# / JAVA DB SCHOOL
# Spring Boot 2

# First Spring Boot project

- https://start.spring.io/

- Dependencies:
  - Spring Data JDBC
  - MySQL Driver
  - Spring Web
  - Thymeleaf

# / Web application

# Static resources

- Web application = an application accessible from the web
- Preconfigured path
  - resources/static

# Templates

- resources/templates – HTML files
- Technologies: JSP (Java Server Pages), Thymeleaf, Groovy, Jade etc.
- Thymeleaf
  - Similar in syntax with HTML
  - Support for defining forms behavior
  - Binding form inputs to data models
  - Validation for form inputs
  - Displaying values from message sources
  - Rendering template fragments

# ModelAndView

- Interface for passing values to a template (view)

```
@GetMapping("/view")
public ModelAndView displayView() {
    ModelAndView modelAndView = new ModelAndView("viewPage");
    modelAndView.addObject("message", “abc");
    return modelAndView;
}
```

# Thymeleaf expressions (1)

- ${...} : Variable expressions. These are OGNL expressions (or model attributes in Spring)
- *{...} : Selection expressions, it will be executed on a previously selected object only
- #{...} : Message (i18n) expressions. Used to retrieve locale-specific messages from external sources
- @{...} : Link (URL) expressions. Used to build URLs
- ~{...} : Fragment expressions. Represent fragments of markup and move them around templates
- Reference: https://www.thymeleaf.org/doc/articles/standarddialect5minutes.html

# Thymeleaf expressions (2)

- Display value

```
<span th:text="${book.author.name}">
```

equivalent to

```
((Book)context.getVariable("book")).getAuthor().getName()
```

- Collections

```
<tbody>
    <tr th:each="student: ${students}">
        <td th:text="${student.id}" />
        <td th:text="${student.name}" />
    </tr>
</tbody>
```

# Thymeleaf expressions (3)

- Conditionals

<td th:text="${teacher.additionalSkills} ?: 'UNKNOWN'" /> (Elvis operator)

<td th:text="${teacher.active} ? 'ACTIVE' : 'RETIRED'" />

```
<td>
    <span th:if="${teacher.gender == 'F'}">Female</span>
    <span th:unless="${teacher.gender == 'F'}">Male</span>
</td>
```

# Thymeleaf expressions (4)

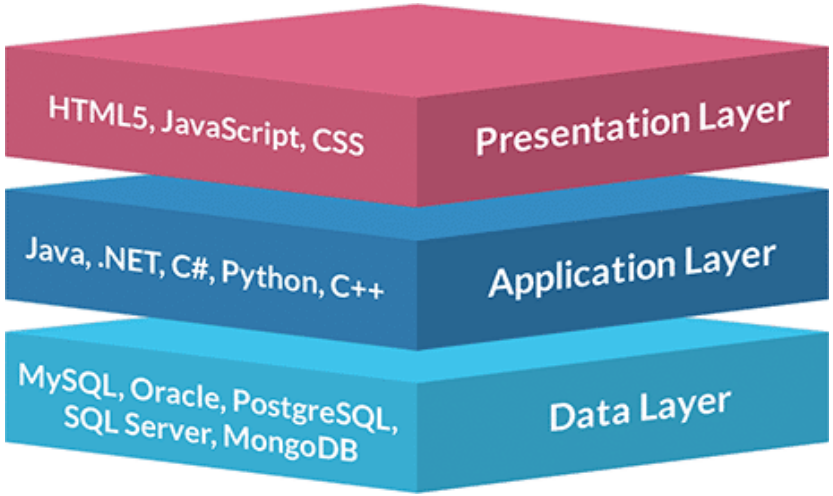
- Conditionals

```
<td th:switch="${#lists.size(teacher.courses)}">
    <span th:case="'0'">NO COURSES YET!</span>
    <span th:case="'1'" th:text="${teacher.courses[0]}"></span>
    <div th:case="*">
        <div th:each="course:${teacher.courses}" th:text="${course}"/>
    </div>
</td>
```

# / Three-tier architecture

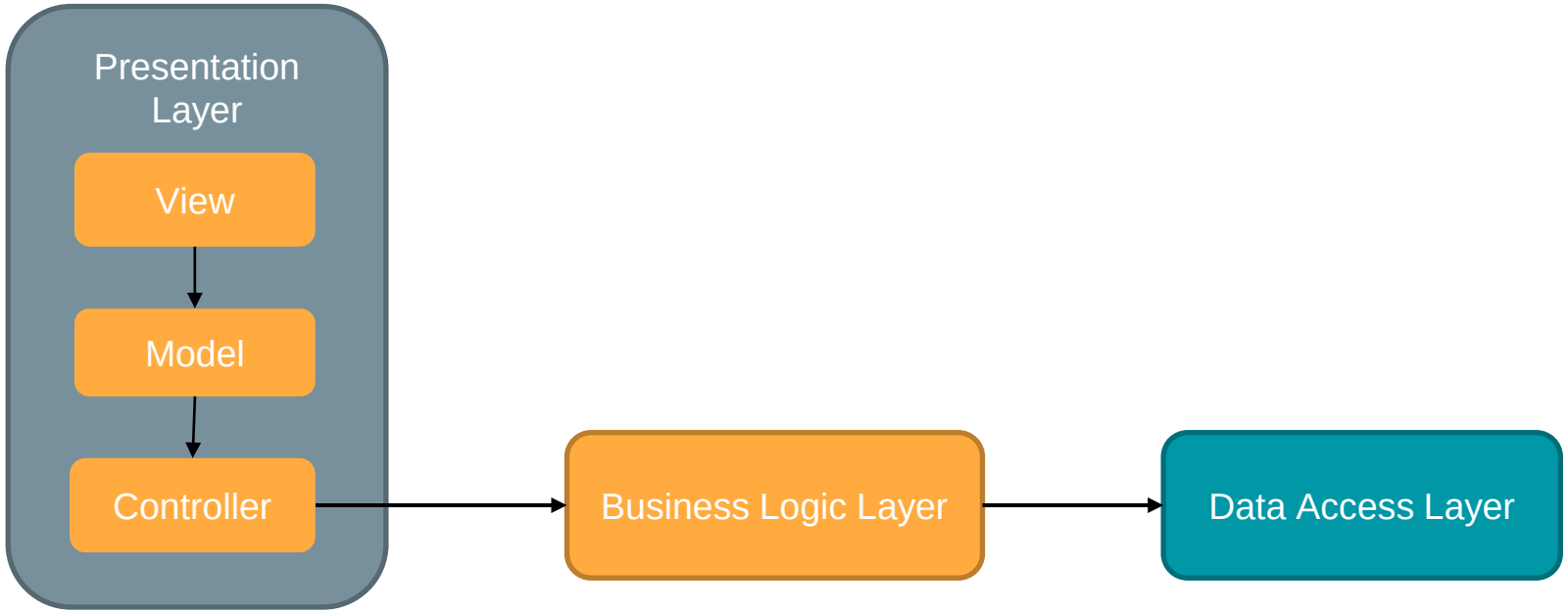
# Three-tier architecture (1)

- Presentation layer – user interface
- Business logic layer – application's core functionalities (making decisions, calculations, evaluations, and processing the data passing between the other two layers)
- Data access layer – interaction with databases to save and restore application data




Reference:
https://medium.com/@velusamyvenkatraman/lets-begin-with-3-tire-architecture-39c1b3f9fa51

# Three-tier architecture (2)



# / Spring three-tire architecture

# Semantic annotations

- Semantic annotations
  - @Controller
  - @Service – annotates classes at the service (business logic) layer
  - @Repository – annotates classes at the persistence layer
  - @Component – generic annotation for any Spring-managed component

# Recommendations

- Controller classes as the presentation layer. Keep this layer as thin as possible and limited to the mechanics of the MVC operations, e.g., receiving and validating the inputs, manipulating the model object, returning the appropriate ModelAndView object, etc.

- Service classes as the business logic layer. Calculations, data transformations, data processes, and cross-record validations (business rules) are usually done at this layer. They get called by the controller classes and might call repositories or other services

- Repository classes as data access layer. This layer’s responsibility is limited to Create, Retrieve, Update, and Delete (CRUD) operations on a data source, which is usually a relational or non-relational database. Repository classes are usually put in a repository package.

# / DAO Pattern

# Data Access Object

- Data Access Object Interface - This interface defines the standard operations to be performed on a model object(s)

- Data Access Object concrete class - This class implements above interface. This class is responsible to get data from a data source which can be database / xml or any other storage mechanism

- Model Object or Value Object - This object is simple POJO containing get/set methods to store data retrieved using DAO class

# / DTO Pattern

# Data Transfer Object

- Data Transfer Object – pattern that calls for the use of objects that aggregate and encapsulate data for transfer
- Should not contain any business logic but should contain serialization and deserialization mechanisms
- Can either contain all the data from a source, or partial data

# / Practice, practice, practice

# Practice

- Implement DAO pattern for the project – customer, product, orders

- Create simple HTML pages to display customers using Thymeleaf:
- All customers
- One customer by id
- Filter customers by username, city, country

# / Q&A


MOBILE/ACADEMY