Workshop:
Spring Basics

Übungen

Ablauf

Dieser Workshop gliedert sich in mehrere Vortragsteile, die den Teilnehmern die Thematik Spring Framework und Spring Boot mit Spring Data usw. näherbringen. Im Anschluss daran sind jeweils einige Übungsaufgaben von den Teilnehmern – idealerweise in Gruppenarbeit – am Rechner zu lösen.

Voraussetzungen

1) Aktuelles JDK 11, idealerweise auch JDK 17, installiert
2) Aktuelles Eclipse installiert (Alternativ: Spring Tool Suite oder IntelliJ IDEA)

Teilnehmer

- Entwickler und Architekten mit Java-Erfahrung, die ihre Kenntnisse zu Spring, Spring Boot und Data vertiefen möchten

Kursleitung und Kontakt

Michael Inden
Derzeit freiberuflicher Buchautor und Trainer
E-Mail: michael.inden@hotmail.com
Blog: https://jaxenter.de/author/minden

Weitere Kurse (Java, Unit Testing, ...) biete ich gerne auf Anfrage als Inhouse-Schulung an.

TAG 1: Spring Framework Basics

Aufgabe 1: Importieren Sie das Projekt `Ex01_TemplateApp` 10 min

Importieren Sie das Projekt `ex01-project-template`, starten die Applikationsklasse und klären Sie folgende einfachen Fragen:

1. Dieses Projekt hat eine transitive Abhängigkeit von spring-context. Richtig / Falsch
2. spring-context enthält die Implementierungen des IoC-Containers. Richtig / Falsch

[Tipp: Schauen Sie in die pom.xml und die Maven-Abhängigkeiten]

Aufgabe 2: Greeting App – Warm Up 25 min

Erweitern Sie das importierte Projekt oder legen Sie basierend darauf ein neues an. Danach sind folgende Aufgaben zu lösen:

a. Create a simple **GreetingService** class with a method **sayHello**() to print the following sentence "Welcome to Spring Workshop :-) "

b. 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.

c. 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`]

[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>
```

d. 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.]

Aufgabe 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

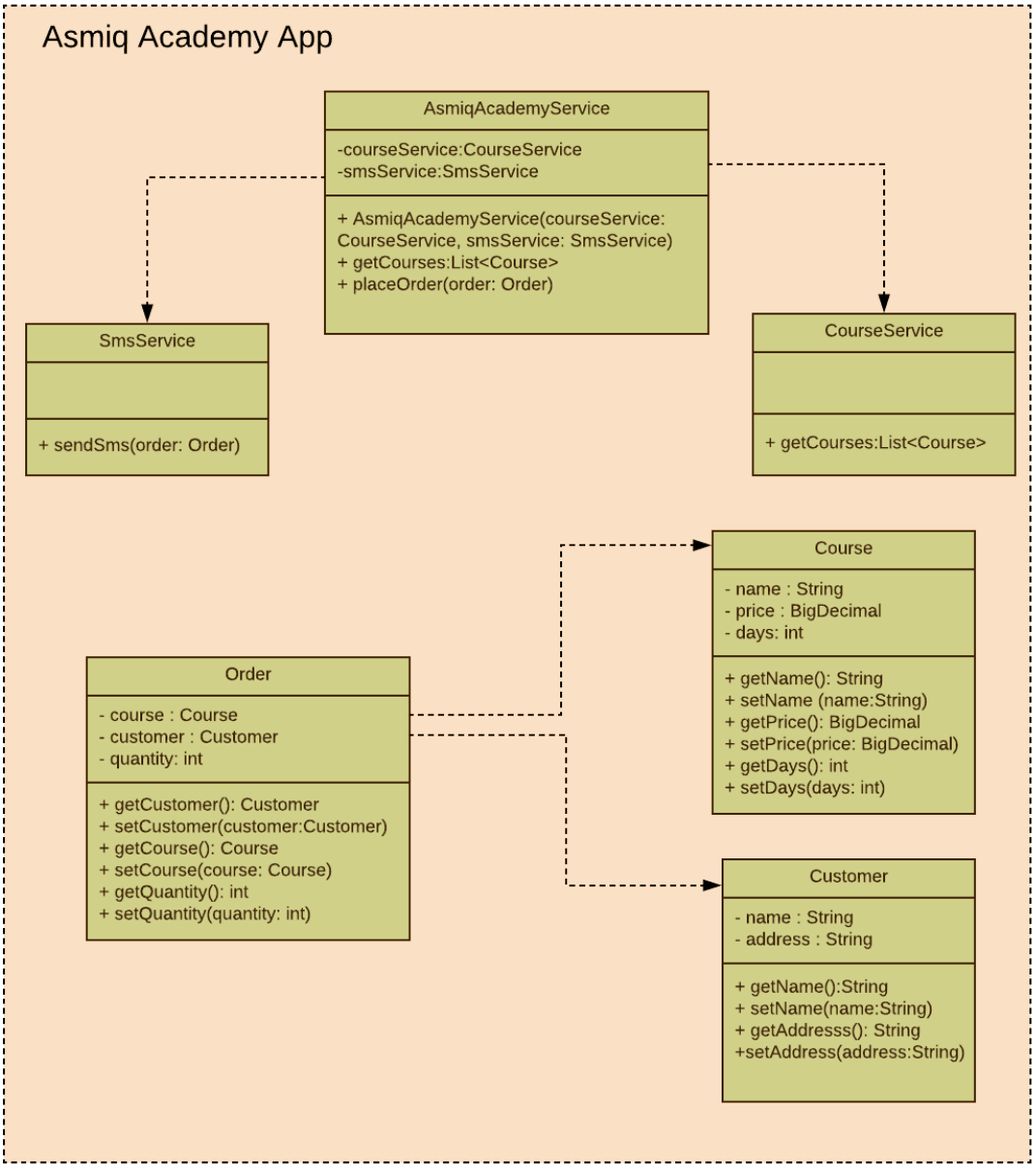


FIGURE 1

Implement the AsmiqAcademyApp by using the Spring IoC wherever possible through

I. XML configuration with autowiring
II. Java configuration with autowiring

[Hint: use the project template "**ex03-asmiq-academy-app-template**"]

Aufgabe 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.

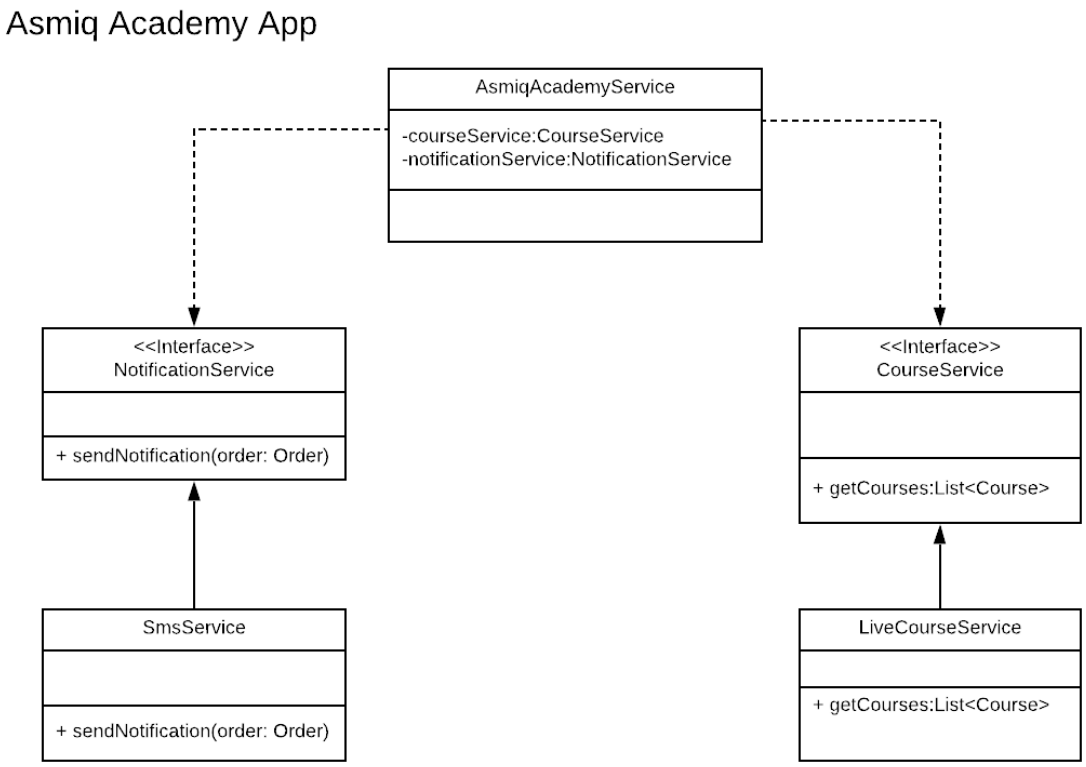


FIGURE 2

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.

Aufgabe 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.

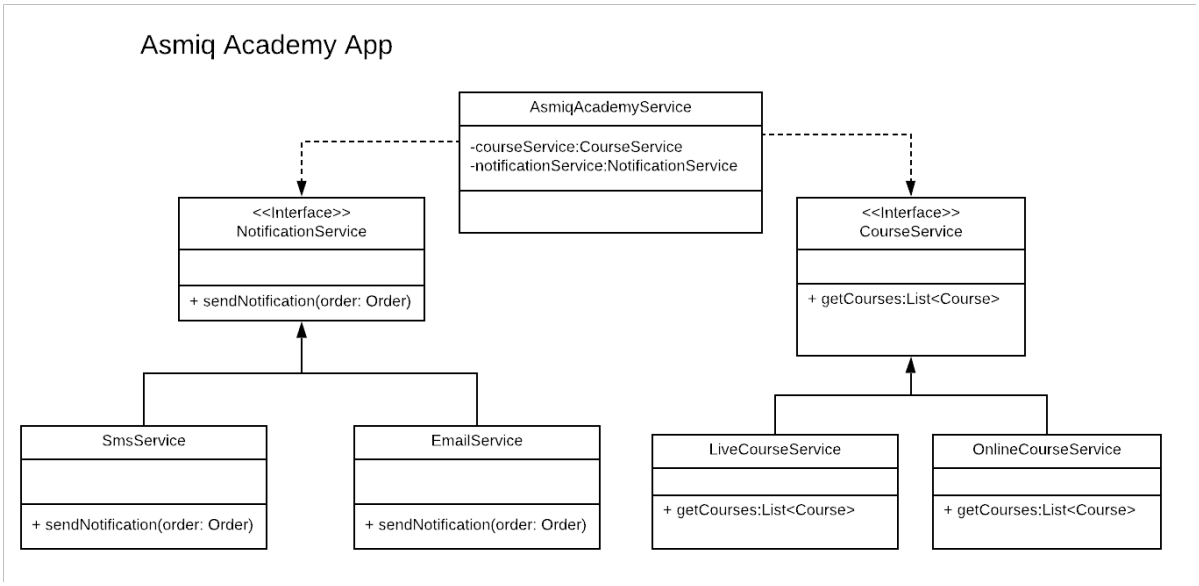


FIGURE 3

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 am Service, @Qualifier]

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

Aufgabe 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.

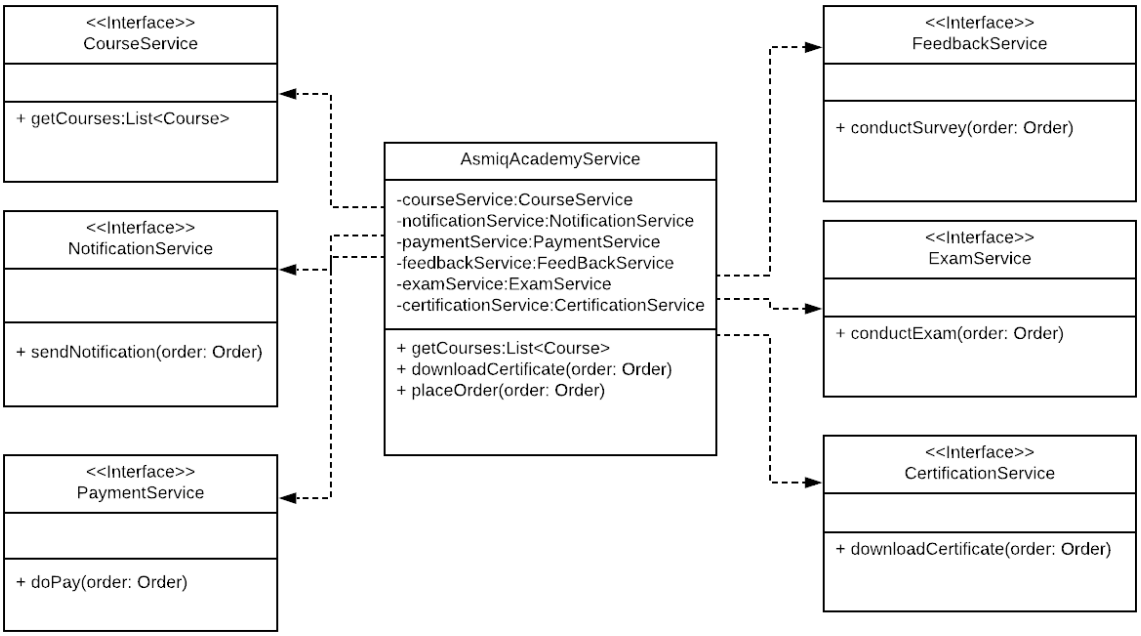


FIGURE 4

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: Diskussion setter-Injection / Konstruktor-Injection / Field-Injection

Aufgabe 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]

Aufgabe 8: Applikationsdesign verbessern – Prepare for Spring 30 min

Gegeben seien die Klassen eines Pizza-Service, die jedoch eng miteinander verbunden sind, da jeweils die Konstruktoren aufgerufen werden.

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

	public PizzaService() {
		// Direkte Abhängigkeiten
		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);
	}
}
```

Bei der Ausführung des Bestellvorgangs werden verschiedene Daten eines Kunden benötigt, dazu erfolgt ein Zugriff auf eine Datenbank. Zum Glück ist dies bereits durch ein sogenanntes Data Access Object (DAO) abstrahiert. Ebenso gibt es hin und wieder Rabattaktionen, die durch die Klasse Discount modelliert werden. Nachfolgend ist gezeigt, dass der `PizzaService` beide Abhängigkeiten durch Instanziieren der jeweiligen Klassen selbst auflöst (oder besser gesagt, selbst bereitstellt).

Wenn diese Applikation getestet oder im Spring-Kontext verwendet werden soll, so ist dieses Design störend. Als Aufgabe soll nun das Design adäquat angepasst werden.

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

Aufgabe 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>
```

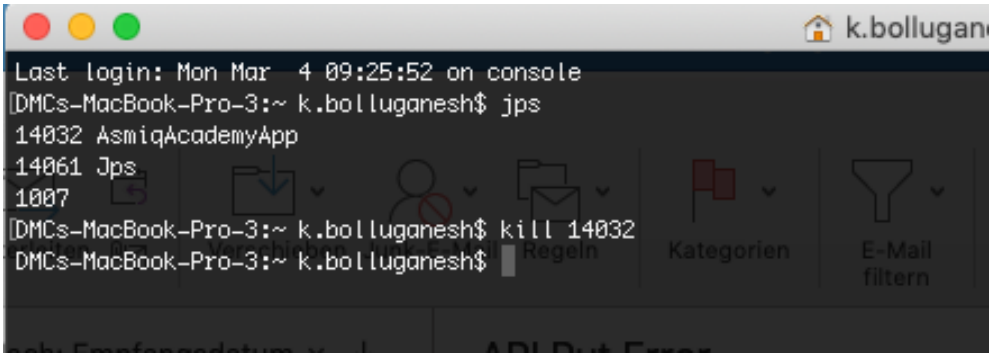
Aufgabe 10: Asmiq Academy App – Graceful Shut Down 15 min

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

a. modify our AsmiqAcademyApp to satisfy this requirement.

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

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



Aufgabe 11: Bean Lifecycle 15 min

Erstellen Sie ein Bean, dass alle möglichen Callbacks implementiert und deren Aufruf auf der Konsole protokolliert. Wie aktiviert man `@PostConstruct` und `@PreDestroy`?

[Hint: Use " **ex11-bean-lifecycle-template**" as the template]

[Hint: Study https://reflectoring.io/spring-bean-lifecycle/ for more *Aware-Interfaces|

PART Spring MVC

Aufgabe 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.

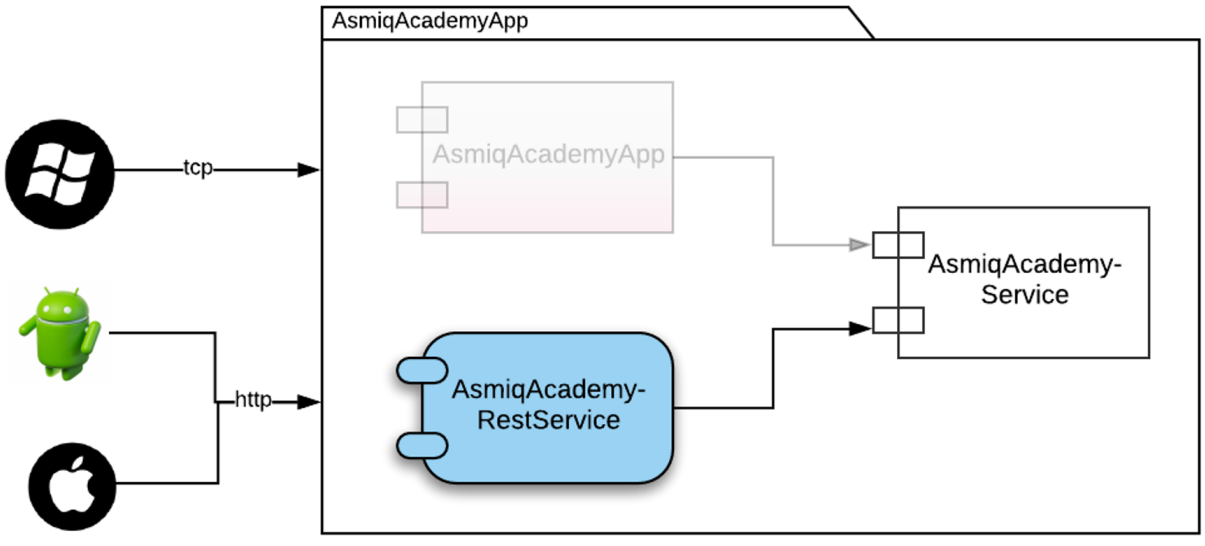
d) Modify and extend the `GreetingController` class to expose a REST endpoint "/hello" which returns the message "Welcome Spring Workshop Participants ☺". Run the app and call *http://localhost:7777/hello* and verify that the browser displays " Welcome Spring Workshop Participants ☺
[Hint: use "&#x1f605" 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?

Aufgabe 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.

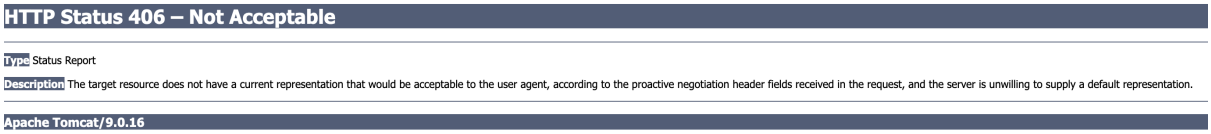


a. Import the maven project "**ex14-asmiq-academy-rest-app-template**" to do the following:

(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:

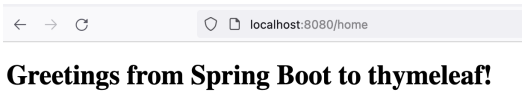



```
Sep. 26, 2021 1:51:48 NACHM.
org.springframework.web.servlet.handler.AbstractHandlerExceptionResolver
logException
WARNING: Resolved
[org.springframework.web.HttpMediaTypeNotAcceptableException: Could not
find acceptable representation]
```

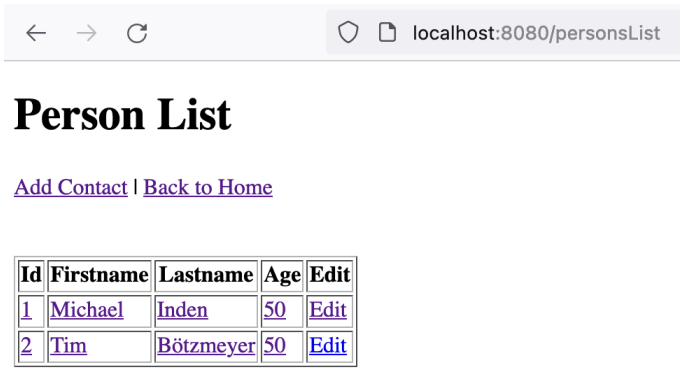
[Hint: Check pom.xml and Jackson Dependency]

Aufgabe 15: Einfache Applikation mit Thymeleaf 25 min

Die Applikation soll etwa einen einfachen Text in Form einer Webseite liefen, wenn man `/home` im Browser eingibt. Nutzen Sie das Maven-Projekt "**ex15-thymeleaf-app-template**" als Basis. Stellen Sie zudem eine Liste mit Personen dar:



Greetings from Spring Boot to thymeleaf!



[Hint: https://www.thymeleaf.org/doc/tutorials/3.0/thymeleafspring.html#integrating-thymeleaf-with-spring]
[Hint: https://spring.io/guides/gs/serving-web-content/ and additionally https://www.baeldung.com/thymeleaf-in-spring-mvc]

Nutzen Sie folgende Dependency:

```
<dependency>
      <groupId>org.thymeleaf</groupId>
      <artifactId>thymeleaf-spring5</artifactId>
      <version>3.0.12.RELEASE</version>
</dependency>
```