CHAPTER 4

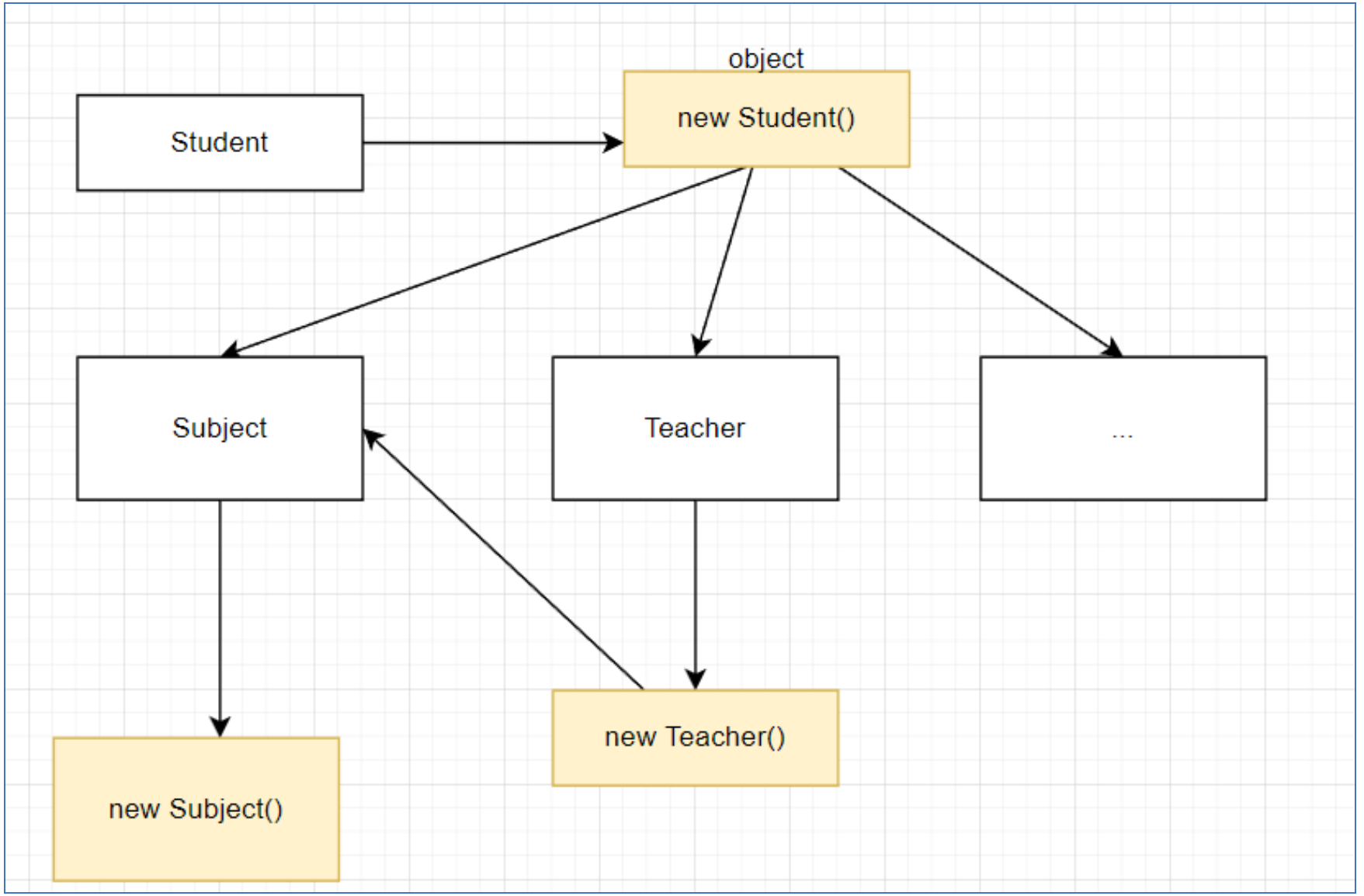
SPRING BOOT - API

Chapter 4: SPRING BOOT - API

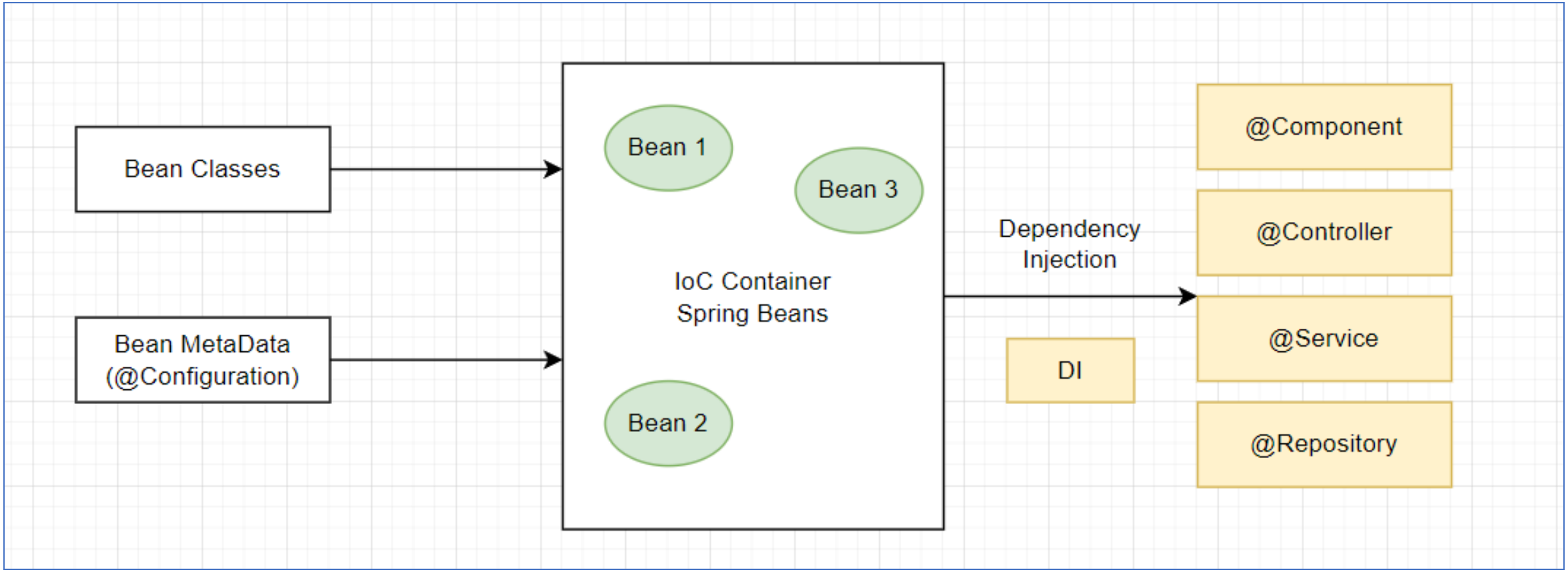
OUTLINE

1. Spring Beans (Build beans vs User-defined beans)
2. Dependency Injection
3. Application Context
4. IoC Container (Inversion of Control)
5. Validation
6. Logging
7. Profiles

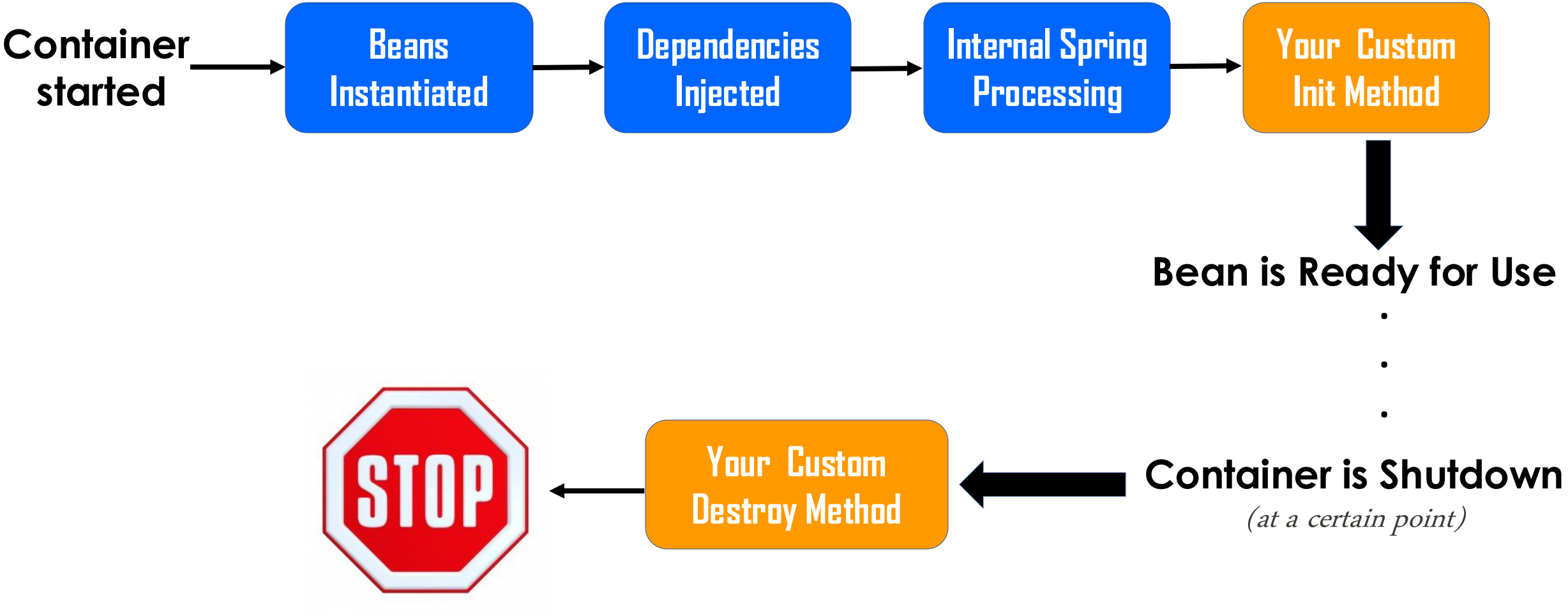
What is Beans?



What is Beans? (cont.)



Spring Bean Life Cycle



User Defined Spring Beans

1. @Controller, @Component, @Service, @Repository
2. @Bean, @Configuration

   @Scope: singleton, prototype, request, session, global-session

Maven

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>6.2.0</version>
</dependency>
```

Gradle

```
implementation 'org.springframework:spring-context:6.2.0'
```

User Defined Spring Beans

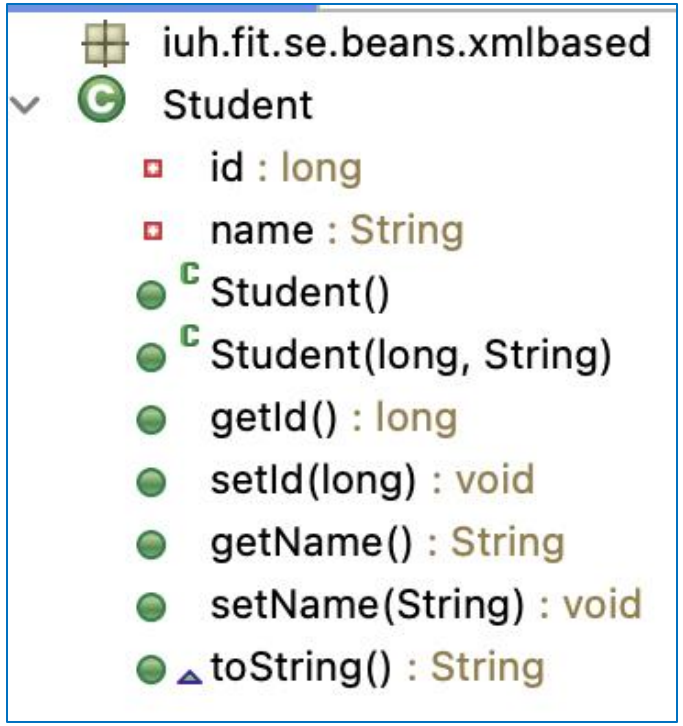
- The primary job of the ***ApplicationContext*** is to manage beans → application must provide the bean configuration to the ***ApplicationContext*** container.
- Type of configurations:
  - XML-Based Configuration: ***ClassPathXmlApplicationContext*** (xml file)
  - Annotation-Based Configuration: ***AnnotationConfigApplicationContext***
  - Java-Based Configuration: ***GenericWebApplicationContext***

Configuring Beans in the Container

XML-Based Configuration – Setter Injection

src/main/resources/beans.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:util="http://www.springframework.org/schema/util" xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/util
        http://www.springframework.org/schema/util/spring-util.xsd">
        <!-- bean definitions here -->
        <bean id="student1" class="iuh.fit.se.beans.xmlbased.Student">
            <property name="id" value="001"></property>
            <property name="name" value="Nguyen Van A"></property>
        </bean>
</beans>
```



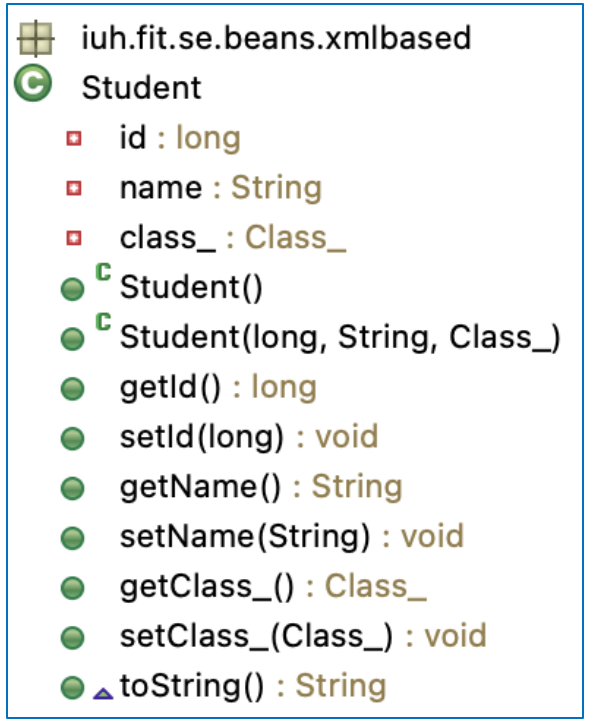

```java
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import iuh.fit.se.beans.xmlbased.Student;

public class Main {
    private static ApplicationContext context;
    public static void main(String[] args) {
        context = new ClassPathXmlApplicationContext("beans.xml");
        Student student1 = context.getBean("student1", Student.class);
        System.out.println(student1);
    }
}
```

Configuring Beans in the Container

XML-Based Configuration – Setter Injection – Object Injection

src/main/resources/beans.xml

```
<bean id="class1" class="iuh.fit.se.beans.xmlbased.Class_">
    <property name="classId" value="DHKTPM17C"></property>
    <property name="className" value="Dai Hoc KTPM17C"></property>
</bean>


<bean id="student2" class="iuh.fit.se.beans.xmlbased.Student">
    <property name="id" value="002"></property>
    <property name="name" value="Nguyen Van"></property>
    <property name="class_" ref="class1"></property>
</bean>
```




```
private static ApplicationContext context;
public static void main(String[] args) {
    context = new ClassPathXmlApplicationContext("beans.xml");
    Student student2 = context.getBean("student2", Student.class);
    System.out.println(student2);
}
```

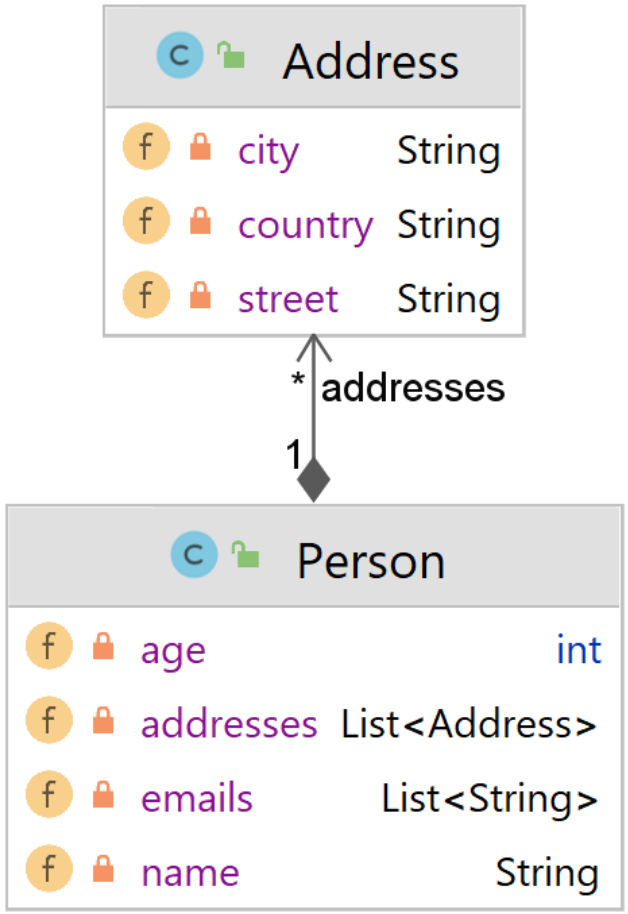
Configuring Beans in the Container

XML-Based Configuration – Setter Injection – Object Injection

```
<!--  Inject by setter -->
<bean id="class1" class="iuh.fit.se.beans.xmlbased.Class_">
    <property name="classId" value="DHKTPM17C"></property>
    <property name="className" value="Dai Hoc KTPM17C"></property>
</bean>

<bean id="student2" class="iuh.fit.se.beans.xmlbased.Student">
    <property name="id" value="002"></property>
    <property name="name" value="Nguyen Van"></property>
    <property name="class_" ref="class1"></property>
</bean>

<!--  Inject by constructor -->
<bean id="student3" class="iuh.fit.se.beans.xmlbased.Student">
    <constructor-arg name="id" value="003"></constructor-arg>
    <constructor-arg name="name" value="Nguyen Van Anh"></constructor-arg>
    <constructor-arg name="class_" ref="class1"></constructor-arg>
</bean>
```

Configuring Beans in the Container

XML-Based Configuration – Collection Injection

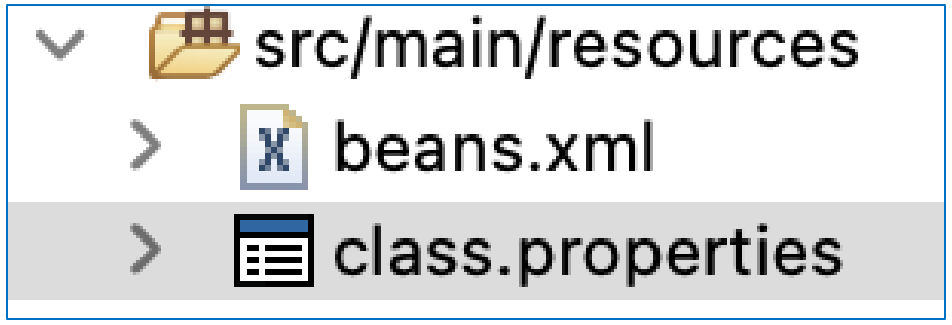


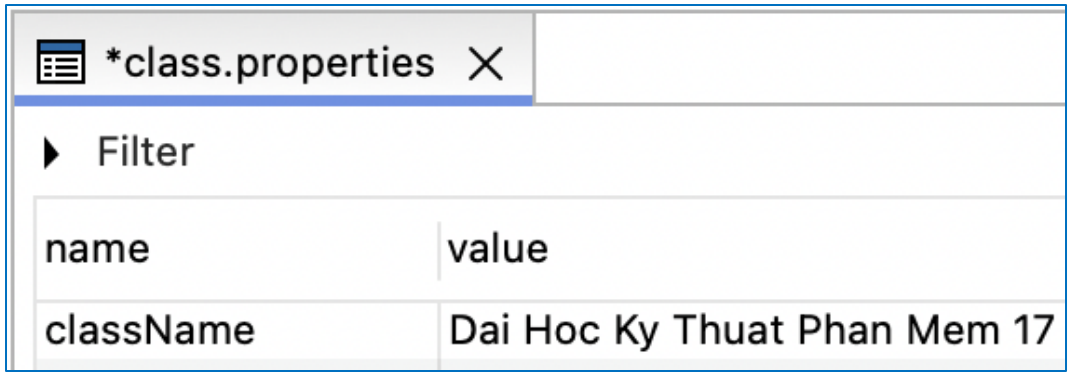

```xml
<bean id="address1" class="iuh.fit.se.beans.xmlbased.Address">
    <property name="street" value="Nguyen Van Bao"></property>
    <property name="city" value="HCM"></property>
    <property name="country" value="Viet Nam"></property>
</bean>

<bean id="address2" class="iuh.fit.se.beans.xmlbased.Address">
    <property name="street" value="Nguyen Thai Son"></property>
    <property name="city" value="HCM"></property>
    <property name="country" value="Viet Nam"></property>
</bean>

<bean id="person" class="iuh.fit.se.beans.xmlbased.Person">
    <property name="name" value="Nguyen Van A1"></property>
    <property name="addresses">
        <list>
            <ref bean="address1"/>
            <ref bean="address1"/>
        </list>
    </property>
    <property name="emails">
        <list>
            <value>a1@gmail.com</value>
            <value>a1@yopmail.com</value>
        </list>
    </property>
</bean>
```

Configuring Beans in the Container

XML-Based Configuration – Literal Values Injection

```
<bean id="classProperties"
    class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="locations">
        <list>
            <value>classpath:class.properties</value>
        </list>
    </property>
</bean>
<bean id="class2" class="iuh.fit.se.beans.xmlbased.Class_">
    <property name="classId" value="DHKTPM"></property>
    <property name="className" value="${className}"></property>
</bean>
```





Spring's Auto-wiring (cont.)

```
<bean id="faculty" class="iuh.fit.se.beans.autowiring.Faculty">
    <property name="id" value="FI"></property>
    <property name="name" value="Faculty of Information Technology"></property>
</bean>
<bean id="dept1" class="iuh.fit.se.beans.autowiring.Department" autowire="no">
    <property name="id" value="DP1"></property>
    <property name="name" value="Department 1"></property>
    <property name="faculty" ref="faculty"></property>
</bean>
```

explicit

Can be omitted

```
public class Department {
    private String id;
    private String name;
    private Faculty faculty;
```

Find bean with the id = "faculty"

```
<bean id="faculty" class="iuh.fit.se.beans.autowiring.Faculty">
    <property name="id" value="FI"></property>
    <property name="name" value="Faculty of Information Technology"></property>
</bean>

<bean id="dept2" class="iuh.fit.se.beans.autowiring.Department" autowire="byName">
    <property name="id" value="DP2"></property>
    <property name="name" value="Department 2"></property>
</bean>
```

Spring's Auto-wiring (cont.)

```
<bean id="faculty" class="iuh.fit.se.beans.autowiring.Faculty">
    <property name="id" value="FI"></property>
    <property name="name" value="Faculty of Information Technology"></property>
</bean>

<bean id="dept3" class="iuh.fit.se.beans.autowiring.Department" autowire="byType">
    <property name="id" value="DP3"></property>
    <property name="name" value="Department 3"></property>
</bean>
```

Find bean with the type "Faculty"

```
public class Department {
    private String id;
    private String name;
    private Faculty faculty;
```

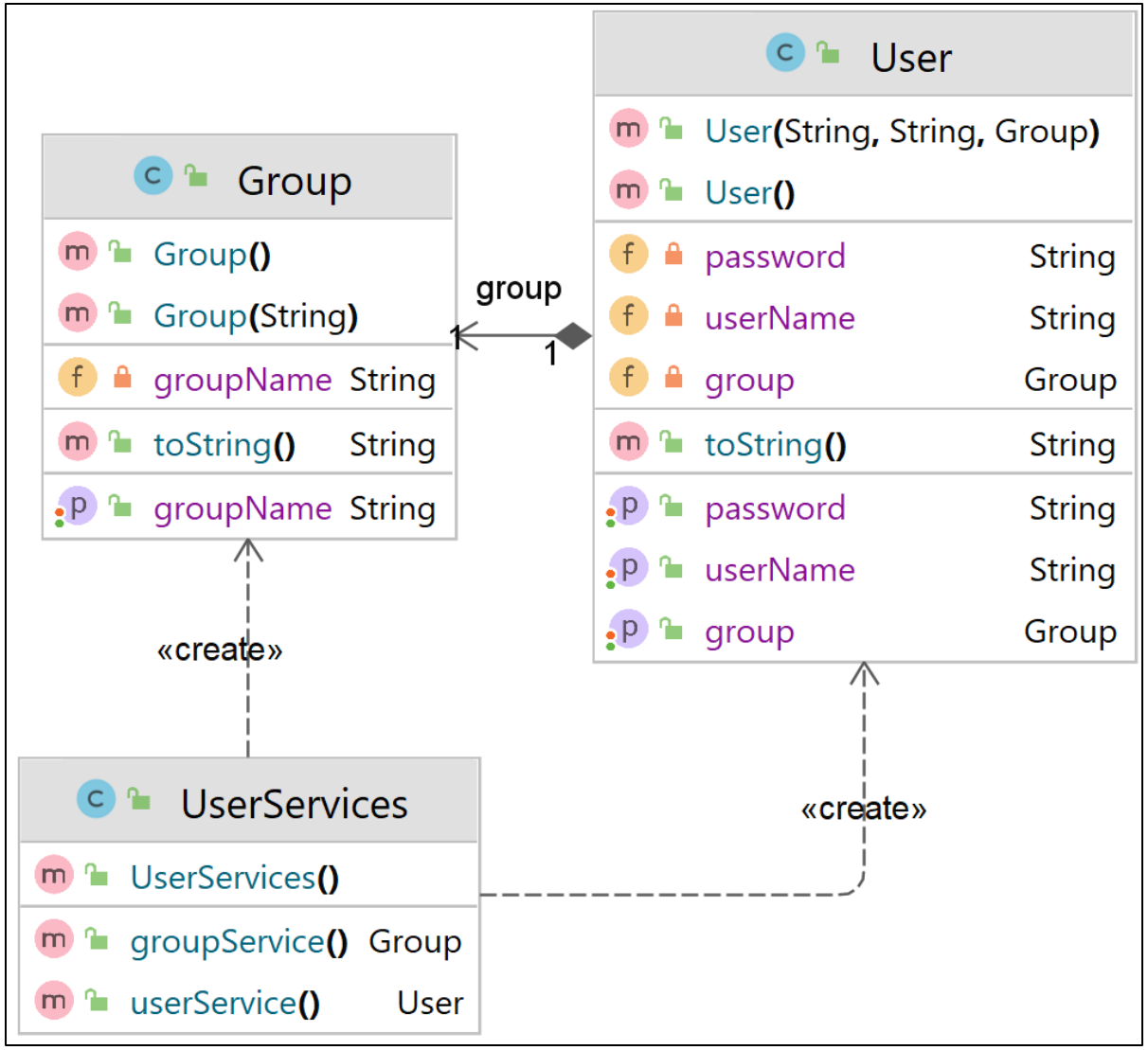
Spring's Auto-wiring (cont.)

```xml
<bean id="faculty" class="iuh.fit.se.beans.autowiring.Faculty">
    <property name="id" value="FI"></property>
    <property name="name" value="Faculty of Information Technology"></property>
</bean>

<bean id="dept4" class="iuh.fit.se.beans.autowiring.Department" autowire="constructor">
    <property name="id" value="DP4"></property>
    <property name="name" value="Department 4"></property>
</bean>
```

Find bean with the type "Faculty"
but it looks for the class type of the constructor arguments

```java
public class Department {
    private String id;
    private String name;
    private Faculty faculty;

    public Department() {
    }
    public Department(Faculty faculty) {
        this.faculty = faculty;
    }
```

Configuring Beans in the Container
Java-Based Configuration(cont.)



Configuring Beans in the Container
Java-Based Configuration(cont.)

```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class UserService {

    @Bean
    public Group groupService() {
        return new Group(1, "Admin Group");
    }

    @Bean
    public User userSevice() {
        return new User(1, "User 01", "123456", groupService());
    }
}
```

Configuring Beans in the Container
Java-Based Configuration(cont.)

```
public static void main(String[] args) {
    ApplicationContext context = new
            AnnotationConfigApplicationContext(UserService.class);
    User user = context.getBean(User.class);
    System.out.println(user);

    Group group = context.getBean(Group.class);
    System.out.println(group);
}
```

Configuring Beans in the Container
Annotation-Based Configuration

Create the XML configuration, ***applicationContext.xml (src/main/resources)*** to enable annotations:

*applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd">

  <context:annotation-config/>
  <context:component-scan base-package="fit.se.annotationbased"/>

</beans>
```

You can also use ***@Configuration*** annotation with the same purpose

Configuring Beans in the Container
Annotation-Based Configuration (cont.)

<context:annotation-config/> only looks for annotations on beans in the same application context in which it is defined.

You can also use ***@Configuration*** annotation with the same purpose

```
@Configuration
@ComponentScan(“fit.se.annotationbased")
public class AppConfig {

}
```

@Autowired xml config

```
<bean id="user" class="iuh.fit.se.annotationbased.User">
  <property name="userName" value="user01"></property>
  <property name="password" value="123456"></property>
</bean>
<bean id="group" class="iuh.fit.se.annotationbased.Group">
  <property name="name" value="Group 1"></property>
</bean>
```

iuh.fit.se.annotationbased

User
- id : int
- userName : String
- password : String
- group : Group
- User()
- User(Group)
- User(int, String, String, Group)
- getId() : int
- setId(int) : void
- getUserName() : String
- setUserName(String) : void
- getPassword() : String
- setPassword(String) : void
- getGroup() : Group
- setGroup(Group) : void
- toString() : String

iuh.fit.se.annotationbased

Group
- id : int
- name : String
- Group()
- Group(int, String)
- getId() : int
- setId(int) : void
- getName() : String
- setName(String) : void
- toString() : String

Field

```
@Autowired
private Group group;
```

Constructor

```
@Autowired
public User(Group group) {
    super();
    this.group = group;
}
```

Setter

```
@Autowired
public void setGroup(Group group) {
    this.group = group;
}
```

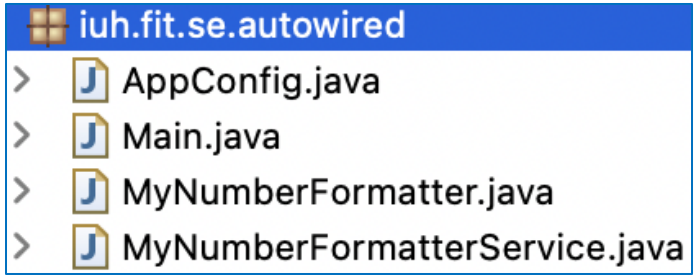
```
public static void main(String[] args) {
    ApplicationContext context = new
            ClassPathXmlApplicationContext("applicationContext.xml");
    User user = context.getBean("user", User.class);
    System.out.println(user);
}
```

@Autowired Java-based config

```java
@Component
public class MyNumberFormatter {
    public String format(double number) {
        return "Number format: " + number;
    }
}
```

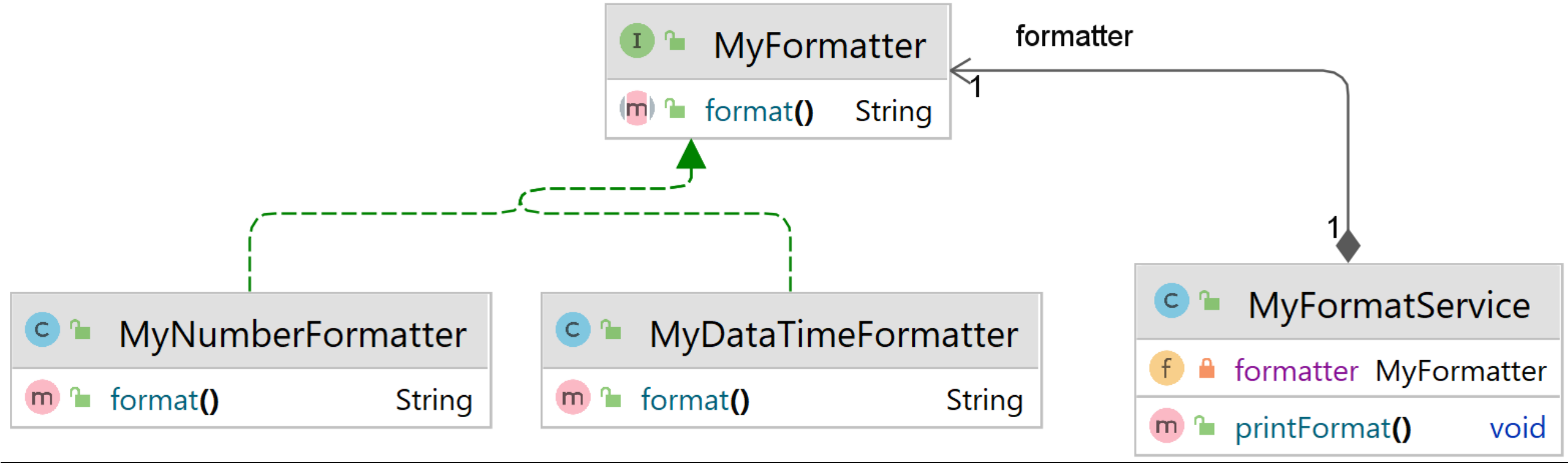
```java
@Configuration
@ComponentScan("iuh.fit.se.autowired")
public class AppConfig {
    // No operation
}
```

```java
@Component
public class MyNumberFormatterService {
    private final MyNumberFormatter myNumberFormatter;

    // @Autowired - not required
    public MyNumberFormatterService(MyNumberFormatter myNumberFormatter) {
        this.myNumberFormatter = myNumberFormatter;
    }

    public void printFormat(double number) {
        System.out.println(myNumberFormatter.format(number));
    }
}
```



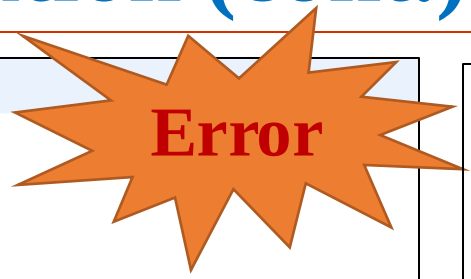

```java
public static void main(String[] args) {
    ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);
    MyNumberFormatterService service = context.getBean(MyNumberFormatterService.class);
    service.printFormat(1000d);
}
```

@Autowired Disambiguation



@Autowired Disambiguation (cont.)

```java
@Component
public class MyFormatService {
    private MyFormatter formatter;

    public MyFormatService(MyFormatter formatter) {
        this.formatter = formatter;
    }

    public void printFormat() {
        System.out.println(formatter.format());
    }
}
```



Caused by: org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean of type 'iuh.fit.se.autowired.disambiguation.

```java
public static void main(String[] args) {
    ApplicationContext context = new
            AnnotationConfigApplicationContext(AppConfig.class);
    MyFormatService service = context.getBean(MyFormatService.class);
    service.printFormat();
}
```

Disambiguation solution: @Qualifier

```java
@Component("myDateTimeFormatter") // bean name
public class MyDateTimeFormatter implements My

    @Override
    public String format() {
        return "MyDateTimeFormatter";
    }

}
```

```java
@Component("myNumberFormatter")
public class MyNumberFormatter implements MyFormatter {

    @Override
    public String format() {
        return "MyNumberFormatter";
    }

}
```

```java
public MyFormatService(@Qualifier("myNumberFormatter") MyFormatter formatter) {
    this.formatter = formatter;
}
```

When there are multiple beans of the same type → use **@Qualifier** to avoid ambiguity.

Spring uses the bean's name as a default qualifier value.

Disambiguation solution: @Primary

```java
@Component
public class MyDateTimeFormatter implements MyFormatter {

    @Override
    public String format() {
        return "MyDateTimeFormatter";
    }
}
```

```java
@Component
@Primary
public class MyNumberFormatter implements MyFor

    @Override
    public String format() {
        return "MyNumberFormatter";
    }

}
```

```java
@Component
public class MyFormatService {
    private MyFormatter formatter;

    public MyFormatService(MyFormatter formatter) {
        this.formatter = formatter;
    }

    public void printFormat() {
        System.out.println(formatter.format());
    }
}
```

@Primary indicates that a particular bean should be given preference when multiple beans are candidates to be autowired to a single-valued dependency.

Inject resources with @Value

```
@Configuration
@ComponentScan(”iuh.fit.se.resources")
@PropertySource("classpath:application.properties
")
public class AppConfig {
    @Bean
    public ClientBean clientBean() {
        return new ClientBean();
    }
}
```

```
public class Main {
    public static void main(String[] args) throws IOException {
        AnnotationConfigApplicationContext context =
                new
AnnotationConfigApplicationContext(AppConfig.class);
        ClientBean bean = context.getBean(ClientBean.class);
        bean.doSomething();
    }
}
```

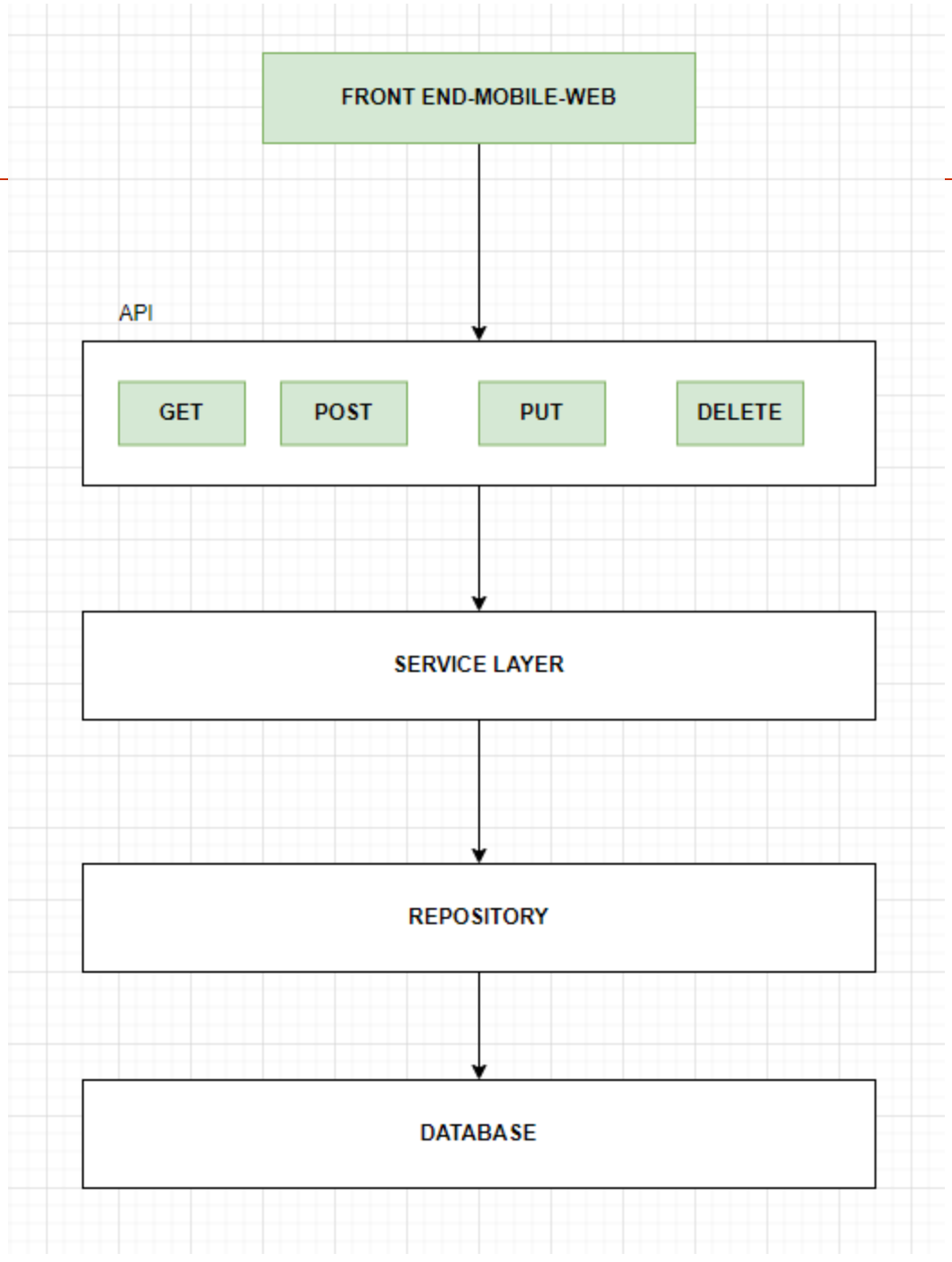
```
public class ClientBean {
    @Value("classpath:beans.xml")
    private Resource myResource;

    @Value("${foo.permission}")
    private String permission;

    public void doSomething() throws IOException {
        File file = myResource.getFile();
        String s = new
String(Files.readAllBytes(file.toPath()));
        System.out.println(s);
        System.out.println(permission);
    }
}
```

application.properties

```
foo.permission=this is sample permission
```

Spring REST API

Ref: 4_Spring_REST_API_Tutorial


FRONT END-MOBILE-WEB
API
GET
POST
PUT
DELETE
SERVICE LAYER
REPOSITORY
DATABASE

Validation

https://jakarta.ee/learn/docs/jakartaee-tutorial/current/beanvalidation/bean-validation/bean-validation.html

`@NotBlank`	The value of the field or property must contain atleast one non-white space character.	`@NotBlank` `String message;`
`@NotEmpty`	The value of the field or property must not be empty. The length of the characters or array, and the size of a collection or map are evaluated.	`@NotEmpty` `String message;;`

`@NotNull`	The value of the field or property must not be null.	JAVA |
		`@NotNull`
		`String username;`

`@Null`	The value of the field or property must be null.	`@Null` `String unusedString;`

Validation (cont.)

`@Email`	The value of the field or property must be a valid email address.	`@Email` `String emailaddress;`
`@Future`	The value of the field or property must be a date in the future.	`@Future` `Date eventDate;`
`@FutureOrPresent`	TThe value of the field or property must be a date or time in present or future.	`@FutureOrPresent` `Time travelTime;`

Validation (cont.)

`@Past`	The value of the field or property must be a date in the past.	`@Past` `Date birthday;`
`@PastOrPresent`	The value of the field or property must be a date or time in the past or present.	`@PastOrPresent` `Date travelDate;`
`@Pattern`	The value of the field or property must match the regular expression defined in the `regexp` element.	`@Pattern(regexp="\\(\\d{3}\\)\\d{3}-\\d{4}")` `String phoneNumber;`

Validation (cont.)

`@Max`	The value of the field or property must be an integer value lower than or equal to the number in the value element.	`@Max(10)` `int quantity;`
`@Min`	The value of the field or property must be an integer value greater than or equal to the number in the value element.	`@Min(5)` `int quantity;`
`@Negative`	The value of the field or property must be a negative number.	`@Negative` `int basementFloor;`
`@NegativeOrZero`	The value of the field or property must be negative or zero.	`@NegativeOrZero` `int debtValue;`

Data Transfer Object - DTO

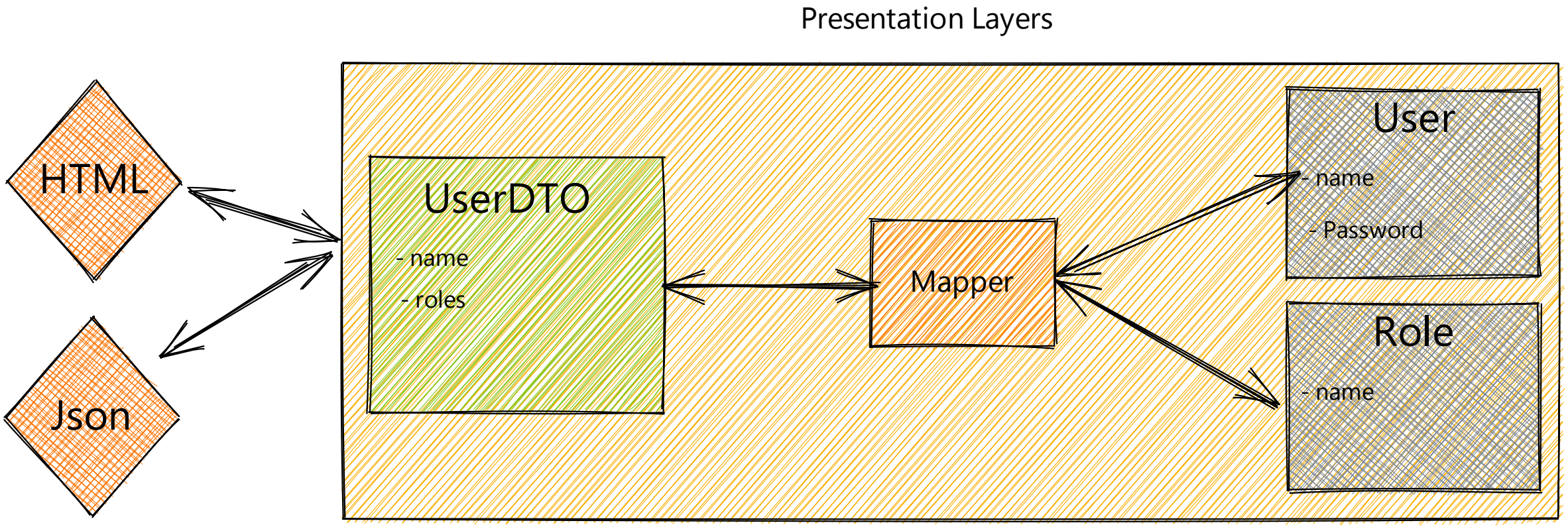
What is Data Transfer Object?

- A data transfer object (DTO) is an object that carries data between processes.
- It is particularly useful when you want to transfer data across different layers of an application or over the network.
- DTOs normally are created as POJOs. They **are flat data structures that contain no business logic.** They only contain storage, accessors and eventually methods related to serialization or parsing.

Data Transfer Object - Advantages

- Advantages
  - Decouple business logic from the communication layer.
  - Hide unnecessary data or protect sensitive information.
  - Avoid multiple calls to the remote server.
  - Prevent breaking changes in the API during updates of the application model.
- Dis-advantages:
  - Overhead: Can introduce additional layers of complexity.
  - Requires mapping: Often requires conversion from and to POJOs or entities.

What is Data Transfer Object?



Data Transfer Object - Example

```java
@Entity
@Table(name = "cr_user")
public class User {
    @Id
    @Column(name = "id", nullable = false)
    private Long id;

    @Column(name = "email")
    @NotNull
    @Email
    private String email;

    @Column
    @NotNull
    private String fullName;
```

```java
public class UserDTO {

    private String email;

    private String fullName;

    private String phone;

    private String avatar;

    public UserDTO(User original) {
        this.email = original.getEmail();
        this.fullName = original.getFullName();
        this.phone = original.getPhone();
        this.avatar = original.getAvatar();
    }
}
```

Data Transfer Object – Example (cont.)

```java
@GetMapping("/users")
public List<UserDTO> getUserList(){
    List<User> users = userService.getListUser();

    List<UserDTO> dtoList = new ArrayList<>();
    for (User user : users){
        dtoList.add(new UserDTO(user));
    }
    return dtoList;
}
```

Logging

Ref: https://docs.spring.io/spring-boot/reference/features/logging.html

Logging

```java
@Autowired
private EmployeeService employeeService;

private final Logger LOGGER =
        LoggerFactory.getLogger(EmployeeController.class);


@PostMapping("/employees")
public Employee saveEmployee(@Valid @RequestBody Employee employee){

    LOGGER.info("Inside saveEmployee of EmployeeController!");

    return employeeService.saveEmployee(employee);
}
```

Logging (cont.)

```
2021-09-14 19:32:26.769  INFO 19076 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
2021-09-14 19:32:26.818  INFO 19076 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2021-09-14 19:32:26.831  INFO 19076 --- [  restartedMain] c.e.s.SpringBootApiApplication           : Started SpringBootApiApplication in 6.064 seconds (JVM running for 8.51)
2021-09-14 19:32:59.864  INFO 19076 --- [nio-8080-exec-2] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2021-09-14 19:32:59.864  INFO 19076 --- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2021-09-14 19:32:59.865  INFO 19076 --- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
2021-09-14 19:33:00.060  INFO 19076 --- [nio-8080-exec-2] c.e.s.controller.EmployeeController      : Inside saveEmployee of EmployeeController!
```

Profiles

Ref: https://docs.spring.io/spring-boot/reference/features/profiles.html

Profiles

- Spring Profiles provide a way to segregate parts of your application configuration and make it be available only in certain environments.

```yaml
spring:
  config:
    activate:
      on-profile: qa
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    password: password
    url: jdbc:mysql://localhost:3306/ktpm_21_22_db-qa
    username: root
  jpa:
    hibernate:
      ddl-auto: update

welcome:
  message: 'welcome to qa'
```

```yaml
spring:
  config:
    activate:
      on-profile: production
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    password: password
    url: jdbc:mysql://localhost:3306/ktpm_21_22_db-production
    username: root
  jpa:
    hibernate:
      ddl-auto: update

welcome:
  message: 'welcome to production'
```

Profiles

```
   (v2.5.4)

20216 --- [  restartedMain] c.e.s.SpringBootApiApplication          : Starting SpringBootApiApplication using Java 16.0.2 on DESKTOP-I628VG6 with PID 20216 (D:\CODE\
20216 --- [  restartedMain] c.e.s.SpringBootApiApplication          : The following profiles are active: production
20216 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-properties' to 'false' to disable
20216 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level.web' property to 'DEBUG'
20216 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
20216 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 62 ms. Found 1 JPA repository interfaces.
20216 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
```

Swagger springdoc-openapi

Ref: https://springdoc.org

Swagger – springdoc-openapi

```
<!--https://mvnrepository.com/artifact/org.springdoc/
springdoc-openapi-starter-webmvc-ui -->
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.6.0</version>
</dependency>
```

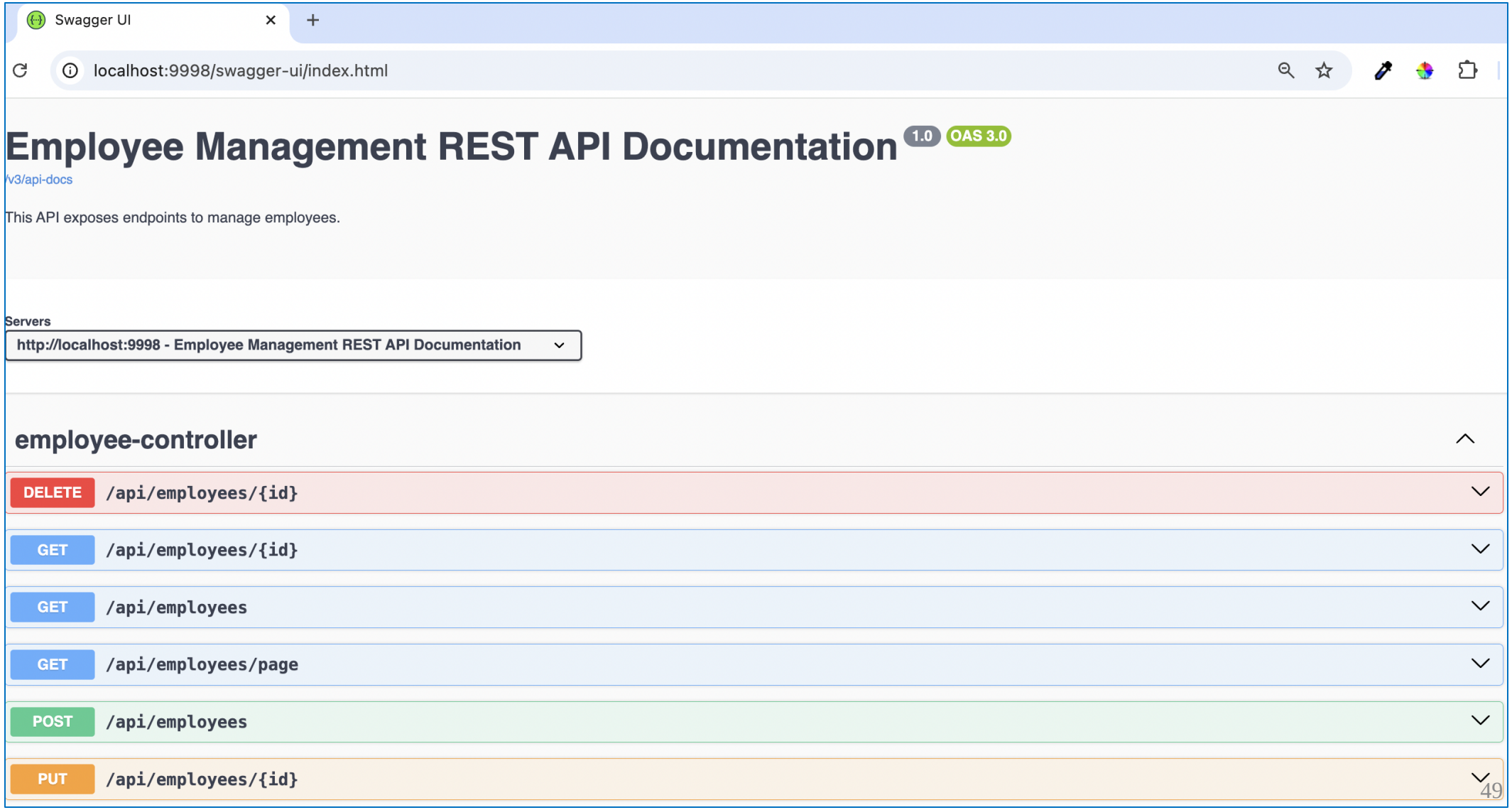
application.properties

```
# Paths to include
springdoc.pathsToMatch=/**
springdoc.paths-to-exclude=/api/profile/**
springdoc.swagger-ui.operationsSorter=method
```

Swagger – springdoc-openapi (cont.)

```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import io.swagger.v3.oas.models.OpenAPI;
import io.swagger.v3.oas.models.info.Info;
import io.swagger.v3.oas.models.servers.Server;

@Configuration
public class OpenAPIConfiguration {
    @Bean
    OpenAPI defineOpenApi() {
        Server server = new Server();
        server.setUrl("http://localhost:9998");
        server.setDescription("Employee Management REST API Documentation");

        Info information = new Info()
                .title("Employee Management REST API Documentation")
                .version("1.0")
                .description("This API exposes endpoints to manage employees.");

        return new OpenAPI().info(information).servers(List.of(server));
    }
}
```

Swagger – springdoc-openapi (cont.)



Q&A