Spring Framework

Przygotowanie projektu

Spring Initializr

- Szkielet projektu przygotujemy za pomocą narzędzia Spring Initializr:
  - wejdź na adres https://start.spring.io/
  - wybierz typ projektu: **Maven Project**, wersja Spring Boot: **1.5.***
  - jako metadane projektu wpisz: **Group** `wizut.tpsi`, **Artifact** `spring-lab1`
  - dodaj następujące zależności (*Dependencies*):
    - Web
    - Thymeleaf
    - DevTools
  - kliknij **Generate Project**

Otwarcie projektu w Netbeans

- Pobrany plik `spring.lab1.zip` rozpakuj do dowolnego folderu.
- W folderze znajdziesz standardowy projekt dla narzędzia **Maven**. Taki projekt można otworzyć lub zaimportować do każdego współczesnego IDE.
- Uruchom **Netbeans**, otwórz projekt (**File** -> **Open Project...** -> wskaż folder `spring.lab1`)

Zmiany w deskryptorze projektu

- Otwórz plik `pom.xml` - znajdziesz go w drzewku projektu w gałęzi **Project Files**
- Jeśli używasz Javy starszej niż 1.8 (np. 1.7 na komputerach w pracowni), konieczne jest wskazanie Mavenowi wersji Javy
  - znajdź odpowiedni fragment w `pom.xml` i zmień 1.8 na 1.7:

```
<java.version>1.7</java.version>
```

- Jeśli chcemy, by zmiany w szablonach stron były automatycznie wykrywane przez aplikację po zapisaniu pliku, musimy wskazać folder z szablonami (*resources*)
  - znajdź element `build`
  - tuż za definicją pluginów dodaj element `resources`, wskazując w nim folder `src/main/resources`:

```
<build>
    <plugins>
        ...
    </plugins>
    <resources>
```

```
        <resource>
            <directory>src/main/resources</directory>
        </resource>
    </resources>
</build>
```

- Zbuduj projekt (prawy klawisz na nazwie projektu, **Build**)

Spring MVC - obsługa strony głównej

Kontroler

- Dodaj do projektu nową klasę o nazwie `HomeController`, w pakiecie `wizut.tpsi`
- Dodaj do klasy adnotację `@Controller`

```
@Controller
public class HomeController {

}
```

- Dodaj metodę `home`, obsługującą stronę główną:

```
@Controller
public class HomeController {

    @RequestMapping("/")
    public String home() {
        return "home";
    }

}
```

  Metoda `home` zwraca nazwę widoku, jaki ma zostać wyrenderowany przy wejściu na stronę główną. Przy domyślnych ustawieniach projektu oznacza to szablon `home.html`.

Widok

- W drzewku projektu rozwiń gałąź **Other Sources** -> `src/main/resources`
- W folderze `templates` stwórz plik HTML `home.html`, umieść w nim następujący kod:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>Pierwszy projekt springowy</title>
```

```
        <meta charset="UTF-8" />
    </head>
    <body>
        <h1>Spring wita!</h1>
    </body>
</html>
```

- Uruchom projekt, wejdź w przeglądarce na adres http://localhost:8080

Przekazywanie danych do widoku

Kontroler

Jeśli z kontrolera chcemy przekazać do widoku dane, musimy umieścić je w modelu jako atrybuty. Obiektem reprezentującym model zarządza Spring Framework - musimy tylko poprosić o udostępnienie go nam.

- Do metody `home()` dodaj parametr `model`:

```
@RequestMapping("/")
public String home(Model model) {
    return "home";
}
```

- Dodaj do modelu atrybut `imie`, jako wartość podaj swoje imię:

```
@RequestMapping("/")
public String home(Model model) {
    model.addAttribute("imie", "Kleofas");

    return "home";
}
```

- Tak dodany atrybut będzie dostępny z poziomu widoku pod nazwą `imie`.

Widok

- W widoku `home.html` zmieńmy komunikat powitalny:

```
<h1>Cześć Anonim!</h1>
```

- Zamiast *Anonim* chcemy wypisać imię przekazane z kontrolera. Użyjemy w tym celu elementów biblioteki **Thymeleaf**
- "Opakuj" tekst *Anonim* w element `span`:

```
<h1>Cześć <span>Anonim</span>!</h1>
```

- Dodaj do `span` atrybut `th:text`:

```
<h1>Cześć <span th:text="${imie}">Anonim</span>!</h1>
```

- Atrybut `th:text` spowoduje zastąpienie zawartości elementu `span` wartością podanego atrybutu (w tym przypadku: atrybutu `imie`).
- Sprawdź działanie aplikacji.

Obsługa formularzy

- Na stronie głównej umieścimy formularz z imieniem, zaś komunikat powitalny przenieśmy do nowego widoku.
- Zrób kopię szablonu `home.html`, zapisz ją pod nazwą `hello.html`
- Usuń z `home.html` komunikat powitalny, zastąp go formularzem:

```
<h1>Przedstaw się ładnie:</h1>
<form action="/hello" method="post">
    Imię: <input type="text" name="imie" />
    <input type="submit" value="Wyślij" />
</form>
```

- W kontrolerze `HomeController`:
  - usuń z metody `home()` odniesienia do modelu
  - dodaj nową metodę `hello()`, zmapowaną na adres `/hello`:

```
@RequestMapping("/hello")
public String hello(Model model) {

    return "hello";
}
```

- W metodzie `hello()` chcemy pobrać wartość parametru `imie`, wpisanego przez użytkownika w formularzu i przesłanego w żądaniu HTTP. Spring Framework może automatycznie zmapować parametry żądania na argumenty metody, jeśli mają one takie same nazwy. Dodaj do metody `hello()` parametr `String imie`:

```
@RequestMapping("/hello")
public String hello(Model model, String imie) {

    return "hello";
}
```

- Następnie dołącz imię do modelu:

```
@RequestMapping("/hello")
public String hello(Model model, String imie) {
    model.addAttribute("imie", imie);

    return "hello";
}
```

- W podobny sposób można przekazać parametry innych typów (np. `Integer`, `Double`), zostaną one automatycznie skonwertowane. Dodajmy do formularza na stronie `home.html` pole z wiekiem:

```
<form action="/hello" method="post">
    Imię: <input type="text" name="imie" />
    Wiek: <input type="number" name="wiek" />
    <input type="submit" value="Wyślij" />
</form>
```

- W kontrolerze w metodzie `hello()` dodajmy argument `Integer wiek` i przekażmy jego wartość do widoku:

```
@RequestMapping("/hello")
public String hello(Model model, String imie, Integer wiek) {
    model.addAttribute("imie", imie);
    model.addAttribute("wiek", wiek);

    return "hello";
}
```

- Wreszcie wyświetlmy wiek na stronie `hello.html`:

```
<p>
    <span th:text="${imie}">Anonim</span> twierdzi, że ma <span th:text="${wiek}">20</s
</p>
```

Zadanie 1

- Przygotuj prosty kalkulator z operacją dodawania:
  - na stronie głównej formularz z dwoma polami `x` i `y`, wpisujemy dwie liczby całkowite;
  - przycisk 'Dodaj';
  - formularz przesyłany do nowego kontrolera `CalculatorController`;
  - wynik dodawania wyświetlany w widoku w postaci `3 + 5 = 8`

Zadanie 2

- Rozbuduj kalkulator:
  - dodaj nowy formularz, z listą pozwalającą na wybranie operacji (dodawanie, odejmowanie, mnożenie);
  - formularz przesyłany do nowej akcji w kontrolerze `CalculatorController`;
  - wyniki obliczeń wyświetlane w widoku w postaci: `3 * 4 = 12`.

Grupowanie danych z formularza w obiektach JavaBean

- Obsługa parametrów żądania w pokazany sposób może być niewygodna, zwłaszcza jeśli parametrów tych jest wiele (np. rozbudowany formularz do edycji danych)

- Spring umożliwia automatyczne mapowanie parametrów żądania na złożone obiekty - najprościej można to osiągnąć, jeśli właściwości obiektu mają takie nazwy, jak parametry żądania HTTP.
- Przykład:
  - w formularzu HTML mamy pola `imie`, `nazwisko`, `wiek`;
  - tworzymy klasę `OsobaForm`, zawierającą właściwości o takich samych nazwach, zgodną ze specyfikacją JavaBean (czyli prywatne pola, publiczne metody `getXXX()`, `setXXX()`):

```
public class OsobaForm {

    private String imie;
    private String nazwisko;
    private Integer wiek;

    public String getImie() {
        return imie;
    }

    public void setImie(String imie) {
        this.imie = imie;
    }

    public String getNazwisko() {
        return nazwisko;
    }

    public void setNazwisko(String nazwisko) {
        this.nazwisko = nazwisko;
    }

    public Integer getWiek() {
        return wiek;
    }

    public void setWiek(Integer wiek) {
        this.wiek = wiek;
    }
}
```

  - w metodzie kontrolera zamiast trzech parametrów (`String imie`, `String nazwisko`, `Integer wiek`) podajemy jeden, typu `OsobaForm`:

```
@RequestMapping("/dodajOsobe")
public String dodajOsobe(Model model, OsobaForm osoba) {
```

```
            ...
}
```

  - Obiekt `OsobaForm osoba` będzie zawierał wszystkie dane z formularza.

Zadanie 3

- Przerób kalkulator - dodaj klasę `CalculatorForm`, zawierającą wszystkie dane z formularza (liczby, operator). Użyj jej do:
  - przekazania danych żądania do kontrolera,
  - przekazania danych do wyświetlenia w widoku.