

Workshop: Spring Intro & Boot & Data

© Michael Inden, 2021 – 2022

PART 1: Spring Framework Basics

Exercise 1: Import Project Ex01_TemplateApp

10 min

Import the project `ex01-project-template`, start the application class and clarify the following simple questions:

1. this project has a transitive dependency on `spring-context`. True / False
2. `spring-context` contains the implementations of the IoC container. True / False

[Hint: Look in the `pom.xml` and the Maven dependencies.]

Exercise 2: Greeting App – Warm Up

25 min

Import the project `ex02-project-template`. After that, the following tasks have to be solved:

- Create a simple **GreetingService** class with a method **sayHello()** to print the following sentence "Welcome to Spring Workshop :-) "
- Create a simple **GreetingApp** class with the `main()` method and use the `GreetingService` and call the `sayHello()` method. Run the app and verify the output.
- Now, the instantiation of the `GreetingService` directly with "new" keyword is forbidden. How would you instantiate them? Run the app and check that it prints out the same output.

[Hint: Use `ClassPathXmlApplicationContext`, adjust pom if necessary]

[Hint: Create spring's IoC configuration metadata for this `GreetingService` class. Use the below configuration as the template]

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

<!--TODO -->

</beans>
```

- The above app simply prints the hardcoded "Welcome to Spring Workshop :-) ", this is a good start. However, it would be better if it's configurable.

[Hint: use constructor injection.]

Exercise 3: Asmiq Academy App – initial setup

30 min

We have received a requirement for the Asmiq Academy to create an application that enables to place an order for the courses for a customer. In Asmiq Academy we have roughly designed the classes and it is shown below Figure 1. The AsmiqAcademyApp should do the following:

- Use the top-level AsmiqAcademyService to retrieve the list of courses
- Filter for a particular course and place an order for a customer
- Sms must be sent after placement of an order

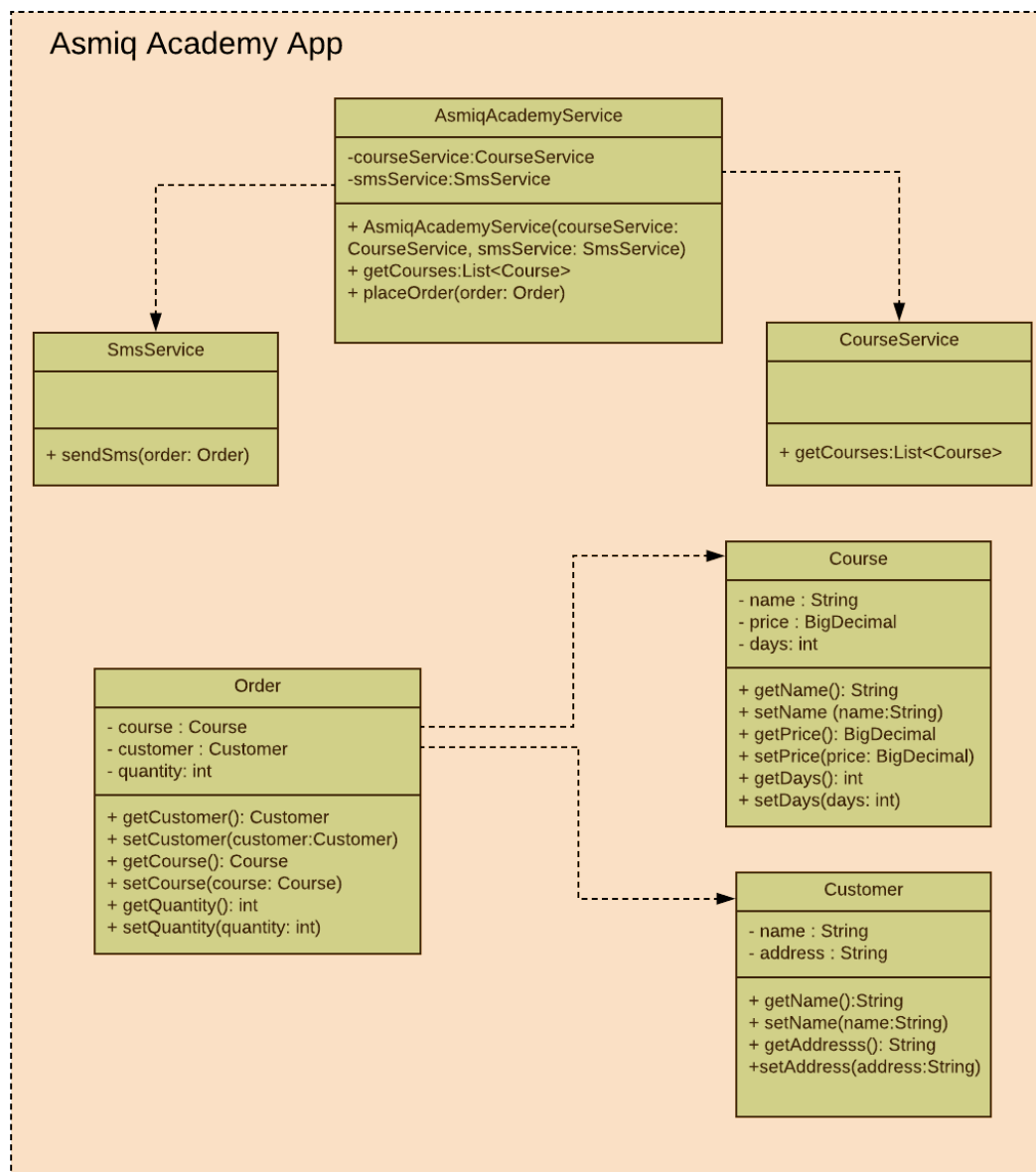


FIGURE 1

Implement the AsmiqAcademyApp by using the Spring IoC for the services through using project template "ex03-asmiq-academy-app-template":

- XML configuration with autowiring
- Java configuration with autowiring

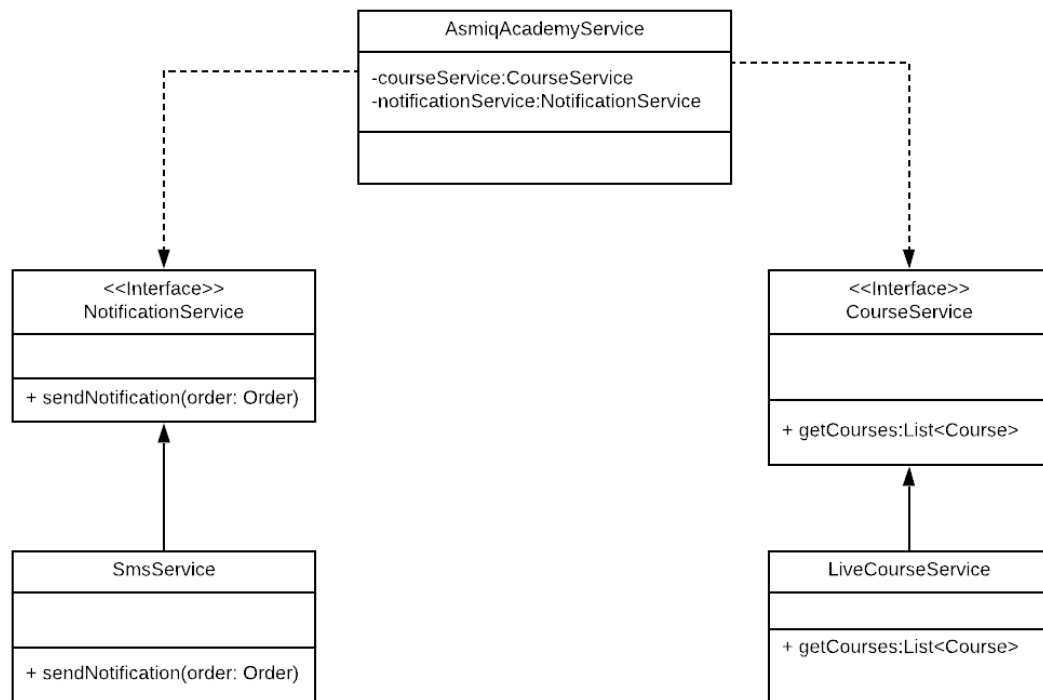
Tip: Specifying the dependencies

```

c:smsService-ref="smsService">
<constructor-arg name="courseService" ref="courseService" />
  
```

Exercise 4: Asmiq Academy App – Interface Extraction**10 min**

Upon reviewing the class design, our team lead mentioned that the AsmiqAcademyService should depend on abstractions and not on concrete implementations. Hence, we came up with the following two new interfaces as shown in Figure 2.

Asmiq Academy App**FIGURE 2**

Exercise: As a result of the extraction of the two new interfaces, modify our AsmiqAcademyService App with these changes and do DI/IoC with Java configuration.

[Hint: use the project template "**ex04-asmiq-academy-app-template**"]

Exercise 5: Asmiq Academy App – Bean Resolution Strategies 30 min

As we have started receiving enquiries/registration from the customers outside Switzerland, we confirm their bookings through E-Mail. Besides some customers requested us to provide online courses. Hence, we have to extend the design (Figure 3) to accommodate these new requirements.

[Hint: use the project template "ex05-asmiq-academy-app-template"]

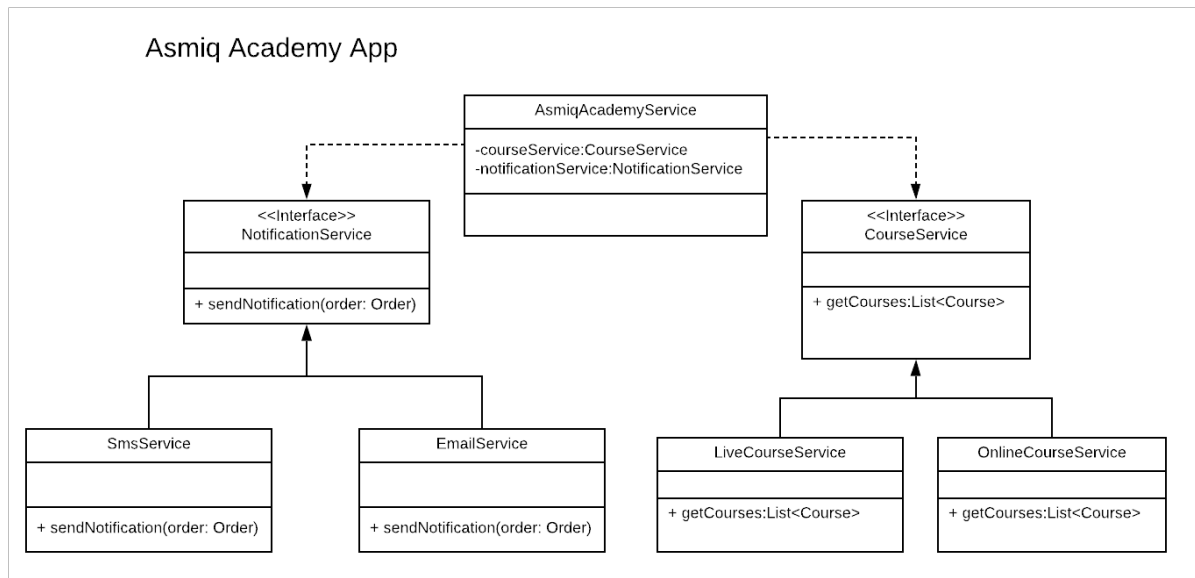


FIGURE 3

- a. Extend our AsmiqAcademy App with two new services by just duplicating the original implementation. Run the app and check for the result.

[Hint: `org.springframework.beans.factory.NoUniqueBeanDefinitionException`: No qualifying bean of type 'ch.asmiq.interfaces.CourseService' available: expected single matching bean but found 2: `liveCourseService`, `onlineCourseService`]

- b. Solve the issue with `NoUniqueBeanDefinitionException`

[Hint: try with `@Primary@Bean` (in Config), `@Primary` in Service, `@Qualifier`]

TRICK: Führe noch ein `AcademyService`-Interface ein, um die Varianten austauschbar zu machen.

Exercise 6: Asmiq Academy App – (No-)Mandatory Dependencies 20 min

We are planning to enhance our app to include services like payment, feedback, exam and certification service as shown below Figure 4. However, right now we have agreed with SixPayment for payment, SurveyMonkey for feedback and Prometric for examination and we are still in search for certification providers.

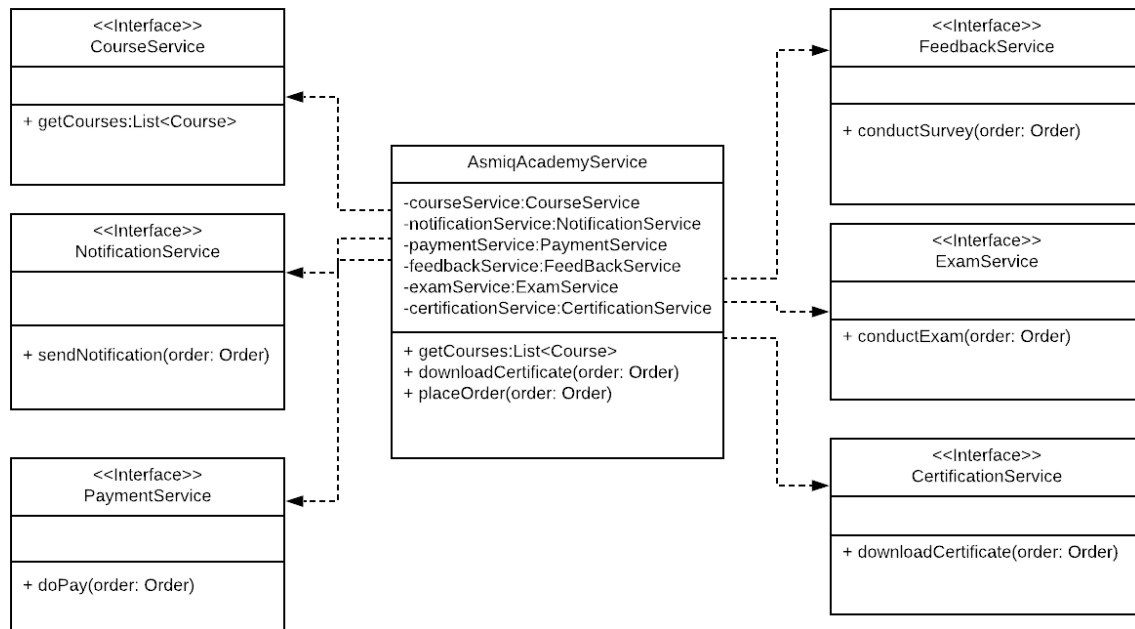


FIGURE 4

Enhance the AsmiqAcademyService as shown in the diagram by taking into consideration that when the AsmiqAcademyService is instantiated by the container all the services have to be injected **except the CertificationService (non-mandatory)**. Constructor injection is not applicable here, hence find a better solution.

[Hint: use the maven project "**ex06-asmiq-academy-app-template**" as the template]

BONUS: Discussion setter-Injection / Constructor-Injection / Field-Injection

Exercise 7: Asmiq Academy App – Configuration Externalization 30 min

The Implementation of the `SixPaymentService` is shown below. It shows that the value for `discountPercent` has been hardcoded in the source-code itself.

```
@Component
public class SixPaymentService implements PaymentService {

    private BigDecimal discountPercent = new BigDecimal("0.25");

    @Override
    public void doPay(Order order) {
        BigDecimal coursePrice = order.getCourse().getPrice();
        BigDecimal discountPrice = coursePrice.multiply(discountPercent);
        BigDecimal totalPrice = coursePrice.subtract(discountPrice);
        //log the order details with discounts
    }
}
```

Modify the `SixPaymentService` class to externalize the `discountPercent` through some property file (for example: `sixPaymentService.properties`) that is placed in the classpath.

[Hint: Use " `ex07-asmiq-academy-app-template`" as the template]

Exercise 8: Improval of application design – Prepare for Spring 30 min

Given are the classes of a pizza service, which are, however, closely connected to each other, since the constructors are called in each case.

```
public class PizzaService {
    private final Discount discount;
    private final CustomerDAO customerDAO;
    private final Map<Long, Receipt> customerToReceipt = new HashMap<>();

    public PizzaService() {
        // direct dependencies
        discount = new Discount();
        customerDAO = new CustomerDAO();
    }

    public void orderPizza(final long customerId, final Pizza pizza) {
        final Customer customer = customerDAO.findById(customerId);
        customerToReceipt.putIfAbsent(customerId,
                                     new Receipt(customer));
        final Receipt receipt = customerToReceipt.get(customerId);

        final double price = discount.apply(pizza);
        receipt.addEntry(pizza, price);
    }
}
```

When executing the order process, various customer data is needed. For this purpose, a database is accessed. Fortunately, a so-called Data Access Object (DAO) has already abstracted this. Likewise, now and then, there are discount promotions, which are modeled by the Discount class. Below, it is shown that `PizzaService` resolves (or rather, self-provides) both dependencies by instantiating the respective classes.

This design is disruptive if this application is to be tested or used in a Spring context. As a challenge, the design should now be adapted adequately.

[Hint: Use project **ex08-design-improvement** as starting point]

Exercise 9: Asmiq Academy App – Pre Init Consistency Checks 10 min

The `SixPaymentService` wants to make sure that the discount percent should not be more than 75%. If it is set to more than 75% the entire App should not start up. Modify the `SixPaymentService` to accommodate this requirement.

[Hint: Use "ex09-asmiq-academy-app-template" as the template]

[Hint: Use bean lifecycle callbacks]

[Hint: Upon demand add the following dependency]

```
<dependency>
  <groupId>javax.annotation</groupId>
  <artifactId>javax.annotation-api</artifactId>
  <version>1.3.2</version>
</dependency>
```

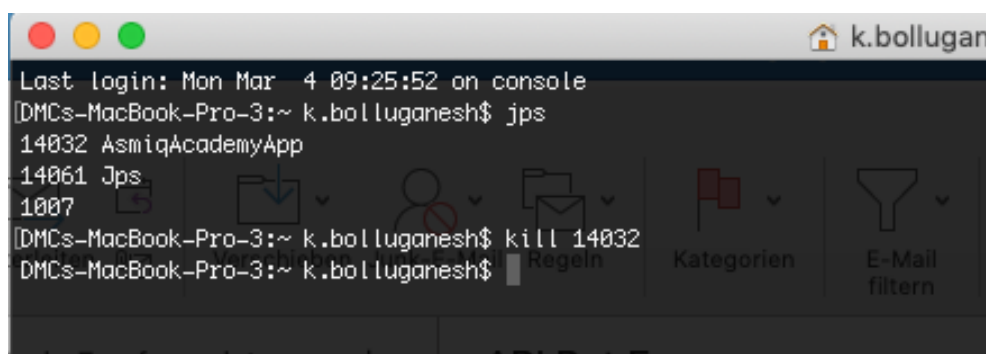
Exercise 10: Asmiq Academy App – Graceful Shut Down 15 min

It is our policy to gracefully shutdown our apps if the JVM process has been terminated intentionally / unintentionally.

- modify our `AsmiqAcademyApp` to satisfy this requirement.

[Hint: a. **registerShutdownHook()**; b. gracefully shutdown after 15 or 30 Seconds]

- Run the app and kill the app's process through the kill <process_id> as shown below and check the console log.



PART Spring MVC

Exercise 13: Explore Web Template Project – Warm Up 30 min

Import the maven project "**ex13-web-project-template**" into your IDE and work on the following tasks:

- a) Run the class `WebTemplateApp` and fix the below exception. Use port 8080. Start again.

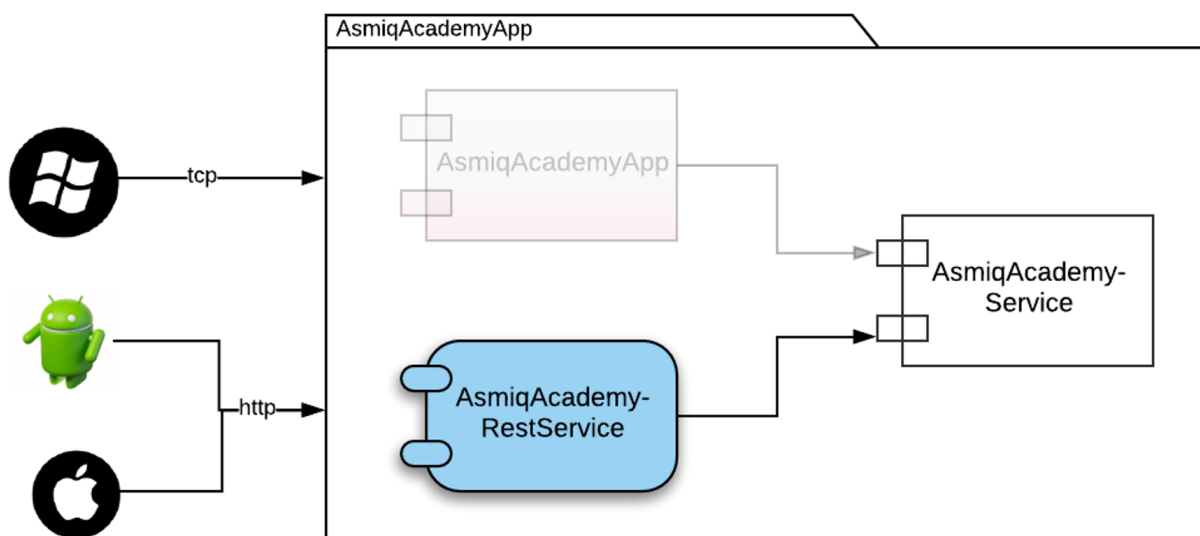
```
Caused by: java.lang.IllegalArgumentException: port out of range:-1
    at java.base/java.net.InetSocketAddress.checkPort(InetSocketAddress.java:143)
    at java.base/java.net.InetSocketAddress.<init>(InetSocketAddress.java:188)
    at org.apache.tomcat.util.net.NioEndpoint.initServerSocket(NioEndpoint.java:235)
    at org.apache.tomcat.util.net.NioEndpoint.bind(NioEndpoint.java:210)
```

- b) Change port so that tomcat listens on port 7777.
- c) Explore the project and identify crucial components.
- d) Modify and extend the `GreetingController` class to expose a REST endpoint `"/hello"` which returns the message "Welcome Spring Workshop Participants ☺". Run the app and call `http://localhost:7777/hello` and verify that the browser displays "Welcome Spring Workshop Participants ☺"
[Hint: use "😅" for smiley]
- e) Expose another REST endpoint `"/hello/<name>"` to return the string "Welcome" + `<name>`, `<name>` is any string, which would be entered by the user. Run the app and call `http://localhost:7777/hello/SPRING` and verify that the browser displays "Welcome SPRING"

BONUS: What are the advantages of using the embedded tomcat?

Exercise 14: REST Service for Asmiq Academy App 30 min

In the last part of the exercise, we implemented a nice standalone `AsmiqAcademyApp`. Now, we have a new requirement from the customers to expose a REST endpoint `"/courses"` to retrieve the list of courses, so that they can consume this endpoint and list the courses in their GUIs/Apps. Due to spring limitations the application code in `main()` got removed.



a. Import the maven project "**ex14-asmik-academy-rest-app-template**" to do the following:

- (1) Work on the TODOs in CourseController class
- (2) Run the App and execute <http://localhost:8080/courses> in the browser.
- (3) Analyze and fix the following problem:

HTTP Status 406 – Not Acceptable

Type Status Report

Description The target resource does not have a current representation that would be acceptable to the user agent, according to the proactive negotiation header fields received in the request, and the server is unwilling to supply a default representation.

Apache Tomcat/9.0.16

Sep. 26, 2021 1:51:48 NACHM.

org.springframework.web.servlet.handler.AbstractHandlerExceptionResolver
[logException](#)

WARNING: Resolved

[org.springframework.web.HttpMediaTypeNotAcceptableException: Could not
find acceptable representation]

[Hint: Check pom.xml and Jackson Dependency]

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-core</artifactId>
  <version>2.12.5</version>
</dependency>
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.12.5</version>
</dependency>
```

For XML you can include the following:

```
<dependency>
  <groupId>com.fasterxml.jackson.dataformat</groupId>
  <artifactId>jackson-dataformat-xml</artifactId>
  <version>2.12.5</version>
</dependency>
```

PART 2: Spring Boot Basics

Exercise 16: Using Spring Initializr

20 min

Using the Spring-Initializr (<https://start.spring.io/>) create a spring boot app (java 11 version) called "ex16-my-first-spring-boot-app" having web as dependency.



Project
☒ Maven Project ☐ Gradle Project

Language
☒ Java ☐ Kotlin ☐ Groovy

Spring Boot
☐ 2.6.0 (SNAPSHOT) ☐ 2.6.0 (M3) ☐ 2.5.6 (SNAPSHOT) ☒ 2.5.5
☐ 2.4.12 (SNAPSHOT) ☐ 2.4.11

Project Metadata

Group com.example

Artifact demo

Name ex16-my-first-spring-boot-app

Description Demo project for Spring Boot

Package name com.example.demo

Packaging ☒ Jar ☐ War

Java ☐ 17 ☒ 11 ☐ 8

- Create a simple `index.html` page just showing a short greeting message.
- Expose an endpoint **`/greeting`** to return "Hi Workshops Participants".
- Change the default Spring Boot banner shown below by providing a file `banner.txt`



to "Enjoy your evening", "on to a beer", "Spring Boot is cool ☺"

[Hint: use <http://bit.ly/2T2ShFU> or <http://bit.ly/2HfReeo>


- Just for fun, rename `animated-banner.gif` into `banner.gif` and start again! Surprise!

Exercise 17: AsmiqAcademyApp from Spring to Spring Boot


30 min

In exercise 14 we implemented a REST-Service for the AsmiqAcademyApp. Now migrate the entire AsmiqAcademyApp to Spring-Boot. To migrate to spring-boot do the following:


- Use the maven project "**ex17-asmiq-academy-boot-app-template**" as starting point and copy this into the project to "**ex17-asmiq-academy-boot-app**"
- Delete the highlighted packages in the above project. Why?

▼  ex17-asmiq-academy-boot-app


▼  src/main/java


>  ch.asmiq


>  ch.asmiq.config

>  ch.asmiq.controller

>  ch.asmiq.interfaces

>  ch.asmiq.model

>  ch.asmiq.service

>  ch.asmiq.tomcat

>  ch.asmiq.webinitializer

- Remove all maven dependencies in pom.xml!**

Just add the "spring-boot-starter-web" as the only dependency. Does it sound good?

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  <version>2.5.5</version>
</dependency>
```

Remove the original class AsmiqAcademyApp and create a new one like the following. Complete the TODOs in the AsmiqAcademySpringBootApplication class

```
//TODO
public class AsmiqAcademySpringBootApplication {
    public static void main(String[] args) {
        //TODO
    }
}
```

- Run the App and execute <http://localhost:8080/courses> and verify with the result:

```
[{"name":"Java (Online)","price":50,"quantity":2}, {"name":"Design
Patterns (Online)","price":60,"quantity":3}, {"name":"Testing
(Online)","price":40,"quantity":1}]
```

- One of customers required the courses response to be in XML instead of JSON. How would you do that? Try with suitable Accept-Headers and **produces =**

```
curl -H "Accept: application/json" http://localhost:8080/courses
curl -H "Accept: application/xml" http://localhost:8080/courses ]
```

```
<dependency>
  <groupId>com.fasterxml.jackson.dataformat</groupId>
  <artifactId>jackson-dataformat-xml</artifactId>
  <version>2.12.5</version>
</dependency>
```

Exercise 19: PersonApp

45 min

Create a Spring Boot app that provides a simple REST controller for a Person class:

- POST once with all attributes as parameters and
- POST with the data as a JSON string
- GET
- GET "/filter-older-than" and GET "/filter-by-nationality".

Use the following fragments as well as a list for data management as a starting point.

```
@RestController
@RequestMapping("/persons")
public class PersonController {

    private List<Person> persons = new ArrayList<>();

    ---

    public class Person
    {
        private String name;
        private String nationality;
        private int age;
        ...
    }
}
```

a) Insert different people by hand.

```
{ "name" : "Beat", "nationality" : "swiss", "age" : 35 }
{ "name" : "Peter", "nationality" : "german", "age" : 29 }
{ "name" : "Heinz", "nationality" : "german", "age" : 55 }
{ "name" : "Yannis", "nationality" : "german", "age" : 6 }
```

Use curl and inout data as follows:

```
curl -d '{ "name" : "Beat", "nationality" : "swiss", "age" : 35 }' \
-H "Content-Type: application/json" \
-X POST http://localhost:8080/persons

curl -d 'name=Mike&nationality=german&age=47' \
-X POST http://localhost:8080/persons/fromAttributes
```

b) Queries with curl

- Determine all persons.
- Determine all Swiss.
- Determine all persons older than 30.

c) Design a simple web page that provides these queries:



Ermittle alle Personen:

Ermittle alle Schweizer:

Ermittle alle Personen aelter als 30:

- d) Add some data already at the start-up of the application. Use the following modification to launch the application:

```
@SpringBootApplication
public class Application implements CommandLineRunner
{
    @Autowired
    private PersonController controller;

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

    public void run(String... args) throws Exception
    {
        // TODO
    }
}
```

Exercise 20: PersonApp - Architecture – Design - Layering

30 min

In exercise 19, we created a simple Spring Boot app with a REST controller and some endpoints. The goal was to understand the basic concepts and APIs prototypically. Analyze what could be possible weaknesses in the design! What architectural improvement potential do you see? Consider the following questions: How can data management be abstracted? How can functionality be provided in a reusable way? What about the single responsibility principle? What tasks does the controller perform, and how can this be divided appropriately?

PART 3: Spring Data Access

Exercise 21: PersonApp - JPA-Connection to DB

45 min

Exercise 21a: Now use a repository to connect to a database in the REST controller. What impact does this have on the two filterings? What is the best way to map these now? How does the Person class need to be modified?

```
@Autowired
```

```
private PersonRepository personRepository;
```

Denke an die weiteren Dependencies, etwa:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
  <version>2.5.6</version>
</dependency>
```

Exercise 21b: Integrate the H2 Web Console through the following WebConfiguration and entries in the application.properties to enable queries on the database and to be able to check the effects of commands.

```
@Configuration
```

```
public class WebConfiguration
```

```
{
```

```
    @Bean
```

```
    ServletRegistrationBean h2servletRegistration()
```

```
{
```

```
        final ServletRegistrationBean registrationBean =
            new ServletRegistrationBean(new WebServlet());
```

```
        registrationBean.addUrlMappings("/console/*");
```

```
        return registrationBean;
```

```
    }
```

```
}
```

Exercise 21c: Call the H2 Web Console with <http://localhost:8080/console>, and use `jdbc:h2:mem:testdb` as connection string for the JDBC URL. Execute SELECTs and familiarize yourself with the possibilities of the Web Console. Try INSERTs like this – what is problematic?

```
INSERT INTO PERSON (ID, NAME, NATIONALITY, AGE) VALUES (-10, 'Mike', 'german', 47);
INSERT INTO PERSON (ID, NAME, NATIONALITY, AGE) VALUES (-11, 'Karthi', 'german', 33);
```

Exercise 22: Discussion of design

20 min

What should a design look like with the knowledge of DAOs and REST applications? Sketch a REST controller, service, and data access with DAO or repository. How to handle conversion from entity to DTO.? Do you still need these? How is the mapping done? Take a look at the MapStruct tool:

<https://mapstruct.org/>

Connecting with MongoDB

Exercise 23: Modify the entity class so it can be used as a document for MongoDB. Modify the repository so that it is suitable for MongoDB. What changes are needed in the REST controller or application?

```
import org.springframework.data.annotation.Id;
```

Remember:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-mongodb</artifactId>
  <version>2.5.6</version>
</dependency>
```

Exercise 24: Install a MongoDB database tool and inspect the database. In which collection do the people data end up?

- <https://docs.mongodb.com/compass/master/install/>
- <https://nosqlbooster.com/downloads>

Exercise 25: Add some people, such as the following:

```
{ "name" : "Beat", "nationality" : "swiss", "age" : 35 }
{ "name" : "Peter", "nationality" : "german", "age" : 29 }
{ "name" : "Heinz", "nationality" : "german", "age" : 55 }
{ "name" : "Yannis", "nationality" : "german", "age" : 6 }
```

Enhance the repository to include actions like the following and run a few of these queries:

```
List<Person> findByAgeBetween(int lower, int upper);
int countByAgeBetween(int lower, int upper);
```

```
List<Person> findTop3ByAgeLessThan(int maxAge);
List<Person> findByAgeLessThanOrderByNameAsc(int maxAge);
```

```
List<Person> getByNameLike(String name);
```

```
List<Person> findByNameOrNationality(String name,
                                     String lastName);
```

```
List<Person> findByNameInAndAgeBetween(Collection<String> names,
                                       int lower, int upper);
```

Validation

Exercise 26: Validation

20 min

Given is a class **Customer.java**, so far without validation. This is now to be added and evaluated. The following conditions should apply:

- The first name is not empty and is between 2 and 100 characters long.
- The last name must not be empty
- The age must be positive or 0
- The date of birth must be in the past.
- The start of the next vacation must be today or in the future.
- Finally, the contact email should have a valid format.

To validate, expand and execute the **CustomerValidationExample.java** class.

The starting point is the project **ex26-spring-validation-app-template**.

MapStruct

Exercise 27: Mapping with MapStruct

20 min

Use the project **ex27-spring-mapstruct-app-template** as a template. There you will find some entity classes that are to be mapped to DTO classes with the same name. In addition, some partial view DTOs are to be created, which only provide excerpts from the entity values as an object.