Java Programming 3

Spring MVC: Controller details

KdG

Agenda this week

Last week - Project Review
Request parameters and path variables
Handling Form Data
Using a ViewModel
Converters
Server side validation
Sessions

KdG

Agenda this week

Last week - Project Review
Request parameters and path variables
Handling Form Data
Using a ViewModel
Converters
Server side validation
Sessions

KdG

Last week - Project Review

- Last week review:
  - Thymeleaf - Bootstrap
- Project Remarks:
  - What went well - What went wrong?
  - Did you tag the sprint?

KdG

Agenda this week

Last week - Project Review
Request parameters and path variables
Handling Form Data
Using a ViewModel
Converters
Server side validation
Sessions

KdG

Request Parameters: @RequestParam

- Used in the @Controller to extract parameters from the HTTP request
- Example:

```
http://localhost:8080/hello?name=jack
```

```
@Controller
public class HelloController {
   @GetMapping("/hello")
   public String sayHello(@RequestParam("name") String name, Model model) {
       model.addAttribute("helloname", "Hello " + name);
       return "hello";
   }
}
```

The request parameter with name “name” is mapped onto the name parameter of the sayHello method. Also works with POST.

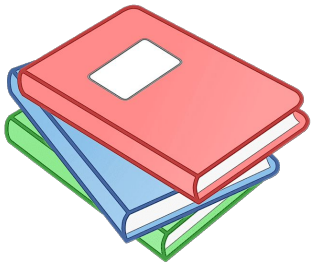
```
<body>
   <h1 th:text="${helloname}">Hello dummy</h1>
</body>
```

Thymeleaf code to show the “Hello Name” result...

KdG

Request Parameters: @RequestParam

- Remarks:
  - If the name of the request parameter matches the name of the method parameter, you don't have to provide the name to the @RequestParam annotation
  - You can make the parameter annotated with @RequestParam **optional**
    - by adding `@RequestParam(required = false)`
    - or by using a **Java Optional**: `@RequestParam Optional<String> name`
  - You can add a **default value:** `@RequestParam(defaultValue = "john doe")`
  - Map **all parameters** using `@RequestParam Map<String, String> allParams`
  - Multivalue parameters (eg: `http://localhost:8080/hello?name=jack,john,ann`): use a List as the parameter type: `@RequestParam List<String> name`
- Tutorial: https://www.baeldung.com/spring-request-param



KdG

Exercise 1: generate random numbers

- Create a small spring application that generates random numbers
- When you go to http://localhost:8080/randomnumber?maxvalue=100 it will return a page containing a random number between 0 and 100
- When you go to http://localhost:8080/randomnumber?maxvalue=30,100 it will return a page containing a random number between 30 and 100
- You can add an **optional** parameter `even=true`, in that case the returned number should be even. If no parameter even is provided or the parameter even=false, it should return any number in the range.

KdG

Path variables with @PathVariable

- Used in @Controller to handle template variables in the request URI mapping
- Demo:

```
@GetMapping("/hello/{name}")
public String sayHelloWithPath(@PathVariable("name") String name, Model model)
{
   model.addAttribute("helloname", "Hello " + name);
   return "hello";
}
```

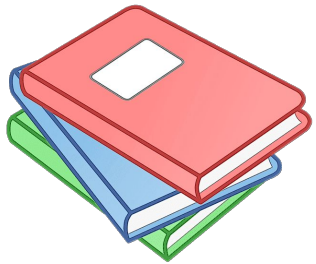
```
http://localhost:8080/hello/jack
```

The part of the URL that matches the template is extracted into the methods parameter.

KdG

Path variables with @PathVariable

- Remarks: very analogue to the @RequestParam remarks...
  - You can have more than 1 path variable in your path
  - You can map more than one path variable into a Map
  - The path variables are required by default, you can set the not required or use Optional
  - You can use default values
- Tutorial: https://www.baeldung.com/spring-pathvariable



KdG

Exercise 2: repeater

- Create a small web application that repeats a word.
- If you surf to http://localhost:8080/repeater/apple?repeat=10 it will show a page containing the repeated word:

KdG

Agenda this week

Project Review
Request parameters and path variables
Handling Form Data
Using a ViewModel
Converters
Server side validation
Sessions

KdG

Handle a form post: write a @PostMapping method

- Add parameters with same name as input fields of the form

```
@PostMapping("/students/add")
public String processAddStudent(String firstname, String lastname,
@DateTimeFormat(pattern = "yyyy-MM-dd") LocalDate birthdate, Double length, Integer
credits) {
   logger.info("Processing " + firstname + " " + birthdate + " " + length + " " +
credits);
   //…
   return "a
}
```

```
<form method="post">
   <label class="form-label" for="firstname">Firstname</label>
   <input type="text" class="form-control" id="firstname"  name="firstname">
   <label class="form-label" for="lastname">Lastname</label>
   <input type="text" class="form-control" id="lastname"  name="lastname">
   <label class="form-label" for="birthdate">Birthdate</label>
   <input type="date" class="form-control" id="birthdate"  name="birthdate">
   <label class="form-label" for="length">length</label>
   <input type="number" step="any" class="form-control" id="length"  name="length">
   <label class="form-label" for="credits">credits</label>
   <input type="number" step="any" class="form-control" id="credits"  name="credits">
   <button type="submit" class="btn btn-primary mt-2">Add</button>
</form>
```

KdG

Handle the form post: write a @PostMapping method

- If the names of the input fields do not match, you can use the @RequestParam annotation:

```
@PostMapping("/students/add")
public String processAddStudent(@RequestParam("fname") String firstname, String
lastname, @DateTimeFormat(pattern = "yyyy-MM-dd") LocalDate birthdate, Double length,
Integer credits) {
   logger.i
credits);
   //…
   return "
}
```

```
<form method="post">
   <label class="form-label" for="firstname">Firstname</label>
   <input type="text" class="form-control" id="firstname"  name="fname">
   <label class="form-label" for="lastname">Lastname</label>
   <input type="text" class="form-control" id="lastname"  name="lastname">
   <label class="form-label" for="birthdate">Birthdate</label>
   <input type="date" class="form-control" id="birthdate"  name="birthdate">
   <label class="form-label" for="length">length</label>
   <input type="number" step="any" class="form-control" id="length"  name="length">
   <label class="form-label" for="credits">credits</label>
   <input type="number" step="any" class="form-control" id="credits"  name="credits">
   <button type="submit" class="btn btn-primary mt-2">Add</button>
</form>
```

KdG

Handle the form: write a @PostMapping method

- The mapping between name attribute and field is case sensitive!
- Input type: text, email, password, … → maps to String fields
- Input type: number → maps to Integer fields (or Double)
- Do not use primitive types as a parameter (use Wrapper classes in stead)
- A select with multiple options maps to a List<String> of List<Integer>
- Input type: date → does not automatically map to LocalDate field!
  - Add the `@DateTimeFormat(pattern = "yyyy-MM-dd")` before the parameter
  - Or write a custom Converter

KdG

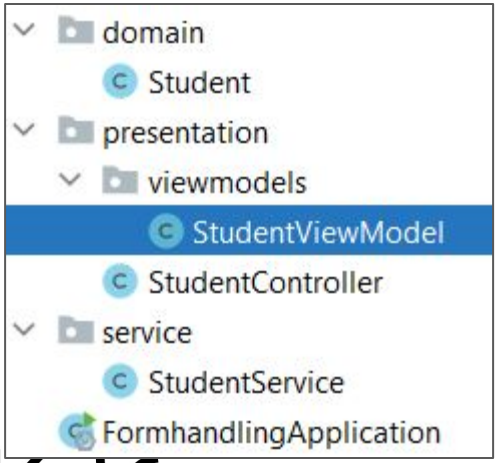
Agenda this week

Project Review
Request parameters and path variables
Handling Form Data
Using a ViewModel
Converters
Server side validation
Sessions

KdG

ViewModel

- A nicer way is to use a dedicated class that contains all the parameters of the form.
- We call this class a ViewModel. Let's put it in a `viewmodel` subpackage of the `presentation` package




```
public class StudentViewModel {
    private String firstname;
    private String lastname;
    private @DateTimeFormat(pattern = "yyyy-MM-dd") LocalDate birthda
    private double length;
    private int credits;

    //Getters and setters
    //toString method
    //…
```

KdG

The @PostMapping method becomes cleaner

- Spring will use the constructor or setters of the viewmodel to set the different input values

```
@PostMapping("/students/add")
public String processAddStudent(StudentViewModel viewModel) {
   logger.info("Processing " + viewModel);
   studentService.addStudent(viewModel.getFirstname() + ' ' + viewModel.getLastname(),
                                    viewModel.getBirthdate(),
                                    viewModel.getLength(),
                                    viewModel.getCredits());
   return "addstudent";
}
```

KdG

Use the ViewModel in Thymeleaf

- You can add the ViewModel to the Model in the GET request

```
@GetMapping("/students/add")
public String getAddStudent(Model model) {
   model.addAttribute("student", new StudentViewModel());
   return "addstudent";
}
```

```
<form method="post" th:object="${student}">
       <label class="form-label" th:for="firstname">Firstname</label>
       <input type="text" class="form-control"  th:field="*{firstname}">
       <label class="form-label" th:for="lastname">Lastname</label>
       <input type="text" class="form-control"  th:field="*{lastname}">
       <label class="form-label" th:for="birthdate">Birthdate</label>
       <input type="text" class="form-control"  th:field="*{birthdate}">
       <label class="form-label" th:for="length">length</label>
       <input type="text" class="form-control"  th:field="*{length}">
       <label class="form-label" th:for="credits">credits</label>
```

Advantage: you can use th:field in the input elements, it will translate to id=”...” and name=”...”. That way the names of the requestparameters will always match the viemodel’s fields!

Mapping form submits to the ViewModel: use `th:field`

```
<form method="post" th:object="${userViewModel}">
   <input type="text" th:field="*{firstname}">
   <button type="submit">Submit</button>
</form>
```

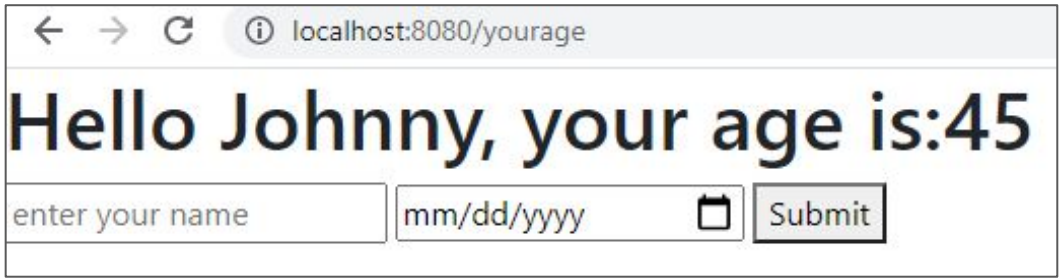
```
public class UserViewModel {
   private String name;
   private String firstname;

   //getters and setters...
}
```

```
@GetMapping
public String getSimpleForm(Model model){
   model.addAttribute("userViewModel", new UserViewModel());
   return "simpleform";
}
@PostMapping
public String postSimpleForm(UserViewModel userViewModel){
   //use the ViewModel
   return "simpleform";
}
```

KdG

Exercise 3: calculate the age

- Create a small web application (only presentationlayer!) that has one form: you can enter your name and your birthday
- Create a ViewModel: PersonViewModel that has a name (String) and a birthday (LocalDate)
- Implement this functionality: when a user submits his name and birthday, the form shows his name and age



KdG

Agenda this week

Project Review
Request parameters and path variables
Handling Form Data
Using a ViewModel
Converters
Server side validation
Sessions

KdG

Writing a custom Converter

- Sometimes you want to use custom conversion of the requestparameters to fields of the viewmodel (for example: if the field is an enum type…)
- Writing a custom converter involves two steps:

1. Implement the `Converter<S,R>` interface
2. Register this converter to the `FormatterRegistry`:
   a. Create a `@Configuration` class that implements the `WebMvcConfigurer` interface
   b. Override the `addFormatters` method of this interface: add an instance of your converter to the `FormatterRegistry` parameter

KdG

Custom Converter: demo

- We will write a converter for an enum type `StudentType`. It will convert a `String` input to a `StudentType`

```
public class StudentViewModel {
    //…
    private StudentType studentType;

    //getters and setters
}
```

```
public enum StudentType {
    ACS, REGULAR, FLEX
}
```

Let's say our form uses a text field to enter the student type. Can we write a converter to convert this to the enum type?

```
<form method="post">
    <input type="text" name="studentType">
    <button type="submit">Submit</button>
</form>
```

KdG

Custom Converter: demo

We implement the Converter interface. Our implementation is very tolerant: case insensitive and only the first 3 letters should be correct...

- We will write a converter for an enum type `StudentType`. It will convert a `String` input to a `StudentType`

```
import org.springframework.core.convert.converter.Converter;

public class StringToStudentTypeConverter implements Converter<String, StudentType> {
   @Override
   public StudentType convert(String source) {
       return switch (source.substring(0,Math.min(source.length(),3)).toUpperCase()){
           case "ACS" -> ACS;
           case "FLE" -> FLEX;
           default    -> REGULAR;
       };
   }
}
```

KdG

Custom Converter: demo

We register this converter via a @Configuration class that implements the WebMvcConfigurer interface.
We could have used the `@SpringBootApplication` class, but we prefer a separate class for all web configuration.

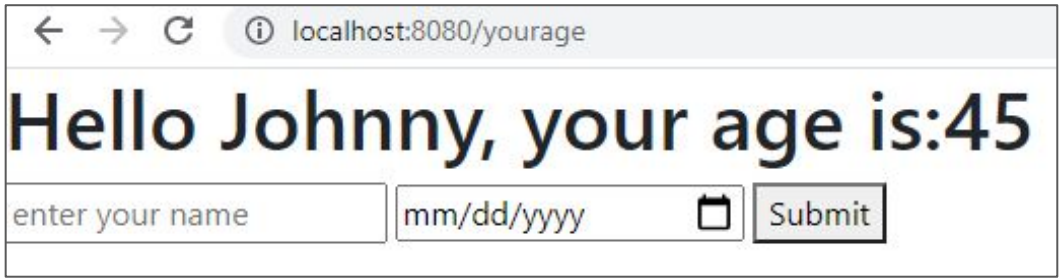
- We will write a converter for an enum type `StudentType`. It will convert a `String` input to a `StudentType`

```
package be.kdg.programming3.fromhandling.configuration;

import...

@Configuration
public class WebConfig implements  WebMvcConfigurer {
    @Override
    public void addFormatters(FormatterRegistry registry) {
        registry.addConverter(new StringToStudentTypeConverter());
    }
}
```

KdG

Exercise 4: writing a converter

- Alter the small web application from exercise 3: the PersonViewModel has a name (String) and an age (Integer)
- Use a custom converter to do the date to age conversion!
- Functionality should stay the same…



KdG

Agenda this week

Project Review
Request parameters and path variables
Handling Form Data
Using a ViewModel
Converters
Server side validation
Sessions

KdG

Server-Side Validation

- Add validation to the ViewModel objects:
  - Implement in the setter methods?
  - Better: use Bean Validation!
- Bean Validation: https://beanvalidation.org
- Add this to the build.gradle:
  - `implementation("org.springframework.boot:spring-boot-starter-validation")`
- We can now annotate the fields of the ViewModel!

Or choose **I/O** → **Validation** as a dependency when you create a new project!

KdG

Server-Side Validation example: StudentViewModel

```
public class StudentViewModel {
    @NotBlank(message = "name is mandatory")
    @Size(min=3, max=100, message = "Name should have length between 2 and 100")
    private String name;

    @NotNull(message = "birthday is mandatory")
    @Past(message = "your birthday should be in the past")
    private LocalDate birthday;

    @Max(value=60, message = "You can enter a maximum of 60 credits")
    private int credits;

    @NotBlank(message = "country is mandatory")
    private String country;

    //getters and setters
}
```

What constraints are possible?

- Check the specification
- Overview of the annotations

KdG

In the @Controller studentController:

```
@GetMapping
public String getStudentForm(Model model){
   model.addAttribute("student", new StudentViewModel());//we add an empty
object
   return "student";
}
@PostMapping
public String handleStudent( @Valid StudentViewModel studentViewModel,
                             BindingResult errors, Model model){
   if (errors.hasErrors()) {
       errors.getAllErrors().forEach(error->{
           logger.error(error.toString());
       });
       model.setAttribute("student", studentViewModel);
       //show the form again
       return "student";    }
   return "formok";
}
```

@Valid takes care of the validation.
BindingResult contains the errors if any

Shortcut: use @ModelAttribute

```
@GetMapping
public String getStudentForm(Model model){
   model.addAttribute("student", new StudentViewModel());//we add an empty
object
   return "student";
}
@PostMapping
public String handleStudent( @Valid @ModelAttribute("student") StudentViewModel
studentViewModel, BindingResult errors){
   if (errors.hasErrors()) {
       errors.getAllErrors().forEach(error->{
           logger.error(error.toString());
       });
       //show the form again
       return "student";    }
   return "formok";
}
```

@ModelAttribute links the studentViewModel back to the form: it adds the studentViewModel as an attribute with name “student” to the model.

KdG

The Thymeleaf form...

```
<form th:object="${student}" method="post">
   <div th:if="${#fields.hasAnyErrors()}" class="alert alert-danger">
       <p>The form contained errors!</p>
   </div>
   <input type="text" th:field="*{name}" placeholder="enter name"><br>
   <input type="date" th:field="*{birthday}" placeholder="select
birthday"><br>
   <input type="number" th:field="*{credits}" placeholder="enter number of
credits"><br>
   <input type="text" th:field="*{country}" placeholder="enter country"><br>
   <button type="submit">Submit</button>
</form>
```

Use ${#fields.hasAnyErrors()} to check for all validation errors. You can also check for a specific field: ${#fields.hasErrors('birthday')}
${#fields.errors('birthday')} returns the errors. You can use this to show the feedback to the user (using a th:each over the errors).
You can also use the th:errors attribute: see the tutorial in next exercise…!

KdG

Exercise 5: follow the tutorial!

- This guide walks you through the process of configuring a web application form to support validation
- https://spring.io/guides/gs/validating-form-input/

KdG

Remark:

- To add simple views (no programming logic in the controller method), add to your webconfiguration:

```
@Override
public void addViewControllers(ViewControllerRegistry registry) {
    registry.addViewController("/results").setViewName("results");
}
```

KdG

Agenda this week

Project Review
Request parameters and path variables
Handling Form Data
Using a ViewModel
Converters
Server side validation
Sessions

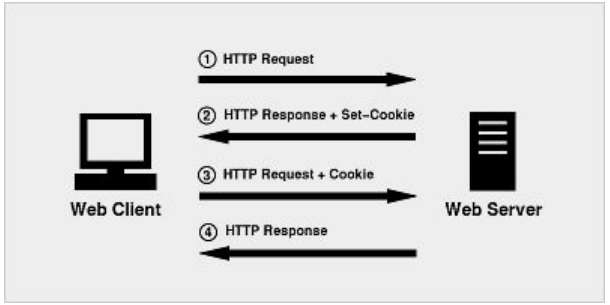
KdG

Sessions: the web is *stateless*

- A web application is stateless: each request to the web server is a completely new request, there is no ‘state’ existing between request
  - As opposed to a desktop application: when you navigate through a desktop application a state is kept in memory. (eg. in a game you can navigate to the settings page, while your complete game-state (score, health, etc..) is kept in memory)
- This can be a problem: you often want to keep some state of the user of your web application between different requests.

KdG

Sessions: the web is *stateless*



- There are workarounds for this problem
  - You can save some state in cookies (in the browser of the user)
  - You can pass on some state info via URL parameters
  - …
- In Java web servers (like Tomcat) these workarounds are implemented using an `HttpSession` interface.
- HttpSession hides the implementation details (like cookies)
- You can use the HttpSession object to save and retrieve state of your users

KdG

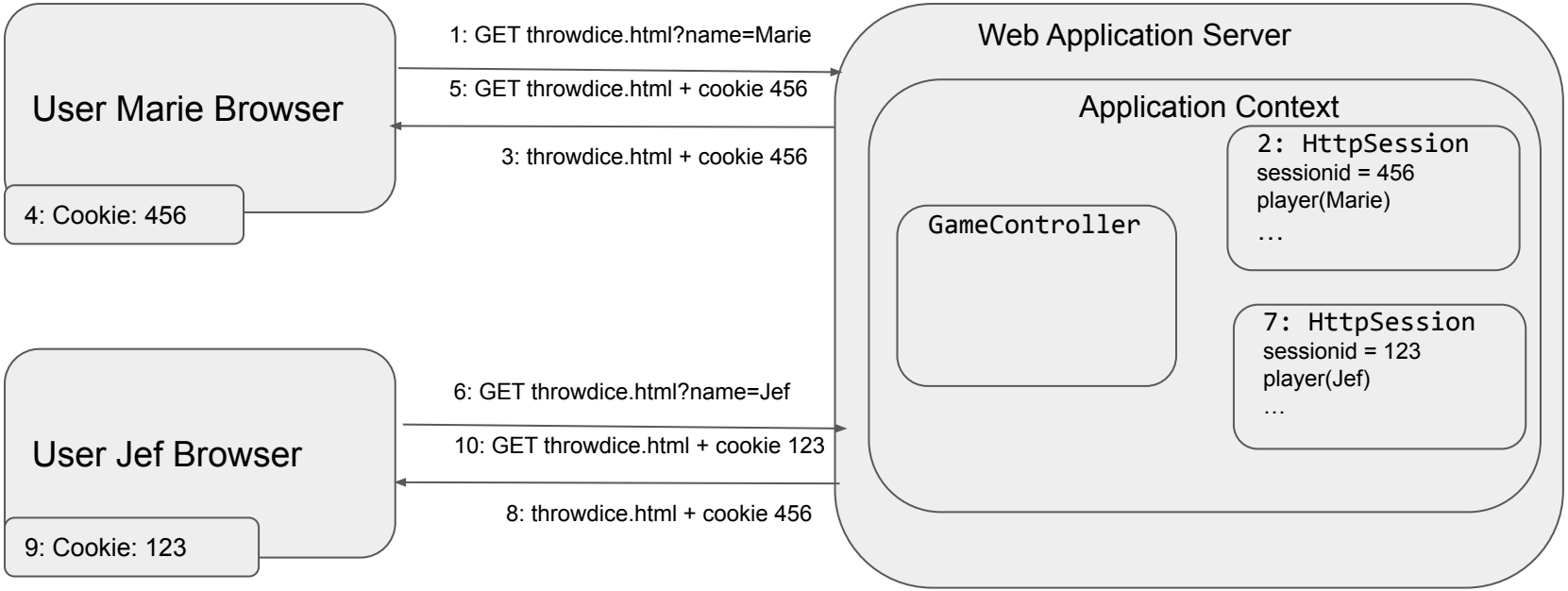
Throw Dice Game

- Let’s create a small dice-throwing game…
- User can enter his/her name
- User can throw dice
- Per user we keep:
  - The username
  - The user total score
  - The last throw (2 values)

this is the *state* of the user

KdG

HttpSession: how?



Throw Dice Game: implementation with HttpSession

- Add HttpSession as a parameter to the Controller methods
  - If there was no session yet, HttpSession will be created (and stored in application context)
  - Otherwise, correct HttpSession object will be retrieved from the application context
- You can add state data to the HttpSession object (using setAttribute)
  - We will add a Player object containing the player state info (name, totalScore, …)
- In the Thymeleaf template an object named ‘session’ can be used to retrieve and use the state info in the HTML page.
- The sessionid cookie (JSESSIONID) will be passed on between the client’s browser and the application server

KdG

Throw Dice Game: use @SessionScope

- You can also make the Player object a Spring Component with scope: @SessionScope
- Add it to the Controller as an autowired attributed
  - Spring will assure that for each session a different player object is created! (it uses HttpSession behind the scenes…)
- You can add the player object to the Model object in your Controller methods
- You can use it in the Thymeleaf template as a normal object (no session object needed)
- The sessionid cookie (JSESSIONID) will be passed on between the client's browser and the application server

KdG

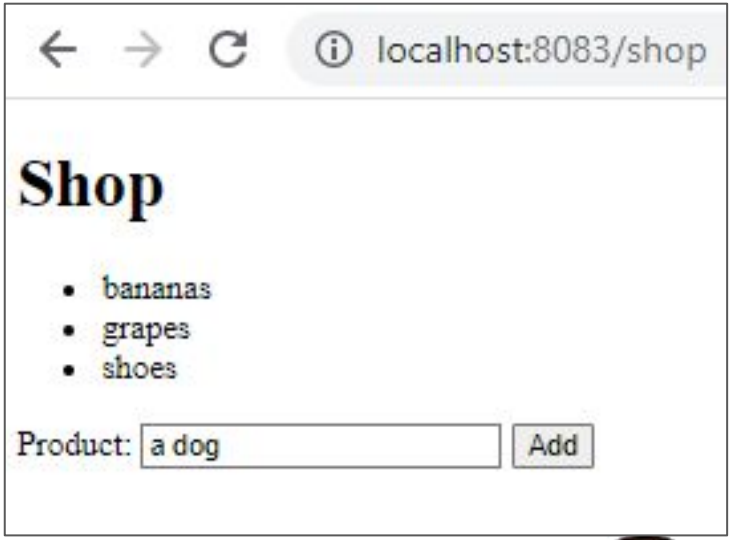
Session: remarks

- You could also AutoWire an HttpSession object as an attribute to the controller
- There is a separate Spring project called **Spring Session**
  - It add lot's of session functionality like storing sessions in database, caching, …
- Session cookies can be used for authenticating a user
  - You need extra security measures (eg json web tokens)!
  - There is Spring project called **Spring Security** that can help with that
- Spring Session and Spring Security will be covered in later courses…

KdG

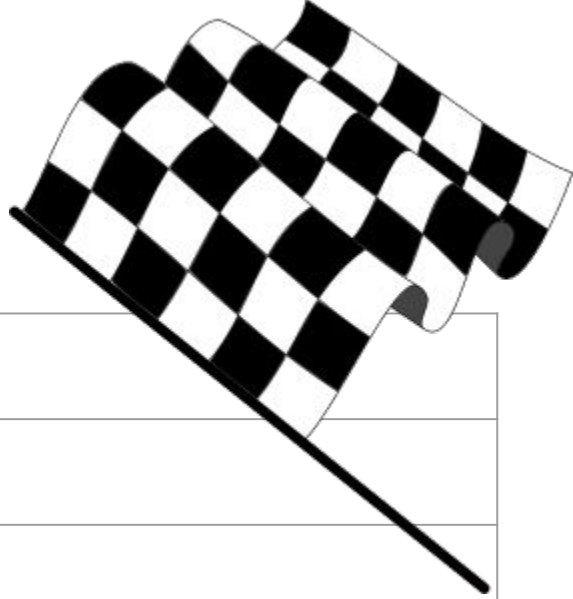
Exercise 6: Workout a small shopping cart application

- You can add products (Product) to a list (ShoppingCart)
- A Product has a name (String)
- A ShoppingCart has a List of Products
- You save the ShoppingCart of the user in his/her session
- Can you implement it both ways? (httpsession parameter or @SessionScope objects)



KdG

Agenda this week



Project Review
Request parameters and path variables
Mapping to the ViewModel
Converters
Server side validation
Sessions

KdG

Project

- Work out the presentation layer for your 2 main entities:
  - Work out ViewModel objects for your 2 forms
  - Work out at least one custom Converter for a field of a ViewModel
  - Add validation to your project using Bean Validation 2.0:
    - Add annotations to the fields of the ViewModels
    - Add messages to the annotations
    - Ensure that the messages are shown in the form when the user enters invalid data
- Add a “session history” page to the application:
  - The pages shows an overview of what pages were visited by the current user at what timestamp
    - You use sessions to implement this…

KdG