContext
2024-05-07T13:12:48.288+05:30  INFO 5204 --- [nio-8090-exec-1] w.s.c.ServletWebServerApplicationContext
Fetchuser api invoked
```

Request:

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

```
@Component
@Scope("request")
public class Student {
    private int id;
    private int age;
    private String name;
    private Address address;
    private Laptop laptop;
    private Scanner sc = new Scanner(System.in);
    public Student(int id, int age, String name, Address address, Laptop laptop) {
        this.id = id;
        this.age = age;
        this.name = name;
        this.address = address;
        this.laptop = laptop;
        System.out.println("Student being created");
    }
    public void setId(int id) {
        this.id = id;
        System.out.println("Student id: " + this.id);
    }
    public void setAge(int age) {
        this.age = age;
        System.out.println("Student age: " + this.age);
    }
    public void setName(String name) {
        this.name = name;
        System.out.println("Student name: " + this.name);
    }
    public void setAddress(Address address) {
        this.address = address;
        System.out.println("Student address: " + this.address);
    }
    public void setLaptop(Laptop laptop) {
        this.laptop = laptop;
        System.out.println("Student laptop: " + this.laptop);
    }
    public void getStudentInfo() {
        System.out.println("Student id: " + this.id);
        System.out.println("Student age: " + this.age);
        System.out.println("Student name: " + this.name);
        System.out.println("Student address: " + this.address);
        System.out.println("Student laptop: " + this.laptop);
    }
}
```

```
@Component
@Scope("request")
public class Address {
    private int id;
    private int houseNo;
    private int pincode;
    private Scanner sc = new Scanner(System.in);
    public Address(int id, int houseNo, int pincode) {
        this.id = id;
        this.houseNo = houseNo;
        this.pincode = pincode;
        System.out.println("Address being created");
    }
    public void setId(int id) {
        this.id = id;
        System.out.println("Address id: " + this.id);
    }
    public void setHouseNo(int houseNo) {
        this.houseNo = houseNo;
        System.out.println("Address house no: " + this.houseNo);
    }
    public void setPincode(int pincode) {
        this.pincode = pincode;
        System.out.println("Address pincode: " + this.pincode);
    }
    public void getAddressInfo() {
        System.out.println("Address id: " + this.id);
        System.out.println("Address house no: " + this.houseNo);
        System.out.println("Address pincode: " + this.pincode);
    }
}
```

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.6.3)

2024-05-07T13:12:48.249+05:30  INFO 5204 ---  [                  main] c.c.c.SpringbootApplication
2024-05-07T13:12:48.249+05:30  INFO 5204 ---  [                  main] c.c.c.SpringbootApplication
2024-05-07T13:12:48.250+05:30  INFO 5204 ---  [                  main] o.apache.catalina.core.StandardService
2024-05-07T13:12:48.250+05:30  INFO 5204 ---  [                  main] o.apache.catalina.core.StandardEngine
2024-05-07T13:12:48.260+05:30  INFO 5204 ---  [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] w.s.c.ServletWebServerApplicationContext
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] w.s.c.ServletWebServerApplicationContext
```

```
2024-05-07T13:12:48.249+05:30  INFO 5204 --- [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.249+05:30  INFO 5204 --- [                  main] c.c.c.SpringbootApplication
2024-05-07T13:12:48.250+05:30  INFO 5204 --- [                  main] o.apache.catalina.core.StandardService
2024-05-07T13:12:48.250+05:30  INFO 5204 --- [                  main] o.apache.catalina.core.StandardEngine
2024-05-07T13:12:48.260+05:30  INFO 5204 --- [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.261+05:30  INFO 5204 --- [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.261+05:30  INFO 5204 --- [                  main] w.s.c.ServletWebServerApplicationContext
2024-05-07T13:12:48.261+05:30  INFO 5204 --- [                  main] w.s.c.ServletWebServerApplicationContext
Initialized Address
Initialized Laptop
Initialized Student
Address being created
Laptop being created
Student being created
Address id: 1
Address house no: 1
Address pincode: 1
Laptop id: 1
Laptop name: Lenovo
Laptop price: 1
Student id: 1
Student age: 21
Student name: Lucky
Student address: com.c.c.Address.Address@1
Student laptop: com.c.c.Laptop.Laptop@1
Address id: 1
Address house no: 1
Address pincode: 1
Laptop id: 1
Laptop name: Lenovo
Laptop price: 1
Student id: 1
Student age: 21
Student name: Lucky
Student address: com.c.c.Address.Address@1
Student laptop: com.c.c.Laptop.Laptop@1
TestController1 instance initialization
User instance
User object hashCode: 1140822370
Student instance initialization
Student being created
Student object hashCode: 1944385571 User object hashCode: 1140822370
TestController2 instance initialization
TestController2 object hashCode: 1325214387 User object hashCode: 1140822370
Fetchuser api invoked
```

Consider below, What will happen?

```
@Component
@Scope("request")
public class Student {
    private int id;
    private int age;
    private String name;
    private Address address;
    private Laptop laptop;
    private Scanner sc = new Scanner(System.in);
    public Student(int id, int age, String name, Address address, Laptop laptop) {
        this.id = id;
        this.age = age;
        this.name = name;
        this.address = address;
        this.laptop = laptop;
        System.out.println("Student being created");
    }
    public void setId(int id) {
        this.id = id;
        System.out.println("Student id: " + this.id);
    }
    public void setAge(int age) {
        this.age = age;
        System.out.println("Student age: " + this.age);
    }
    public void setName(String name) {
        this.name = name;
        System.out.println("Student name: " + this.name);
    }
    public void setAddress(Address address) {
        this.address = address;
        System.out.println("Student address: " + this.address);
    }
    public void setLaptop(Laptop laptop) {
        this.laptop = laptop;
        System.out.println("Student laptop: " + this.laptop);
    }
    public void getStudentInfo() {
        System.out.println("Student id: " + this.id);
        System.out.println("Student age: " + this.age);
        System.out.println("Student name: " + this.name);
        System.out.println("Student address: " + this.address);
        System.out.println("Student laptop: " + this.laptop);
    }
}
```

```
@Component
@Scope("request")
public class Address {
    private int id;
    private int houseNo;
    private int pincode;
    private Scanner sc = new Scanner(System.in);
    public Address(int id, int houseNo, int pincode) {
        this.id = id;
        this.houseNo = houseNo;
        this.pincode = pincode;
        System.out.println("Address being created");
    }
    public void setId(int id) {
        this.id = id;
        System.out.println("Address id: " + this.id);
    }
    public void setHouseNo(int houseNo) {
        this.houseNo = houseNo;
        System.out.println("Address house no: " + this.houseNo);
    }
    public void setPincode(int pincode) {
        this.pincode = pincode;
        System.out.println("Address pincode: " + this.pincode);
    }
    public void getAddressInfo() {
        System.out.println("Address id: " + this.id);
        System.out.println("Address house no: " + this.houseNo);
        System.out.println("Address pincode: " + this.pincode);
    }
}
```

```
2024-05-07T13:12:48.249+05:30  INFO 5204 ---  [                  main] c.c.c.SpringbootApplication
2024-05-07T13:12:48.250+05:30  INFO 5204 ---  [                  main] o.apache.catalina.core.StandardService
2024-05-07T13:12:48.250+05:30  INFO 5204 ---  [                  main] o.apache.catalina.core.StandardEngine
2024-05-07T13:12:48.260+05:30  INFO 5204 ---  [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] w.s.c.ServletWebServerApplicationContext
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] w.s.c.ServletWebServerApplicationContext
Initialized Address
Initialized Laptop
Initialized Student
Address being created
Laptop being created
Student being created
Address id: 1
Address house no: 1
Address pincode: 1
Laptop id: 1
Laptop name: Lenovo
Laptop price: 1
Student id: 1
Student age: 21
Student name: Lucky
Student address: com.c.c.Address.Address@1
Student laptop: com.c.c.Laptop.Laptop@1
Address id: 1
Address house no: 1
Address pincode: 1
Laptop id: 1
Laptop name: Lenovo
Laptop price: 1
Student id: 1
Student age: 21
Student name: Lucky
Student address: com.c.c.Address.Address@1
Student laptop: com.c.c.Laptop.Laptop@1
TestController1 instance initialization
User instance
User object hashCode: 1140822370
Student instance initialization
Student being created
Student object hashCode: 1944385571 User object hashCode: 1140822370
TestController2 instance initialization
TestController2 object hashCode: 1325214387 User object hashCode: 1140822370
Fetchuser api invoked
```

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.

```
@Component
@Scope("request")
public class Student {
    private int id;
    private int age;
    private String name;
    private Address address;
    private Laptop laptop;
    private Scanner sc = new Scanner(System.in);
    public Student(int id, int age, String name, Address address, Laptop laptop) {
        this.id = id;
        this.age = age;
        this.name = name;
        this.address = address;
        this.laptop = laptop;
        System.out.println("Student being created");
    }
    public void setId(int id) {
        this.id = id;
        System.out.println("Student id: " + this.id);
    }
    public void setAge(int age) {
        this.age = age;
        System.out.println("Student age: " + this.age);
    }
    public void setName(String name) {
        this.name = name;
        System.out.println("Student name: " + this.name);
    }
    public void setAddress(Address address) {
        this.address = address;
        System.out.println("Student address: " + this.address);
    }
    public void setLaptop(Laptop laptop) {
        this.laptop = laptop;
        System.out.println("Student laptop: " + this.laptop);
    }
    public void getStudentInfo() {
        System.out.println("Student id: " + this.id);
        System.out.println("Student age: " + this.age);
        System.out.println("Student name: " + this.name);
        System.out.println("Student address: " + this.address);
        System.out.println("Student laptop: " + this.laptop);
    }
}
```

```
@Component
@Scope("request")
public class User {
    private int id;
    private int age;
    private Scanner sc = new Scanner(System.in);
    public User(int id, int age) {
        this.id = id;
        this.age = age;
        System.out.println("User being created");
    }
    public void setId(int id) {
        this.id = id;
        System.out.println("User id: " + this.id);
    }
    public void setAge(int age) {
        this.age = age;
        System.out.println("User age: " + this.age);
    }
    public void getUserInfo() {
        System.out.println("User id: " + this.id);
        System.out.println("User age: " + this.age);
    }
}
```

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.6.3)

2024-05-07T13:12:48.249+05:30  INFO 5204 ---  [                  main] c.c.c.SpringbootApplication
2024-05-07T13:12:48.250+05:30  INFO 5204 ---  [                  main] o.apache.catalina.core.StandardService
2024-05-07T13:12:48.250+05:30  INFO 5204 ---  [                  main] o.apache.catalina.core.StandardEngine
2024-05-07T13:12:48.260+05:30  INFO 5204 ---  [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] w.s.c.ServletWebServerApplicationContext
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] w.s.c.ServletWebServerApplicationContext
Initialized Address
Initialized Laptop
Initialized Student
Address being created
Laptop being created
Student being created
Address id: 1
Address house no: 1
Address pincode: 1
Laptop id: 1
Laptop name: Lenovo
Laptop price: 1
Student id: 1
Student age: 21
Student name: Lucky
Student address: com.c.c.Address.Address@1
Student laptop: com.c.c.Laptop.Laptop@1
TestController1 instance initialization
User instance
User object hashCode: 1140822370
Student instance initialization
Student being created
Student object hashCode: 1944385571 User object hashCode: 1140822370
TestController2 instance initialization
TestController2 object hashCode: 1325214387 User object hashCode: 1140822370
Fetchuser api invoked
```

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.

```
@Component
@Scope("session")
public class Student {
    private int id;
    private int age;
    private String name;
    private Address address;
    private Laptop laptop;
    private Scanner sc = new Scanner(System.in);
    public Student(int id, int age, String name, Address address, Laptop laptop) {
        this.id = id;
        this.age = age;
        this.name = name;
        this.address = address;
        this.laptop = laptop;
        System.out.println("Student being created");
    }
    public void setId(int id) {
        this.id = id;
        System.out.println("Student id: " + this.id);
    }
    public void setAge(int age) {
        this.age = age;
        System.out.println("Student age: " + this.age);
    }
    public void setName(String name) {
        this.name = name;
        System.out.println("Student name: " + this.name);
    }
    public void setAddress(Address address) {
        this.address = address;
        System.out.println("Student address: " + this.address);
    }
    public void setLaptop(Laptop laptop) {
        this.laptop = laptop;
        System.out.println("Student laptop: " + this.laptop);
    }
    public void getStudentInfo() {
        System.out.println("Student id: " + this.id);
        System.out.println("Student age: " + this.age);
        System.out.println("Student name: " + this.name);
        System.out.println("Student address: " + this.address);
        System.out.println("Student laptop: " + this.laptop);
    }
}
```

```
@Component
@Scope("session")
public class Address {
    private int id;
    private int houseNo;
    private int pincode;
    private Scanner sc = new Scanner(System.in);
    public Address(int id, int houseNo, int pincode) {
        this.id = id;
        this.houseNo = houseNo;
        this.pincode = pincode;
        System.out.println("Address being created");
    }
    public void setId(int id) {
        this.id = id;
        System.out.println("Address id: " + this.id);
    }
    public void setHouseNo(int houseNo) {
        this.houseNo = houseNo;
        System.out.println("Address house no: " + this.houseNo);
    }
    public void setPincode(int pincode) {
        this.pincode = pincode;
        System.out.println("Address pincode: " + this.pincode);
    }
    public void getAddressInfo() {
        System.out.println("Address id: " + this.id);
        System.out.println("Address house no: " + this.houseNo);
        System.out.println("Address pincode: " + this.pincode);
    }
}
```

```
@Controller
@RestController("/")
public class TestController {
    private int id;
    private int age;
    private Scanner sc = new Scanner(System.in);
    public TestController(int id, int age) {
        this.id = id;
        this.age = age;
        System.out.println("TestController being created");
    }
    @Autowired
    User user;
    @Autowired
    Student student;
    @Autowired
    Address address;
    @Autowired
    Laptop laptop;
    public void setId(int id) {
        this.id = id;
        System.out.println("TestController id: " + this.id);
    }
    public void setAge(int age) {
        this.age = age;
        System.out.println("TestController age: " + this.age);
    }
    @GetMapping("/")
    public ResponseEntity<String> getUserInfo() {
        System.out.println("Fetchuser api invoked");
        return ResponseEntity.status(HttpStatus.OK).body("");
    }
    @GetMapping("/{id}")
    public ResponseEntity<String> getUserInfo(@PathVariable int id) {
        System.out.println("Fetchuser api invoked");
        return ResponseEntity.status(HttpStatus.OK).body("");
    }
    @GetMapping("/{id}/{age}")
    public ResponseEntity<String> getUserInfo(@PathVariable int id, @PathVariable int age) {
        System.out.println("Fetchuser api invoked");
        return ResponseEntity.status(HttpStatus.OK).body("");
    }
}
```

```
2024-05-07T13:12:48.249+05:30  INFO 5204 ---  [                  main] c.c.c.SpringbootApplication
2024-05-07T13:12:48.250+05:30  INFO 5204 ---  [                  main] o.apache.catalina.core.StandardService
2024-05-07T13:12:48.250+05:30  INFO 5204 ---  [                  main] o.apache.catalina.core.StandardEngine
2024-05-07T13:12:48.260+05:30  INFO 5204 ---  [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] w.s.c.ServletWebServerApplicationContext
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] w.s.c.ServletWebServerApplicationContext
Initialized Address
Initialized Laptop
Initialized Student
Address being created
Laptop being created
Student being created
Address id: 1
Address house no: 1
Address pincode: 1
Laptop id: 1
Laptop name: Lenovo
Laptop price: 1
Student id: 1
Student age: 21
Student name: Lucky
Student address: com.c.c.Address.Address@1
Student laptop: com.c.c.Laptop.Laptop@1
TestController being created
TestController1 instance initialization
User instance
User object hashCode: 1140822370
Student instance initialization
Student being created
Student object hashCode: 1944385571 User object hashCode: 1140822370
TestController2 instance initialization
TestController2 object hashCode: 1325214387 User object hashCode: 1140822370
Fetchuser api invoked
```

1. localhost:8090/api/fetchUser

```
2024-05-07T13:12:48.249+05:30  INFO 5204 ---  [                  main] c.c.c.SpringbootApplication
2024-05-07T13:12:48.250+05:30  INFO 5204 ---  [                  main] o.apache.catalina.core.StandardService
2024-05-07T13:12:48.250+05:30  INFO 5204 ---  [                  main] o.apache.catalina.core.StandardEngine
2024-05-07T13:12:48.260+05:30  INFO 5204 ---  [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] w.s.c.ServletWebServerApplicationContext
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] w.s.c.ServletWebServerApplicationContext
Initialized Address
Initialized Laptop
Initialized Student
Address being created
Laptop being created
Student being created
Address id: 1
Address house no: 1
Address pincode: 1
Laptop id: 1
Laptop name: Lenovo
Laptop price: 1
Student id: 1
Student age: 21
Student name: Lucky
Student address: com.c.c.Address.Address@1
Student laptop: com.c.c.Laptop.Laptop@1
TestController being created
TestController1 instance initialization
User instance
User object hashCode: 1140822370
Student instance initialization
Student being created
Student object hashCode: 1944385571 User object hashCode: 1140822370
TestController2 instance initialization
TestController2 object hashCode: 1325214387 User object hashCode: 1140822370
Fetchuser api invoked
```

2. localhost:8090/api/fetchUser

```
2024-05-07T13:12:48.249+05:30  INFO 5204 ---  [                  main] c.c.c.SpringbootApplication
2024-05-07T13:12:48.250+05:30  INFO 5204 ---  [                  main] o.apache.catalina.core.StandardService
2024-05-07T13:12:48.250+05:30  INFO 5204 ---  [                  main] o.apache.catalina.core.StandardEngine
2024-05-07T13:12:48.260+05:30  INFO 5204 ---  [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] w.s.c.ServletWebServerApplicationContext
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] w.s.c.ServletWebServerApplicationContext
Initialized Address
Initialized Laptop
Initialized Student
Address being created
Laptop being created
Student being created
Address id: 1
Address house no: 1
Address pincode: 1
Laptop id: 1
Laptop name: Lenovo
Laptop price: 1
Student id: 1
Student age: 21
Student name: Lucky
Student address: com.c.c.Address.Address@1
Student laptop: com.c.c.Laptop.Laptop@1
TestController being created
TestController1 instance initialization
User instance
User object hashCode: 1140822370
Student instance initialization
Student being created
Student object hashCode: 1944385571 User object hashCode: 1140822370
TestController2 instance initialization
TestController2 object hashCode: 1325214387 User object hashCode: 1140822370
Fetchuser api invoked
```

3. localhost:8090/api/fetchUser

```
2024-05-07T13:12:48.249+05:30  INFO 5204 ---  [                  main] c.c.c.SpringbootApplication
2024-05-07T13:12:48.250+05:30  INFO 5204 ---  [                  main] o.apache.catalina.core.StandardService
2024-05-07T13:12:48.250+05:30  INFO 5204 ---  [                  main] o.apache.catalina.core.StandardEngine
2024-05-07T13:12:48.260+05:30  INFO 5204 ---  [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] w.s.c.ServletWebServerApplicationContext
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] w.s.c.ServletWebServerApplicationContext
Initialized Address
Initialized Laptop
Initialized Student
Address being created
Laptop being created
Student being created
Address id: 1
Address house no: 1
Address pincode: 1
Laptop id: 1
Laptop name: Lenovo
Laptop price: 1
Student id: 1
Student age: 21
Student name: Lucky
Student address: com.c.c.Address.Address@1
Student laptop: com.c.c.Laptop.Laptop@1
TestController being created
TestController1 instance initialization
User instance
User object hashCode: 1140822370
Student instance initialization
Student being created
Student object hashCode: 1944385571 User object hashCode: 1140822370
TestController2 instance initialization
TestController2 object hashCode: 1325214387 User object hashCode: 1140822370
Fetchuser api invoked
```

4. localhost:8090/api/fetchUser

```
2024-05-07T13:12:48.249+05:30  INFO 5204 ---  [                  main] c.c.c.SpringbootApplication
2024-05-07T13:12:48.250+05:30  INFO 5204 ---  [                  main] o.apache.catalina.core.StandardService
2024-05-07T13:12:48.250+05:30  INFO 5204 ---  [                  main] o.apache.catalina.core.StandardEngine
2024-05-07T13:12:48.260+05:30  INFO 5204 ---  [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] o.a.c.c.C.[Tomcat].[localhost].[/]
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] w.s.c.ServletWebServerApplicationContext
2024-05-07T13:12:48.261+05:30  INFO 5204 ---  [                  main] w.s.c.ServletWebServerApplicationContext
Initialized Address
Initialized Laptop
Initialized Student
Address being created
Laptop being created
Student being created
Address id: 1
Address house no: 1
Address pincode: 1
Laptop id: 1
Laptop name: Lenovo
Laptop price: 1
Student id: 1
Student age: 21
Student name: Lucky
Student address: com.c.c.Address.Address@1
Student laptop: com.c.c.Laptop.Laptop@1
TestController being created
TestController1 instance initialization
User instance
User object hashCode: 1140822370
Student instance initialization
Student being created
Student object hashCode: 1944385571 User object hashCode: 1140822370
TestController2 instance initialization
TestController2 object hashCode: 1325214387 User object hashCode: 1140822370
Fetchuser api invoked
```

Sprigboot: Dynamic Bean Initialization

Concept && Coding YT Notes

Unsatisfied Dependency problem

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

    @Autowired
    Order order;

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

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

```
public interface Order {

    public void createOrder();
}
```

```
@Component
public class OnlineOrder implements Order{

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

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

```
@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'
```

Report Abuse

@Qualifier

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

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


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

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

```
public interface Order {

    public void createOrder();
}
```

```
@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");
    }
}
```

```
@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:

```
@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: "");
    }
}
```

```
public interface Order {

    public void createOrder();
}
```

```
@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");
    }
}
```

```
@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:

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

    @Autowired
    Order order;

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

        order.createOrder();

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

```
public interface Order {

    public void createOrder();
}
```

```
public class OnlineOrder implements Order{

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

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

```
public class OfflineOrder implements Order{

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

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

```
@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();
    }
}
```

Spring boot: @ConditionalOnProperty

@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

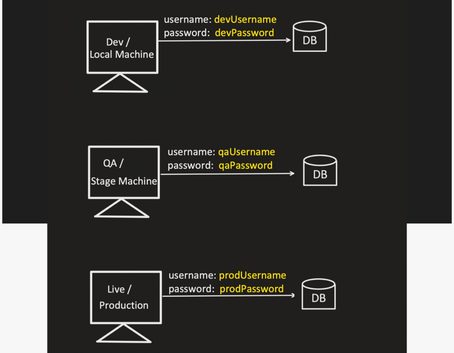
Disadvantages:

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

Report Abuse

Spring boot: @Profile

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



Report Abuse

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

application.properties

application-{profile1}.properties | application-{profile2}.properties | . . . . . | application-{profileN}.properties

```
@Component
public class MyDbConnection {

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

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

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

application.properties
```
username=defaultUsernamе
password=defaultPassword
```

application-dev.properties
```
username=devUsernamе
password=devPassword
```

application-qa.properties
```
username=qaUsernamе
password=qaPassword
```

application-prod.properties
```
username=prodUsernamе
password=prodPassword
```

```
2021-06-02T00:20:46.134+05:30  INFO 10524 ---[           main] m.c.c.s.MyDbConnectionApplication
2021-06-02T00:20:46.139+05:30  INFO 10524 ---[           main] m.c.c.s.MyDbConnectionApplication
2021-06-02T00:20:46.140+05:30  INFO 10524 ---[           main] m.c.c.s.MyDbConnectionApplication
2021-06-02T00:20:46.140+05:30  INFO 10524 ---[           main] m.c.c.s.MyDbConnectionApplication
2021-06-02T00:20:46.141+05:30  INFO 10524 ---[           main] m.c.c.s.MyDbConnectionApplication
2021-06-02T00:20:46.141+05:30  INFO 10524 ---[           main] m.c.c.s.MyDbConnectionApplication
2021-06-02T00:20:46.142+05:30  INFO 10524 ---[           main] m.c.c.s.MyDbConnectionApplication
```

Spring boot : AOP

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>
```

Some important AOP concepts :

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.concepts.learningspringboot.Employee.getHelloWorldMessage())")
   - () wildcard : matches any single item
     - Matches any return type
       @Before("execution(* com.concepts.learningspringboot.Employee.getHelloWorldMessage())")
       System.out.println("inside beforeMethod Aspect");
     - Matches any method with single parameter String
       @Before("execution(* com.concepts.learningspringboot.Employee.*(String))")
       System.out.println("inside beforeMethod Aspect");
     - Matches fetchEmployee method that take any single parameter
       @Before("execution(* com.concepts.learningspringboot.Employee.fetchEmployee(*))")
       System.out.println("inside beforeMethod Aspect");
   - (..) wildcard : matches 0 or More item
     - Matches fetchEmployee method that take any 0 or More parameters
       @Before("execution(public String com.concepts.learningspringboot.Employee.fetchEmployee(..))")
       System.out.println("inside beforeMethod Aspect");
     - Matches fetchEmployee method in 'com.concepts.mydomain' package and subpackage classes
       @Before("execution(public String com.concepts.mydomain.*.fetchEmployee())")
       System.out.println("inside beforeMethod Aspect");
     - Matches any method in 'com.concepts.mydomain' package and subpackage classes
       @Before("execution(* com.concepts.mydomain.*.*())")
       System.out.println("inside beforeMethod Aspect");

2. Within: matches all method within any class or groups.
   - This pointcut will run for each method in the class Employee
     @Before("within(com.concepts.mydomain.Employee)")
   - This pointcut will run for each method in this package and subpackage
     @Before("within(com.concepts.mydomain.*)")

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

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

5. Arg: matches any method with particular arguments (or parameters)
   @Before("arg(String)")

If instead of primitive type, we need object, then we can give like this
@Before("arg(com.concepts.mydomain.Employee)")

6. @arg: matches any method with particular parameters and that parameter class is annotated with particular annotation.
   @Before("@arg(org.springframework.stereotype.Service)")

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

interface

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

Combining two pointcuts using:
- && (boolean and)
- || (boolean or)

Named Pointcuts

Advice:
Its an action, which is taken @before or @After or @Around the method execution.

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

@Around:
As the name says, it surrounds the method execution before and after (with).

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 100s of pointcut, so whenever i invoke a method, does matching happens with 100s of pointcuts?

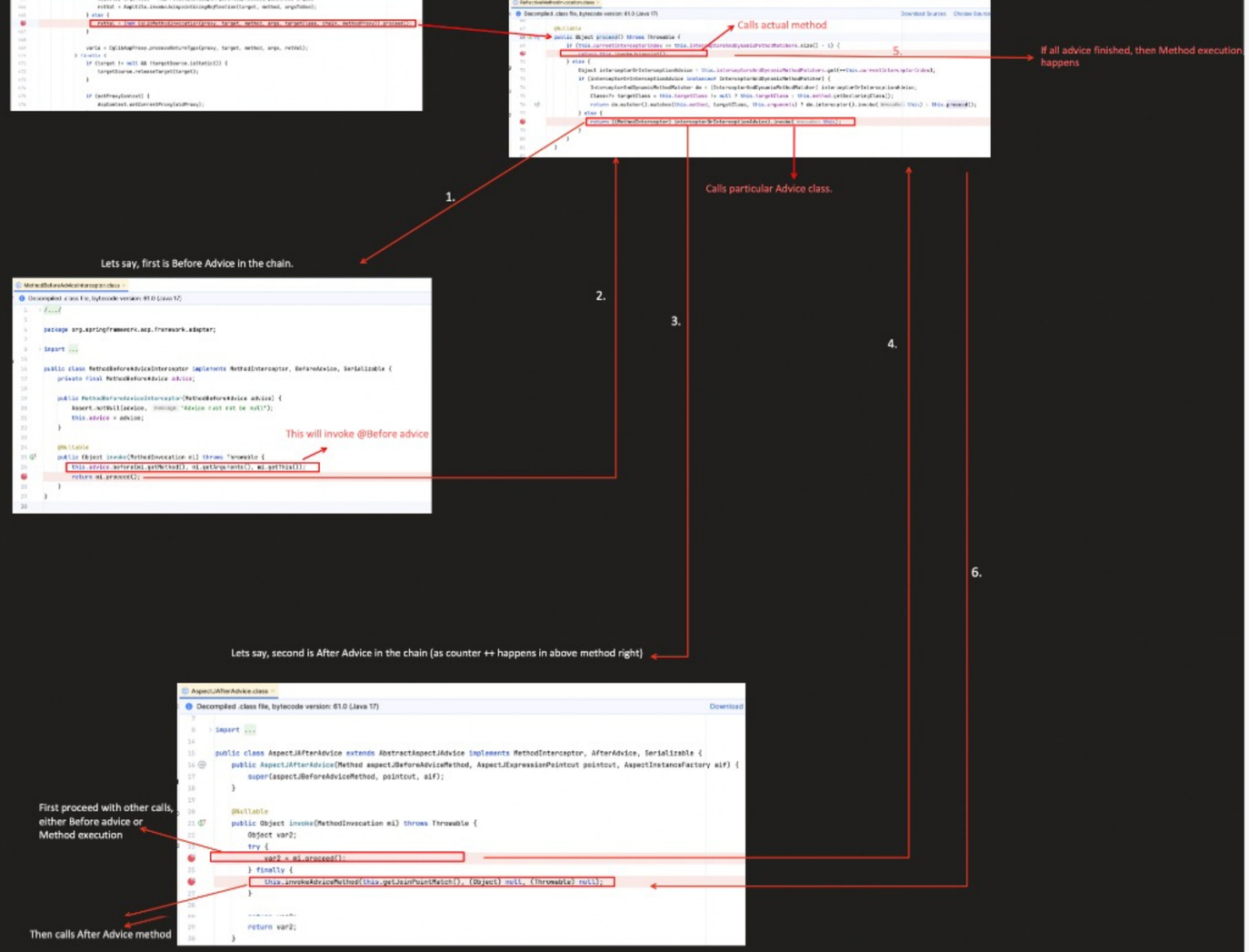
Let's understand the AOP flow, to get an answer of above doubts:

1. When Application startup happens
   - Look for @Aspect annotation Classes
   - Parse the Pointcut Expression
     - Store by **PointcutParser.java** class
   - Stored in a different data structure or cache after parsing.
   - Look for @Component, @Service @Controller etc.. annotation Classes
   - For each class, it checks if its eligible for interception based on pointcut expression
     - Done by **AbstractAutoProxyCreator.java** class

     If yes, it creates a Proxy using CGI Dynamic proxy or CGLIB proxy
     This proxy class, has code, which executes 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.

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

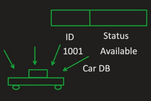

Calls actual method
If all advice finished, then Method execution happens
Calls particular Advice class.
1.
2.
3.
4.
5.
6.
Lets say, first is Before Advice in the chain.
This will invoke @Before advice
Lets say, second is After Advice in the chain (as counter ++ happens in above method right)
First proceed with other calls, either Before advice or Method execution
Then calls After Advice method

Spring boot: @Transactional (Part1)

Critical Section

Code segment, where shared resources are being accessed and modified.

{
Read Car Row with id: 1001
If Status is Available:
Update it to Booked
}



When multiple request try to access this critical section, Data Inconsistency can happen.

Its solution is usage of **TRANSACTION**

- It helps to achieve ACID property.

Report Abuse

A (Atomicity):
Ensures all operations within a transaction are completed successfully. If any operation fails, the entire transaction will get rollback.

C (Consistency):
Ensures that DB state before and after the transactions should be Consistent only.

I (Isolation):
Ensures that, even if multiple transactions are running in parallel, they do not interfere with each other.

Durability:
Ensures that committed transaction will never lost despite system failure or crash.

```
BEGIN_TRANSACTION:

        - Debit from A
        - Credit to B
        if all success:
                COMMIT;
        Else
                ROLLBACK;

END_TRANSACTION;
```

In Spring boot , we can use **@Transactional** annotation.

And for that:

1. we need to add below Dependency in pom.xml
   (based on DB we are using, suppose we are using RELATIONAL DB)

Spring boot Data JPA (Java persistence API): helps to interact with Relational databases without writing much code.

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

+

Database driver dependency is also required (that we will see in next topic)

2. Activate, Transaction Management by using **@EnableTransactionManagment** in main class.
   (spring boot generally Auto configure it, so we don't need to specially add it)

```
@SpringBootApplication
@EnableTransactionMangagment
public class SpringbootApplication {

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

@Transactional

At Class level

- **Transaction applied to all Public methods**

```
@Component
@Transactional
public class CarService {

    public void updateCar1(){
        //this method will get executed within a transaction
    }

    public void updateCarDetails2(){
        //this method will get executed within a transaction
    }

    private void helperMethod3(){
        //this method will not get affected by Transactional annotation.
    }
```

At Method level

- **Transaction applied to particular method only**

```
@Component
public class CarService {

    @Transactional
    public void updateCar1(){
        //this method will get executed within a transaction
    }

    public void updateDetailsCar2(){
        //this method will NOT be executed within a transaction
    }
}
```

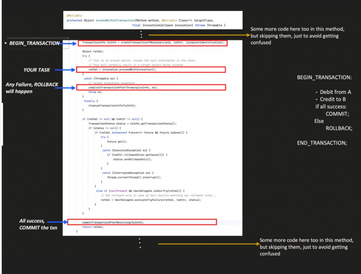
Transaction Management in Spring boot uses AOP.

1. **Uses Point cut expression to search for method, which has @Transactional annotation like:**
   @within(*org.springframework.transaction.annotation.Transactional)*

2. **Once Point cut expression matches, run an "Around" type Advice.**
   Advice is:
   invokeWithinTransaction method present present in ***TransactionalInterceptor*** class.

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

    @Autowired
    User user;

    @GetMapping(path = "/updateuser")
    public String updateUser(){
        user.updateUser();
        return "user is updated successfully";
    }
}
```

```
@Component
public class User {

    @Transactional
    public void  updateUser(){
        System.out.println("UPDATE QUERY TO update the user db values");
    }
}
```



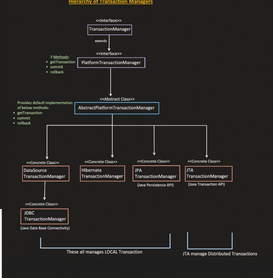
NOW, we know, how TRANSACTIONAL works, we will now see below topics in depth:

- **Transaction Context**
- **Transaction Manager**
  - Programmatic
  - Declarative
- **Propagation**
  - REQUIRED
  - REQUIRED_NEW
  - SUPPORTS
  - NOT_SUPPORTED
  - MANDATORY
  - NEVER
  - NESTED
- **Isolation level**
  - READ_UNCOMMITTED
  - READ_COMMITTED
  - REPEATABLE_READ
  - SERIALIZABLE
- **Configure Transaction Timeout**
- **What is Read only transaction**

  etc..

Spring boot: @Transactional (Part-2)

Hierarchy of Transaction Managers



Declarative

Transaction Management through Annotation

```
@Component
public class User {

    @Transactional
    public void updateUser(){
        System.out.println("UPDATE QUERY TO update the user db values");
    }
}
```

Here, based on underling DataSource used like JDBC or JPA etc. Spring boot will choose appropriate Transaction manager.

```
@Configuration
public class AppConfig {

    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("org.h2.Driver");
        dataSource.setUrl("jdbc:h2:mem:testdb");
        dataSource.setUsername("sa");
        dataSource.setPassword("");
        return dataSource;
    }

    @Bean
    public PlatformTransactionManager userTransactionManager(DataSource dataSource) {
        return new DataSourceTransactionManager(dataSource);
    }
}
```

```
@Component
public class UserDeclarative {

    @Transactional(transactionManager = "userTransactionManager")
    public void updateUserProgrammatic() {
        //SOME DB OPERATIONS
        System.out.println("Insert Query ran");
        System.out.println("Update Query ran");
    }
}
```

2 way to implement by Programmatic

Approach 1

```
@Service
public class UserService {
    private final TransactionTemplate transactionTemplate;
    private final JdbcTemplate jdbcTemplate;
    private final UserRepository userRepository;
    private final ObjectMapper objectMapper;
    private final ApplicationEventPublisher eventPublisher;
    private final ExecutorService executorService;
    private final Logger log = LoggerFactory.getLogger(UserService.class);

    public UserService(
            @Qualifier("userTransactionManager") PlatformTransactionManager transactionManager,
            JdbcTemplate jdbcTemplate,
            UserRepository userRepository,
            ObjectMapper objectMapper,
            ApplicationEventPublisher eventPublisher,
            @Qualifier("customExecutorService") ExecutorService executorService) {
        this.transactionTemplate = new TransactionTemplate(transactionManager);
        this.jdbcTemplate = jdbcTemplate;
        this.userRepository = userRepository;
        this.objectMapper = objectMapper;
        this.eventPublisher = eventPublisher;
        this.executorService = executorService;
        this.transactionTemplate.setIsolationLevel(TransactionDefinition.ISOLATION_READ_COMMITTED);
        this.transactionTemplate.setTimeout(5);
        this.transactionTemplate.setName("UserTxTemplate");
        this.transactionTemplate.setReadOnly(false);
        log.info("UserService initialized with transaction template");
    }

    public void updateUserProgrammaticAsync(User user) {
        executorService.submit(() -> {
            long threadId = Thread.currentThread().getId();
            log.info("Thread {} starting async update for user: {}", threadId, user.getUsername());
            try {
                transactionTemplate.execute(status -> {
                    log.info("Thread {} inside transaction template", threadId);
                    boolean isNewTx = !status.isTransactionActive();
                    log.info("Thread {} is new transaction: {}", threadId, isNewTx);
                    User savedUser = userRepository.save(user);
                    log.info("Thread {} saved user with id: {}", threadId, savedUser.getId());
                    String jsonResult = objectMapper.writeValueAsString(savedUser);
                    log.info("Thread {} serialized user to JSON: {}", threadId, jsonResult);
                    eventPublisher.publishEvent(new UserCreatedEvent(this, savedUser));
                    log.info("Thread {} published UserCreatedEvent", threadId);
                    return null;
                });
                log.info("Thread {} transaction completed successfully", threadId);
            } catch (Exception e) {
                log.error("Thread {} transaction failed: {}", threadId, e.getMessage(), e);
                e.printStackTrace();
            }
        });
    }
}
```

```
@Component
public class UserDeclarative {

    @Transactional(transactionManager = "userTransactionManager")
    public void updateUserProgrammatic() {
        //SOME DB OPERATIONS
        System.out.println("Insert Query ran");
        System.out.println("Update Query ran");
    }
}
```

Approach 2

Usage of TransactionTemplate

```
@Service
public class UserDeclarative {

    @Transactional(transactionManager = "userTransactionManager")
    public void updateUserProgrammatic() {
        //SOME DB OPERATIONS
        System.out.println("Insert Query ran");
        System.out.println("Update Query ran");
    }
}
```

```
@Service
public class UserTransactionTemplate {

    TransactionTemplate transactionTemplate;

    public UserTransactionTemplate(UserTransactionManager transactionManager) {
        this.transactionTemplate = new TransactionTemplate(transactionManager);
    }

    public void updateUserProgrammatic() {
        transactionTemplate.execute(status -> {
            boolean isNewTx = !status.isTransactionActive();
            System.out.println("Thread {} is new transaction: {}", isNewTx);
            User savedUser = userRepository.save(user);
            System.out.println("Thread {} saved user with id: {}", savedUser.getId());
            String jsonResult = objectMapper.writeValueAsString(savedUser);
            System.out.println("Thread {} serialized user to JSON: {}", jsonResult);
            eventPublisher.publishEvent(new UserCreatedEvent(this, savedUser));
            System.out.println("Thread {} published UserCreatedEvent");
            return null;
        });
        System.out.println("Thread {} transaction completed successfully");
    }
}
```

Now, lets see Propagation

When we try to create a new Transaction, it first check the PROPAGATION value set, and this tell whether we have to create new transaction or not.

- REQUIRED (default propagation):

```
@Transactional(propagation=Propagation.REQUIRED)
    if(parent txn present)
        Use it;
    else
        Create new transaction;
```

- REQUIRED_NEW:

```
@Transactional(propagation=Propagation.REQUIRED_NEW)
    if(parent txn present)
        Suspend the parent txn;
        Create a new Txn and once finished;
        Resume the parent txn;
    else
        Create new transaction and execute the method;
```

- SUPPORTS:

```
@Transactional(propagation=Propagation.SUPPORTS)
    if(parent txn present)
        Use it;
    Else
        Execute the method without any transaction;
```

- NOT_SUPPORTED:

```
@Transactional(propagation=Propagation.NOT_SUPPORTED)
    if(parent txn present)
        Suspend the parent txn;
        Execute the method without any transaction;
        Resume the parent txn;
    else
        Execute the method without any transaction;
```

- MANDATORY:

```
@Transactional(propagation=Propagation.MANDATORY)
    if(parent txn present)
        Use it;
    Else
        Throw exception;
```

- NEVER:

```
@Transactional(propagation=Propagation.MANDATORY)
    if(parent txn present)
        Throw exception;
    Else
        Execute the method without any transaction;
```

Declarative way of usage:

```
@Service
public class UserDeclarative {

    @Autowired
    UserRepository user;

    @Transactional
    public void updateUser() {
        System.out.println("Is transaction active: " + TransactionSynchronizationManager.isActualTransactionActive());
        System.out.println("Current transaction name: " + TransactionSynchronizationManager.getCurrentTransactionName());

        System.out.println("Some initial DB operation");
        user.save();
        System.out.println("Some final DB operation");
    }

    public void updateUserFromNoTransactionalMethod() {
        System.out.println("Is transaction active: " + TransactionSynchronizationManager.isActualTransactionActive());
        System.out.println("Current transaction name: " + TransactionSynchronizationManager.getCurrentTransactionName());
        System.out.println("Some initial DB operation");
        user.save();
        System.out.println("Some final DB operation");
    }
}
```

```
@Component
public class UserNever {

    PlatformTransactionManager userTransactionManager;

    UserNever(PlatformTransactionManager userTransactionManager) {
        this.userTransactionManager = userTransactionManager;
    }

    void
    // Ignored tx present
    // Use it
    // else
    //   Create one

    public void doSomethingWithPropagationOnly(PropagationCallbackWithOutTx callback) {
        DefaultTransactionDefinition transactionDefinition = new DefaultTransactionDefinition();
        transactionDefinition.setName("NEVER_TEST");
        transactionDefinition.setPropagationBehavior(TransactionDefinition.PROPAGATION_NEVER);
        TransactionStatus status = userTransactionManager.getTransaction(transactionDefinition);
        try {
            // Execute operation
            boolean isNewTx = !status.isTransactionActive();
            System.out.println("Transaction NEVER: Is transaction active: " + TransactionSynchronizationManager.isActualTransactionActive());
            System.out.println("Transaction NEVER: Current transaction name: " + TransactionSynchronizationManager.getCurrentTransactionName());
            System.out.println("==========================");
            callback.execute();
            userTransactionManager.commit(status);
        } catch (Exception ex) {
            userTransactionManager.rollback(status);
            throw ex;
        }
    }
}
```

(Approach 2) using TransactionTemplate

```
@Service
public class UserService {
    private final TransactionTemplate transactionTemplate;
    private final JdbcTemplate jdbcTemplate;
    private final UserRepository userRepository;
    private final ObjectMapper objectMapper;
    private final ApplicationEventPublisher eventPublisher;
    private final ExecutorService executorService;
    private final Logger log = LoggerFactory.getLogger(UserService.class);

    public UserService(
            @Qualifier("userTransactionManager") PlatformTransactionManager transactionManager,
            JdbcTemplate jdbcTemplate,
            UserRepository userRepository,
            ObjectMapper objectMapper,
            ApplicationEventPublisher eventPublisher,
            @Qualifier("customExecutorService") ExecutorService executorService) {
        this.transactionTemplate = new TransactionTemplate(transactionManager);
        this.jdbcTemplate = jdbcTemplate;
        this.userRepository = userRepository;
        this.objectMapper = objectMapper;
        this.eventPublisher = eventPublisher;
        this.executorService = executorService;
        this.transactionTemplate.setIsolationLevel(TransactionDefinition.ISOLATION_READ_COMMITTED);
        this.transactionTemplate.setTimeout(5);
        this.transactionTemplate.setName("UserTxTemplate");
        this.transactionTemplate.setReadOnly(false);
        log.info("UserService initialized with transaction template");
    }

    public void updateUserProgrammaticAsync(User user) {
        executorService.submit(() -> {
            long threadId = Thread.currentThread().getId();
            log.info("Thread {} starting async update for user: {}", threadId, user.getUsername());
            try {
                transactionTemplate.execute(status -> {
                    log.info("Thread {} inside transaction template", threadId);
                    boolean isNewTx = !status.isTransactionActive();
                    log.info("Thread {} is new transaction: {}", threadId, isNewTx);
                    User savedUser = userRepository.save(user);
                    log.info("Thread {} saved user with id: {}", threadId, savedUser.getId());
                    String jsonResult = objectMapper.writeValueAsString(savedUser);
                    log.info("Thread {} serialized user to JSON: {}", threadId, jsonResult);
                    eventPublisher.publishEvent(new UserCreatedEvent(this, savedUser));
                    log.info("Thread {} published UserCreatedEvent", threadId);
                    return null;
                });
                log.info("Thread {} transaction completed successfully", threadId);
            } catch (Exception e) {
                log.error("Thread {} transaction failed: {}", threadId, e.getMessage(), e);
                e.printStackTrace();
            }
        });
    }
}
```

```
@Service
public class UserTxTemplate {
    TransactionTemplate transactionTemplate;

    public UserTxTemplate(
            @Qualifier("userTransactionManager")
            PlatformTransactionManager transactionManager) {
        this.transactionTemplate = new TransactionTemplate(transactionManager);
        this.transactionTemplate.setIsolationLevel(TransactionDefinition.ISOLATION_READ_COMMITTED);
        this.transactionTemplate.setTimeout(5);
        this.transactionTemplate.setName("UserTxTemplate");
        this.transactionTemplate.setReadOnly(false);
    }

    public void updateUserProgrammatic() {
        transactionTemplate.execute(status -> {
            boolean isNewTx = !status.isTransactionActive();
            System.out.println("Thread {} is new transaction: {}", isNewTx);
            User savedUser = userRepository.save(user);
            System.out.println("Thread {} saved user with id: {}", savedUser.getId());
            String jsonResult = objectMapper.writeValueAsString(savedUser);
            System.out.println("Thread {} serialized user to JSON: {}", jsonResult);
            eventPublisher.publishEvent(new UserCreatedEvent(this, savedUser));
            System.out.println("Thread {} published UserCreatedEvent");
            return null;
        });
        System.out.println("Thread {} transaction completed successfully");
    }
}
```

Output:

```
Is transaction active: true
Current transaction name: com.consultantbuddy.learnspringboot.TransactionManagement.UserDeclarative.updateUser
Some initial DB operation
==========================
com.consultantbuddy.learnspringboot.
Propagation.REQUIRED: Is  transaction active: true
Propagation.REQUIRED: Current transaction name: com.consultantbuddy.learnspringboot.TransactionManagement.UserDeclarative.updateUser
==========================
Some final DB operation
```

```
Is transaction active: false
Current transaction name: null
Some initial DB operation
==========================
Propagation.REQUIRED: Is  transaction active: true
Propagation.REQUIRED: Current transaction name: TRANSACTION TEMPLATE REQUIRED PROPAGATION
==========================
Some final DB operation
```

Spring boot - @Transaction Part3 (Isolation Level)

Isolation Level

Isolation level:
It tells, how the changes made by one transaction are visible to other transactions running in parallel.

```
@Transactional(propagation = Propagation.REQUIRED, isolation = Isolation.READ_COMMITTED)
public void updateUser() {

    //some operations here
}
```

Isolation Level	Dirty Read Possible	Non-Repeatable Read Possible	Phantom Read Possible
READ_UNCOMMITTED	Yes	Yes	Yes
READ_COMMITTED	No	Yes	Yes
REPEATABLE_COMMITTED	No	No	Yes
SERIALIZABLE	No	No	No

Concurrency High

Concurrency Low

Default isolation level, depends upon the DB which we are using.
Like Most relational Databases uses READ_COMMITTED as default isolation, but again it depends upon DB to DB.

Report Abuse

Dirty Read Problem

Transaction A reads the un-committed data of other transaction.
and
If other transaction get ROLLED BACK, the un-committed data which is read by Transaction A is known as Dirty Read.

Time	Transaction A	Transaction B	DB Status
T1	BEGIN_TRANSACTION	BEGIN_TRANSACTION	Id: 123 Status: free
T2		Update Row id:123 Status = booked	Id: 123 Status: booked (Not Committed by Transaction B yet)
T3	Read Row id:123 (Got status = booked)		Id: 123 Status: booked (Not Committed by Transaction B yet)
T4		Rollback	Id: 123 Status: Free (Un-committed changes of Txn B got Rolled Back)

Non-Repeatable Read Problem

If suppose Transaction A, reads the same row several times and there is a chance that it get different value, then its known as Non-Repeatable Read problem.

	Transaction A	DB
T1	BEGIN_TRASACTION	ID: 1 Status: Free
T2	Read Row ID:1 (reads status: **Free**)	ID: 1 Status: Free
T3		ID: 1 Status: Booked
T4	Read Row ID:1 (reads status: **Booked**)	ID: 1 Status: Booked
T5	COMMIT	

Some other Transaction changed and committed the changes.

Phantom Read Problem

If suppose Transaction A, executes same query several times but there is a chance that rows returned are different.
Than its known as Phantom Read problem.

	Transaction A	DB
T1	BEGIN_TRASACTION	ID: 1, Status: Free ID: 3, Status: Booked
T2	Read Row where ID>0 and ID<5 (reads 2 rows ID:1 and ID:3)	ID: 1, Status: Free ID: 3, Status: Booked
T3		ID: 1, Status: Free ID:2, Status: Free ID: 3, Status: Booked
T4	Read Row where ID>0 and ID<5 (reads 3rows ID:1, ID:2 and ID:3)	ID: 1, Status: Free ID:2, Status: Free ID: 3, Status: Booked
T5	COMMIT	

Some other Transaction Inserted the row with ID:2 and Committed

DB Locking Types

Locking make sure that, no other transaction update the locked rows.

Lock Type	Another Shared Lock	Another Exclusive Lock
Have Shared Lock	Yes	NO
Have Exclusive Lock	NO	NO

- Shared Lock (S) also knows as READ LOCK.
- Exclusive Lock(X) also know as WRITE LOCK.

Let's see this table again:

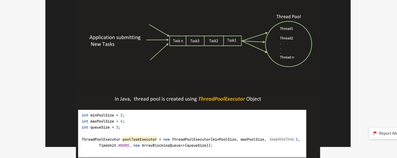
Isolation Level	Dirty Read Possible	Non-Repeatable Read Possible	Phantom Read Possible
READ_UNCOMMITTED	Yes	Yes	Yes
READ_COMMITTED	No	Yes	Yes
REPEATABLE_COMMITTED	No	No	Yes
SERIALIZABLE	No	No	No

ISOLATION LEVEL	Locking Strategy
Read Uncommitted	**Read** : No Lock acquired **Write**: No Lock acquired
Read Committed	**Read**: Shared Lock acquired and Released as soon as Read is done **Write**: Exclusive Lock acquired and keep till the end of the transaction
Repeatable Read	**Read**: Shared Lock acquired and Released only at the end of the Transaction **Write**: Exclusive Lock acquired and Released only at the end of the Transaction
Serializable	Same as Repeatable Read Locking Strategy + apply Range Lock and lock is release only at the end of the Transaction.

Spring boot - Async Annotation (Part1)

What is ThreadPool:

- It's a collection of threads (aka workers), which are available to perform the submitted tasks.
- Once task completed, worker thread get back to Thread Pool and wait for new task to assigned.
- Means threads can be reused.



In Java, thread pool is created using *ThreadPoolExecutor* Object

```
int minPoolSize = 2;
int maxPoolSize = 4;
int queueSize = 3;

ThreadPoolExecutor poolTaskExecutor = new ThreadPoolExecutor(minPoolSize, maxPoolSize, 1minutes, TimeUnit.HOURS, new ArrayBlockingQueue<>(queueSize));
```

Report Abuse

Now lets start understanding, how @Async Annotation works

Async Annotation

- Used to mark method that should run asynchronously.
- Runs in a new thread, without blocking the main thread.

Example:

```
@RestController
public class AsyncMethodTest {
    @Autowired
    private ExecutorService executorService = Executors.newFixedThreadPool(2);

    @Async
    @GetMapping("/asyncmethodtest")
    public void asyncMethodTest() {
        System.out.println("inside asyncMethodTest: " + Thread.currentThread().getName());
        return;
    }
}
```

```
@SpringBootApplication
@EnableAsync
public class SpringbootAsyncApplication {

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

```
curl -X GET localhost:8080/asyncmethodtest
```

```
2024-08-11T15:27:22.905+05:30  INFO 14004 --- [nio-8080-exec-2] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-08-11T15:27:22.906+05:30  INFO 14004 --- [nio-8080-exec-2] o.springframework.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-08-11T15:27:22.907+05:30  INFO 14004 --- [nio-8080-exec-2] o.springframework.web.servlet.DispatcherServlet : Completed initialization in 1 ms
inside asyncMethodTest: Thread-2
```

Output:

```
2024-08-11T15:27:22.905+05:30  INFO 14004 --- [nio-8080-exec-2] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-08-11T15:27:22.906+05:30  INFO 14004 --- [nio-8080-exec-2] o.springframework.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-08-11T15:27:22.907+05:30  INFO 14004 --- [nio-8080-exec-2] o.springframework.web.servlet.DispatcherServlet : Completed initialization in 1 ms
inside asyncMethodTest: task-1
inside getUserMethod: http-nio-8080-exec-2
inside asyncMethodTest: task-2
inside getUserMethod: http-nio-8080-exec-3
inside asyncMethodTest: task-3
inside getUserMethod: http-nio-8080-exec-4
inside asyncMethodTest: task-4
inside getUserMethod: http-nio-8080-exec-5
inside asyncMethodTest: task-5
inside getUserMethod: http-nio-8080-exec-6
inside asyncMethodTest: task-6
inside getUserMethod: http-nio-8080-exec-7
inside asyncMethodTest: task-7
inside getUserMethod: http-nio-8080-exec-8
inside asyncMethodTest: task-8
```

Creating new thread, each time we run

So, how does this "Async" Annotation, creates a new thread?

Many places you will find, which says

Spring boot uses by default "***SimpleAsyncTaskExecutor***", which creates new thread every time.

I will say, this is not fully correct answer.

So, what's the right answer, What's the default Executor Spring boot uses?

If we see below Spring boot framework code, it first looks for *defaultExecutor*, if no *defaultExecutor* found, only then *SimpleAsyncTaskExecutor* is used.

AsyncExecutionInterceptor.java

```
@Override
protected Executor getDefaultExecutor(BeanFactory beanFactory) {
    Executor defaultExecutor = super.getDefaultExecutor(beanFactory);
    return (defaultExecutor != null ? defaultExecutor : new SimpleAsyncTaskExecutor());
}
```

UseCase1:

```
@ConfigurationProperties
@EnableAsyncs
public class AsyncConfig {
}
```

```
@SpringBootApplication
@EnableAsyncs
public class SpringbootAsyncApplication {

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

```
@RestController
public class UserService {

    @Async
    public void asyncMethodTest() {
        System.out.println("inside asyncMethodTest: " + Thread.currentThread().getName());
    }
}
```

During Application startup, Spring boot sees that, no **ThreadPoolTaskExecutor** Bean present, so it creates its default "**ThreadPoolTaskExecutor**" with below configurations.

```
defaultExecutor = ThreadPoolTaskExecutor@7871
poolSizeMonitor = ChildExec@7872
corePoolSize = 8
maxPoolSize = 2147483647
keepAliveSeconds = 60
queueCapacity = 2147483647
```

ThreadPoolTaskExecutor is nothing but a Spring boot Object, which is just a wrapper around Java ThreadPoolExecutor.

ThreadPoolTaskExecutor.java

```
public class ThreadPoolTaskExecutor extends CustomizableThreadFactory implements AsyncTaskExecutor, SchedulingTaskExecutor {
    private boolean prefersShortLivedTasks = true;
    private int threadPriority = Thread.NORM_PRIORITY;
    private boolean daemon = false;
    private int corePoolSize = 1;
    private int maxPoolSize = Int.MAX_VALUE;
    private int keepAliveSeconds = 60;
    private int queueCapacity = Int.MAX_VALUE;
    private boolean allowCoreThreadTimeOut = false;
    private int awaitTerminationSeconds = 0;
    private String beanName;

    private ExecutorConfigurationSupport threadPoolExecutor;

    @Override
    public void execute(Runnable task, long startTimeout) {
        ExecutorService executor = getThreadPoolExecutor();
        if (!(executor instanceof AsyncListenableTaskExecutor ||
                executor instanceof AsyncTaskExecutor)) {
            log.warn("Spring async task executor [" + executor + "] does not support timeout for task [" + task + "] - falling back to execute");
        }
        executor.execute(task);
    }

    @Override
    public void execute(Runnable task) {
        execute(task, TIMEOUT_INDEFINITE);
    }

    @Override
    public void shutdown() {
        if (logger.isInfoEnabled()) {
            logger.info("Shutdown executor");
        }
        ExecutorService executor = this.threadPoolExecutor;
        if (executor != null) {
            boolean beforeShutdown = (this.threadPoolExecutor == executor);
            this.threadPoolExecutor = null;
            int1 1 = (beforeShutdown ? executor.shutdownNow().size() : 0);
            int2 2 = 0;
            boolean interrupted = false;
            int3 3 = this.awaitTerminationSeconds;
            while (3 > 0) {
                ...
                ...
            }
            if (1 > 0 && logger.isWarnEnabled()) {
                logger.warn("Shutdown " + 1 + " runnable" + (1 > 1 ? "s" : "") +
                        " from " + this);
            }
            else if (2 > 0 && logger.isInfoEnabled()) {
                logger.info("Shutdown " + 2 + " runnable" + (2 > 1 ? "s" : "") +
                        " from " + this);
            }
            else if (logger.isDebugEnabled()) {
                logger.debug("Shutdown " + this);
            }
        }
    }

    protected ExecutorService getThreadPoolExecutor() throws IllegalStateException {
        Assert.state(this.threadPoolExecutor != null, "EmbeddedValueResolver not initialized yet");
        return this.threadPoolExecutor;
    }

    protected ExecutorService initializeExecutor(
            ThreadFactory threadFactory, RejectedExecutionHandler rejectedExecutionHandler) {

        BlockingQueue queue = createQueue(this.queueCapacity);
        ThreadPoolExecutor executor = new ThreadPoolExecutor(
                this.corePoolSize, this.maxPoolSize, this.keepAliveSeconds, TimeUnit.SECONDS,
                queue, threadFactory, rejectedExecutionHandler);
        if (this.allowCoreThreadTimeOut) {
            executor.allowCoreThreadTimeOut(true);
        }

        this.threadPoolExecutor = executor;
        return executor;
    }

    protected BlockingQueue<Runnable> createQueue(int queueCapacity) {
        if (logger.isDebugEnabled()) {
            logger.debug("Creating ExecutorBlockingQueue with capacity " + queueCapacity);
        }
        return new ExecutorBlockingQueue(queueCapacity);
    }

    public void setThreadPriority(int threadPriority) {
        this.threadPriority = threadPriority;
    }

    public void setDaemon(boolean daemon) {
        this.daemon = daemon;
    }

    public void setCorePoolSize(int corePoolSize) {
        this.corePoolSize = corePoolSize;
    }

    public void setMaxPoolSize(int maxPoolSize) {
        this.maxPoolSize = maxPoolSize;
    }

    public void setKeepAliveSeconds(int keepAliveSeconds) {
        this.keepAliveSeconds = keepAliveSeconds;
    }

    public void setQueueCapacity(int queueCapacity) {
        this.queueCapacity = queueCapacity;
    }

    public void setAllowCoreThreadTimeOut(boolean allowCoreThreadTimeOut) {
        this.allowCoreThreadTimeOut = allowCoreThreadTimeOut;
    }

    public void setAwaitTerminationSeconds(int awaitTerminationSeconds) {
        this.awaitTerminationSeconds = awaitTerminationSeconds;
    }

    public void setBeanName(String beanName) {
        this.beanName = beanName;
    }

    @Override
    public void afterPropertiesSet() {
        ExecutorService executor = initializeExecutor(getThreadFactory(), getRejectedExecutionHandler());
        this.threadPoolExecutor = executor;
    }

    @Override
    public void destroy() {
        shutdown();
    }
}
```

And its not recommended at all, why?

1. Underutilization of Threads: With Fixed Min pool size and Unbounded Queue(size is too big), its possible that most of the tasks will sit in the queue rather than creating new thread.
2. High Latency: Since queue size is too big, tasks will queue up till queue is not fill, high latency might occur during high load.
3. Thread Exhaustion: Lets say, if Queue also get filled up, then Executor will try to create new threads till Max pools size is not reached, which is integer.MAX_VALUE. This can lead to thread exhaustion. And server might go down because of overhead of managing so many threads.
4. High Memory Usage: Each threads need some memory, when we are creating these many threads, which may consume large amount of memory too, which might lead to performance degradation too.

UseCase2: Creating our own custom, ThreadPoolTaskExecutor

```
@ConfigurationProperties
@EnableAsyncs
public class SpringbootAsyncApplication {

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

```
@Configuration
public class AsyncConfig {

    @Bean("myThreadPoolTaskExecutor")
    public ThreadPoolTaskExecutor myThreadPoolTaskExecutor() {
        ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        executor.setCorePoolSize(2);
        executor.setMaxPoolSize(4);
        executor.setQueueCapacity(3);
        executor.setThreadNamePrefix("myThreadPoolTaskExecutor-");
        executor.setRejectedExecutionHandler(new ThrowsRejectedExecutionException());
        executor.setWaitForTasksToCompleteOnShutdown(true);
        executor.setAwaitTerminationSeconds(10);
        return executor;
    }
}
```

```
@RestController
public class UserService {

    @Async("myThreadPoolTaskExecutor")
    public void asyncMethodTest() {
        System.out.println("inside asyncMethodTest: " + Thread.currentThread().getName());
    }
}
```

or

```
@RestController
public class UserService {

    @Async
    public void asyncMethodTest() {
        System.out.println("inside asyncMethodTest: " + Thread.currentThread().getName());
    }
}
```

During Application startup, Spring boot sees that, **ThreadPoolTaskExecutor** Bean present, so it makes it default only.

*And even when we use @Async without any name, our "**myThreadPoolExecutor**" will get picked only.*

Output:

```
2024-08-12T15:02:35.102+05:30  INFO 17152 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-08-12T15:02:35.103+05:30  INFO 17152 --- [nio-8080-exec-1] o.springframework.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-08-12T15:02:35.104+05:30  INFO 17152 --- [nio-8080-exec-1] o.springframework.web.servlet.DispatcherServlet : Completed initialization in 1 ms
inside asyncMethodTest: myThreadPoolTaskExecutor-1
inside asyncMethodTest: myThreadPoolTaskExecutor-2
inside asyncMethodTest: myThreadPoolTaskExecutor-3
inside asyncMethodTest: http-nio-8080-exec-1
inside asyncMethodTest: http-nio-8080-exec-2
inside asyncMethodTest: myThreadPoolTaskExecutor-4
inside asyncMethodTest: http-nio-8080-exec-3
inside asyncMethodTest: myThreadPoolTaskExecutor-5
inside asyncMethodTest: myThreadPoolTaskExecutor-6
inside asyncMethodTest: http-nio-8080-exec-4
inside asyncMethodTest: myThreadPoolTaskExecutor-7
inside asyncMethodTest: http-nio-8080-exec-5
inside asyncMethodTest: myThreadPoolTaskExecutor-8
inside asyncMethodTest: http-nio-8080-exec-6
inside asyncMethodTest: myThreadPoolTaskExecutor-9
inside asyncMethodTest: http-nio-8080-exec-7
inside asyncMethodTest: myThreadPoolTaskExecutor-10
inside asyncMethodTest: http-nio-8080-exec-8
```

Output: (after putting sleep in async method, to simulate load)

```
@Configuration
@EnableAsyncs
public class AsyncConfig {

    @Bean("myThreadPoolTaskExecutor")
    public ThreadPoolTaskExecutor myThreadPoolTaskExecutor() {
        ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        executor.setCorePoolSize(2);
        executor.setMaxPoolSize(4);
        executor.setQueueCapacity(3);
        executor.setThreadNamePrefix("myThreadPoolTaskExecutor-");
        executor.setRejectedExecutionHandler(new ThrowsRejectedExecutionException());
        executor.setWaitForTasksToCompleteOnShutdown(true);
        executor.setAwaitTerminationSeconds(10);
        return executor;
    }
}
```

```
2024-08-12T15:07:31.112+05:30  INFO 17152 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
inside asyncMethodTest: myThreadPoolTaskExecutor-1
inside asyncMethodTest: myThreadPoolTaskExecutor-2
inside asyncMethodTest: myThreadPoolTaskExecutor-3
inside asyncMethodTest: http-nio-8080-exec-1
inside asyncMethodTest: http-nio-8080-exec-2
inside asyncMethodTest: http-nio-8080-exec-3
inside asyncMethodTest: myThreadPoolTaskExecutor-4
inside asyncMethodTest: myThreadPoolTaskExecutor-5
2024-08-12T15:07:31.112+05:30  INFO 17152 --- [nio-8080-exec-1] o.springframework.web.servlet.DispatcherServlet : Completed initialization in 1 ms
inside asyncMethodTest: myThreadPoolTaskExecutor-6
inside asyncMethodTest: myThreadPoolTaskExecutor-7
inside asyncMethodTest: myThreadPoolTaskExecutor-8
inside asyncMethodTest: myThreadPoolTaskExecutor-9
2024-08-12T15:07:31.112+05:30  INFO 17152 --- [nio-8080-exec-1] com.sun.jersey.spi.container.ContainerResponse : Exception thrown by rollback: com.sun.jersey.spi.container.ContainerResponseWriterException: java.io.IOException: Broken pipe
```

Queue get full

Max Thread reached

New Threads in Capacity

Max Thread, till Max Pool

Any New Request get Rejected

Its recommended, as its solve all the issues existing with the previous use case

UseCase3: Creating our own custom, ThreadPoolExecutor (java one)

```
@ConfigurationProperties
@EnableAsyncs
public class SpringbootAsyncApplication {

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

```
@Configuration
@EnableAsyncs
public class AsyncConfig {

    @Bean("myThreadPoolExecutor")
    public ThreadPoolExecutor myThreadPoolExecutor() {
        int minPoolSize = 2;
        int maxPoolSize = 4;
        int queueSize = 3;

        ThreadPoolExecutor poolTaskExecutor = new ThreadPoolExecutor(minPoolSize, maxPoolSize, 1minutes,
                TimeUnit.HOURS, new ArrayBlockingQueue<>(queueSize));
        return poolTaskExecutor;
    }
}
```

```
@RestController
public class UserService {

    @Async("myThreadPoolExecutor")
    public void asyncMethodTest() {
        System.out.println("inside asyncMethodTest: " + Thread.currentThread().getName());
    }
}
```

During Application startup, Spring boot sees that, **ThreadPoolExecutor (java one)** Bean is present, so it do not create its own default **ThreadPoolTaskExecutor (spring wrapper one)** instead it set the default executor is "**SimpleAsyncTaskExecutor**"

Now, if we run the above code, what we see.

```
2024-08-12T15:12:52.024+05:30  INFO 18012 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-08-12T15:12:52.024+05:30  INFO 18012 --- [nio-8080-exec-1] o.springframework.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-08-12T15:12:52.024+05:30  INFO 18012 --- [nio-8080-exec-1] o.springframework.web.servlet.DispatcherServlet : Completed initialization in 1 ms
inside asyncMethodTest: SimpleAsyncTaskExecutor-2
inside asyncMethodTest: SimpleAsyncTaskExecutor-3
inside asyncMethodTest: SimpleAsyncTaskExecutor-4
inside asyncMethodTest: http-nio-8080-exec-1
inside asyncMethodTest: http-nio-8080-exec-2
inside asyncMethodTest: SimpleAsyncTaskExecutor-5
inside asyncMethodTest: http-nio-8080-exec-3
inside asyncMethodTest: SimpleAsyncTaskExecutor-6
inside asyncMethodTest: http-nio-8080-exec-4
inside asyncMethodTest: SimpleAsyncTaskExecutor-7
inside asyncMethodTest: http-nio-8080-exec-5
inside asyncMethodTest: SimpleAsyncTaskExecutor-8
inside asyncMethodTest: http-nio-8080-exec-6
inside asyncMethodTest: SimpleAsyncTaskExecutor-9
inside asyncMethodTest: http-nio-8080-exec-7
inside asyncMethodTest: SimpleAsyncTaskExecutor-10
inside asyncMethodTest: http-nio-8080-exec-8
```

And its not recommended at all to use **"SimpleAsyncTaskExecutor"**, why?

It just creates new thread every time. So it may lead to

1. Thread Exhaustion: just blindly creating new thread with every Async request, might lead up to thread exhaustion.
2. Thread Creation Overhead: Since Threads are not reused, so thread management (creation, destroying) is an additional overhead.
3. High Memory Usage: Each threads need some memory, when we are creating these many threads, which may consume large amount of memory too, which might lead to performance degradation too.

So, whenever we have defined our own **ThreadPoolExecutor** (java one), always specify the name also with Async annotation.

```
@RestController
public class UserService {

    @Async("myThreadPoolExecutor")
    public void asyncMethodTest() {
        System.out.println("inside asyncMethodTest: " + Thread.currentThread().getName());
    }
}
```

```
@Configuration
@EnableAsyncs
public class AsyncConfig {

    @Bean("myThreadPoolExecutor")
    public ThreadPoolExecutor myThreadPoolExecutor() {
        int minPoolSize = 2;
        int maxPoolSize = 4;
        int queueSize = 3;

        ThreadPoolExecutor poolTaskExecutor = new ThreadPoolExecutor(minPoolSize, maxPoolSize, 1minutes,
                TimeUnit.HOURS, new ArrayBlockingQueue<>(queueSize));
        return poolTaskExecutor;
    }
}
```

Output:

```
2024-08-12T15:12:52.024+05:30  INFO 18012 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
inside asyncMethodTest: myThreadPoolExecutor-1
inside asyncMethodTest: myThreadPoolExecutor-2
inside asyncMethodTest: myThreadPoolExecutor-3
inside asyncMethodTest: http-nio-8080-exec-1
inside asyncMethodTest: http-nio-8080-exec-2
inside asyncMethodTest: myThreadPoolExecutor-4
inside asyncMethodTest: http-nio-8080-exec-3
inside asyncMethodTest: myThreadPoolExecutor-5
inside asyncMethodTest: myThreadPoolExecutor-6
inside asyncMethodTest: http-nio-8080-exec-4
inside asyncMethodTest: myThreadPoolExecutor-7
inside asyncMethodTest: http-nio-8080-exec-5
inside asyncMethodTest: myThreadPoolExecutor-8
inside asyncMethodTest: http-nio-8080-exec-6
inside asyncMethodTest: myThreadPoolExecutor-9
inside asyncMethodTest: http-nio-8080-exec-7
inside asyncMethodTest: myThreadPoolExecutor-10
inside asyncMethodTest: http-nio-8080-exec-8
```

Using Java ThreadPoolExecutor

Hey, I don't want all this confusion, Unwanted, Unwanted or Unwanted.

I always want to set my executor as default one, even if anyone use @Async, my executor only should be picked.

```
@Configuration
@EnableAsyncs
public class AsyncConfig {

    @Bean("myThreadPoolExecutor")
    public ThreadPoolExecutor myThreadPoolExecutor() {
        int minPoolSize = 2;
        int maxPoolSize = 4;
        int queueSize = 3;

        ThreadPoolExecutor poolTaskExecutor = new ThreadPoolExecutor(minPoolSize, maxPoolSize, 1minutes,
                TimeUnit.HOURS, new ArrayBlockingQueue<>(queueSize));
        return poolTaskExecutor;
    }
}
```

```
@RestController
public class UserService {

    @Async
    public void asyncMethodTest() {
        System.out.println("inside asyncMethodTest: " + Thread.currentThread().getName());
    }
}
```

Output:

```
2024-08-12T15:12:52.024+05:30  INFO 18012 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
inside asyncMethodTest: myThreadPoolExecutor-1
inside asyncMethodTest: myThreadPoolExecutor-2
inside asyncMethodTest: myThreadPoolExecutor-3
inside asyncMethodTest: http-nio-8080-exec-1
inside asyncMethodTest: http-nio-8080-exec-2
inside asyncMethodTest: myThreadPoolExecutor-4
inside asyncMethodTest: http-nio-8080-exec-3
inside asyncMethodTest: myThreadPoolExecutor-5
inside asyncMethodTest: myThreadPoolExecutor-6
inside asyncMethodTest: http-nio-8080-exec-4
inside asyncMethodTest: myThreadPoolExecutor-7
inside asyncMethodTest: http-nio-8080-exec-5
inside asyncMethodTest: myThreadPoolExecutor-8
inside asyncMethodTest: http-nio-8080-exec-6
inside asyncMethodTest: myThreadPoolExecutor-9
inside asyncMethodTest: http-nio-8080-exec-7
inside asyncMethodTest: myThreadPoolExecutor-10
inside asyncMethodTest: http-nio-8080-exec-8
```

Yes, No Executor name provided

Still, default executor configuration picked is mine one, not SimpleAsyncTaskExecutor

Output:

```
2024-08-12T15:12:52.024+05:30  INFO 18012 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
inside asyncMethodTest: myThreadPoolExecutor-1
inside asyncMethodTest: myThreadPoolExecutor-2
inside asyncMethodTest: myThreadPoolExecutor-3
inside asyncMethodTest: http-nio-8080-exec-1
inside asyncMethodTest: http-nio-8080-exec-2
inside asyncMethodTest: myThreadPoolExecutor-4
inside asyncMethodTest: http-nio-8080-exec-3
inside asyncMethodTest: myThreadPoolExecutor-5
inside asyncMethodTest: myThreadPoolExecutor-6
inside asyncMethodTest: http-nio-8080-exec-4
inside asyncMethodTest: myThreadPoolExecutor-7
inside asyncMethodTest: http-nio-8080-exec-5
inside asyncMethodTest: myThreadPoolExecutor-8
inside asyncMethodTest: http-nio-8080-exec-6
inside asyncMethodTest: myThreadPoolExecutor-9
inside asyncMethodTest: http-nio-8080-exec-7
inside asyncMethodTest: myThreadPoolExecutor-10
inside asyncMethodTest: http-nio-8080-exec-8
```