Exercise 5:
Introduction to Spring

Presentation of progress and results in next *two* labs /
(no submission to AULIS required, but individual presentation in lab
on 27.11.25 or –if 27.11. is skipped– 04.12.25)

Assignment 1: Understanding the Spring Demo Application "PetClinic"

The Spring Framework is a collection of small, well-focused, loosely coupled Java frameworks that can be used independently or collectively to build industrial strength applications of many different types. The PetClinic sample application (https://github.com/spring-projects/spring-petclinic) is designed to show how the Spring application frameworks can be used to build simple, but powerful database-oriented applications.

The users of the application are employees of the clinic who in the course of their work need to view and manage information regarding the veterinarians, the clients, and their pets. The sample application supports the following use cases:

- View a list of veterinarians and their specialties
- View information pertaining to a pet owner
- Update the information pertaining to a pet owner
- Add a new pet owner to the system
- View information pertaining to a pet
- Update the information pertaining to a pet
- Add a new pet to the system
- View information pertaining to a pet's visitation history
- Add information pertaining to a visit to the pet's visitation history

Your task:
Understand the basic structure and dynamics of the PetClinic application. You should be able to explain both to the audience.

Hints and steps:

- Download the PetClinic application from github:

  ```
  git clone https://github.com/spring-projects/spring-petclinic.git
  ```

- Open the project in your IDE, preferably using IntelliJ IDEA Ultimate (the Ultimate edition is free for students, but requires registration). When asked, please choose the Maven version.
- Part of the project is a file named `readme.md`. Read this file carefully! Among others, it contains information on how to run the PetClinic project from the command line or from an IDE.
- Build[1] and run the project; the PetClinic should then be available under http://localhost:8080 in your browser.
- Some overview information on the project's structure:
  - The project is built using the build management tool *Maven*. The Maven plugin should be installed in IntelliJ IDEA Ultimate; if not: install it!
  - Java code is contained in the package *org.springframework.samples.petclinic* and its sub-packages:
    - *model*: base classes that are used for the model classes in other packages
    - *owner*: model classes for owners, pets, and visits (together with a repository) as well as controller classes; controller classes implement the methods that are invoked when an action (button click, link) is triggered in the browser.
    - *vet*: model and controller classes for veterinarians.
  - Folder *resources/db*: contains SQL code for setting up the database schema (DDL) and inserting test data (DML). The PetClinic supports three different database types by default. The database management system is selected in the configuration file *application.properties* (*resources* folder). By default, a H2 in-memory database is used.
  - Folder *resources/messages*: contains the texts used in the PetClinic application in different languages, i.e. the PetClinic supports internationalization.
  - Folder *resources/static/resources*: contains static resources such as images, fonts, and – after building – also CSS files.
  - Folder *resources/templates*: contains HTML files used in the PetClinic web application. These HTML files contain *Thymeleaf* code. Thymeleaf is a template engine used for processing dynamic data and generating according HTML during runtime.
  - Folder *scss*: *Sass* is a stylesheet language that's compiled to CSS (e.g., *petclinic.css* in *src/main/resources/static/resources/css* was generated from the *petclinic.scss* source). The SCSS syntax uses the file extension .scss. With a few small exceptions, it's a superset of CSS, which means essentially all valid CSS is valid SCSS as well.
  - File *pom.xml*: declares the dependencies of the PetClinic on other projects / frameworks / libraries and plugins used for the build process.

[1] If you are using a Mac and encounter build problems that are related to the new "Apple silicon" (error message contains "incompatible architecture (have 'x86_64', need 'arm64')", see notes on last page.

Assignment 2: Changing the Spring Demo Application "PetClinic"

The second assignment builds upon the first and asks you to apply changes to the *PetClinic*. These changes will touch different aspects of the project:

- User interface / Thymeleaf code
- Representation of domain objects as model classes
- Behaviour encoded in controller classes
- Database schema and initial data

Your tasks:

1.) So far, the PetClinic only takes care of birds, cats, dogs, hamsters, lizards, and snakes. Extend the PetClinic's range of animal types by adding turtles and fish.

2.) Owners should have a property for the zip code. Apply according changes to the project such that owners with an additional property *zipCode* can be created, edited, presented, and persisted.

3.) Some owners are not satisfied with the PetClinic's services and expressed their will to never come back. Add a "Delete Owner" button to the UI at an appropriate place (e.g. on the owner details page or as part of the list of owners) and implement according behaviour in the backend.

4.) Still deleting owners: are the pets that belong to a deleted owner still "alive" in the system? Check using the database console provided via http://localhost:8080/h2-console/, where 8080 is the port the PetClinic is started.[2]
   a. Enter the JDBC URL that is printed to the IntelliJ run console upon project startup (looks like jdbc:h2:mem:7b3c1df7-a547-4cfc-808a-176342fa5aa5)[3]
   b. Use "sa" as user name and leave the password field empty.

   If the pets are still in the database, what could you do to correct this and delete the pets? If they are not: why?

5.) Since this is a pet clinic, pets are visited by veterinarians ("vets"). Users may add such visits on the owner's detail page; however, the visiting vet is not yet recorded. Modify the functionality (frontend and backend) for adding visits such that the vet can be selected from a dropdown list and is permanently stored with the visit. Since we want to have a look at a pet's "visits history" later, the table showing the previous visits needs to be adapted accordingly as well.

[2] If you should encounter an error, add the following line to the *application.properties*:
`spring.h2.console.enabled=true`

[3] You can set a fixed database URL using the `spring.datasource.url` property in *application.properties* to a valid JDBC URL of your choice.

Resources

- Spring and PetClinic:
  - https://spring.io and https://spring.io/projects/spring-boot
  - Craig Walls: "Spring in Action" / "Spring im Einsatz", online access available via https://www.suub.uni-bremen.de (you'll have to be in HSB network, from home using a VPN connection)
  - https://www.baeldung.com: various blog articles and courses on Spring.
- Thymeleaf: https://www.thymeleaf.org
- Sass: https://sass-lang.com/
- Maven: https://maven.apache.org

Build problems on Mac with "Apple silicon"

As it seems, the plugin used for generating css files out of scss sources is incompatible with newer Macs that use processors with ARM architecture. If you should want to compile changed scss code, you will encounter an error message that contains "(mach-o file, but is an incompatible architecture (have 'x86_64', need 'arm64'))".

Solution:

1. Open the file "pom.xml" in the project's root folder.
2. Comment (or remove) the <plugin> section for groupId *com.gitlab.haynes* and artifactId *libsass-maven-plugin*.
3. Copy the following plugin dependency into the file instead:

```
<plugin>
  <groupId>io.github.cleydyr</groupId>
  <artifactId>dart-sass-maven-plugin</artifactId>
  <version>1.1.0</version>
  <executions>
    <execution>
      <id>generate-css-using-sass</id>
      <phase>generate-resources</phase>
      <goals>
        <goal>compile-sass</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <inputFolder>${basedir}/src/main/scss/</inputFolder>

<outputFolder>${basedir}/src/main/resources/static/resources/css/</outp
utFolder>
    <loadPaths>${project.build.directory}/webjars/META-
INF/resources/webjars/bootstrap/${webjars-
bootstrap.version}/scss/</loadPaths>
  </configuration>
</plugin>
```

4. Update the Maven dependencies by clicking the button "Load Maven Changes" that should appear in the top right corner of the editor window.
5. Build the project.