



Spring  
**Boot**

# Dlaczego Spring Boot?

- uproszczenie konfiguracji aplikacji
- (nie)lubiany XML
- mikroserwisy
- integracja z bibliotekami Netflix
- zapotrzebowanie na deweloperów ; )



# Hello World

- szablon projektu: Spring Initializr
- dodaj zależność Web
- utwórz kontroler (minimum jedno URI)
- uruchom i przetestuj program



# Spring Initializr

**SPRING INITIALIZR** bootstrap your application now

Generate a Maven Project with Java and Spring Boot 2.0.1

### Project Metadata

Artifact coordinates

Group

Artifact

### Dependencies


Add Spring Boot Starters and dependencies to your application

Search for dependencies

Selected Dependencies

Web ×

Generate Project 



Spring  
Boot

# Utwórz kontroler

```
@RestController
public class HelloController {

    @GetMapping("/hello")
    public String greetName(@RequestParam(value="name", required=false) String name) {
        String greeting = "Hello, ";
        String defaultName = "World!";
        return name != null ? greeting + name : greeting + defaultName;
    }
}
```



# Przetestuj program - MockMvc

```
@RunWith(SpringRunner.class)
@WebMvcTest
public class HelloworldApplicationTests {

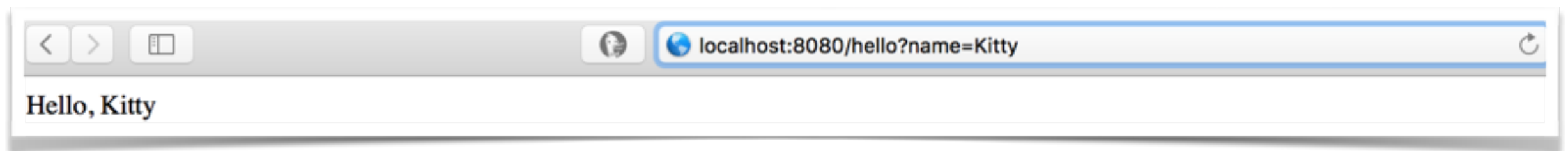
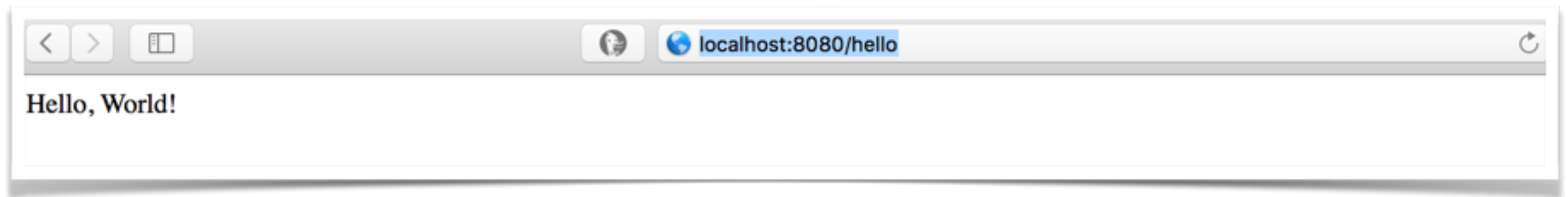
    @Autowired
    private MockMvc mockMvc;

    @Test
    public void shouldReturnDefaultMessage() throws Exception {
        this.mockMvc.perform(get(urlTemplate: "/hello"))
            .andExpect(status().isOk())
            .andExpect(content().string(containsString(substring: "Hello, World!")));
    }

    @Test
    public void shouldReturnHelloWithNamePassedAsParam() throws Exception {
        this.mockMvc.perform(get(urlTemplate: "/hello?name=Kitty"))
            .andExpect(status().isOk())
            .andExpect(content().string(containsString(substring: "Hello, Kitty")));
    }
}
```



# Przetestuj program - web



# Czym jest Spring Boot?

- bardzo łatwy start ze Springiem
  - konwencja ponad konfiguracją
  - przejrzyste 'wejście' do aplikacji
  - startowe POM'y
- Spring CLI
- umożliwia 'zmieszczenie' aplikacji w tweecie:
  - [https://twitter.com/rob\\_winch/status/364871658483351552?lang=en](https://twitter.com/rob_winch/status/364871658483351552?lang=en)





# Ćwiczenie - aplikacja calculator

- projekt startowy : `_2_calculator_start`
- rozwiązanie : `_2_calculator_final`



# Spring Boot - podstawy

- **I**nversion **O**f **C**ontrol
- **D**ependency **I**njection
- beans & collaborators
- @Primary, @Qualifier
- @SpringBootApplication
- konfiguracja
  - properties/yml
  - @ConfigurationProperties
  - profile
  - Java vs XML
- scope



# Spring Boot - podstawy

- **Inversion Of Control**
- Dependency Injection
- beans & collaborators
- @Primary, @Qualifier
- @SpringBootApplication
- konfiguracja
  - properties/yml
  - @ConfigurationProperties
  - profile
  - Java vs XML
- scope



# IoC

- kontrola nad obiektem lub częścią programu, zostaje przekazana do kontenera
  - rozdzielenie wykonania zadania od implementacji
  - lepsza segmentacja programu
  - ułatwienie zmiany implementacji
  - ułatwienie testowania
- IoC można zaimplementować na różne sposoby, jednym z nich jest wzorzec DI



# Spring Boot - podstawy

- Inversion Of Control
- **Dependency Injection**
- beans & collaborators
- @Primary, @Qualifier
- @SpringBootApplication
- konfiguracja
  - properties/yml
  - @ConfigurationProperties
  - profile
  - Java vs XML
- scope



# Dependency Injection

"Dependency Injection" is a 25-dollar term for a 5-cent concept. (...)

**Dependency injection means giving an object its instance variables.**

Really. That's it.

- James Shore



# Spring Boot - podstawy

- Inversion Of Control
- Dependency Injection
- **beans & collaborators**
- @Primary, @Qualifier
- @SpringBootApplication
- konfiguracja
  - properties/yml
  - @ConfigurationProperties
  - profile
  - Java vs XML
- scope



# Beans & collaborators

- **beans (ziarna)**
  - obiekty zarządzane przez Spring, tworzące aplikację
- **collaborators**
  - wstrzykiwane do obiektu zależności





# Wstrzykiwanie w Springu

- konstruktor
- setter
- property



# Wstrzykiwanie: konstruktor

```
@RestController
public class GreetController {

    private GreetService greetService;

    @Autowired
    public GreetController(GreetService greetService) {
        this.greetService = greetService;
    }

    // Użycie
}
```

Spring >= 4.3 && 1 konstruktor = @Autowired



Spring  
Boot

# Wstrzykiwanie: setter

```
@RestController
public class GreetController {

    private GreetService greetService;

    @Autowired
    public void setGreetService(GreetService greetService) {
        this.greetService = greetService;
    }

    @GetMapping("/greet")
    public String greet() { return greetService.getMsg(); }
}
```



# Wstrzykiwanie: property

```
@RestController
public class GreetController {

    @Autowired
    private GreetService greetService;

    // Użycie
}
```



# DI - wstrzykiwanie - ćwiczenie

- projekt startowy : `_3_constructor-setter-field-start`
- rozwiązanie : `_3_constructor-setter-field-final`



# Spring Boot - podstawy

- Inversion Of Control
- Dependency Injection
- beans & collaborators
- **@Primary, @Qualifier**
- @SpringBootApplication
- konfiguracja
  - properties/yml
  - @ConfigurationProperties
  - profile
  - Java vs XML
- scope



# @Primary

```
@Service  
@Primary  
public class GreetServiceFirstImplementation implements GreetService {
```

```
@Autowired  
private GreetService greetService;
```



# @Primary - ćwiczenie

Projekt startowy : \_4\_using-primary-start

Rozwiązanie : \_4\_using-primary-final



Spring  
Boot



# Dependency Injection - Spring

## @Qualifier

```
@Service("second")  
public class GreetServiceSecondImplementation implements GreetService
```

```
@Autowired  
@Qualifier("second")  
private GreetService greetService;
```



# @Qualifier - ćwiczenie

Projekt startowy : `_5_using-qualifier-start`

Rozwiązanie : `_5_using-qualifier-final`



# Spring Boot - podstawy

- Inversion Of Control
- Dependency Injection
- beans & collaborators
- @Primary, @Qualifier
- **@SpringBootApplication**
- konfiguracja
  - properties/yml
  - @ConfigurationProperties
  - profile
  - Java vs XML
- scope



# @SpringBootApplication

- @SpringBootApplication to skrót dla 3 poniższych adnotacji:
  - **@Configuration**
    - źródło definicji ziaren dla kontekstu aplikacji
  - **@EnableAutoConfiguration**
    - konwencja ponad konfiguracją
      - SpringBoot automatycznie załaduje niezbędne ziarna
        - w zależności od tego co znalazł na classpath
  - **@ComponentScan**
    - przeskanuj od bieżącego pakietu w dół
      - komponenty, konfiguracje i serwisy



# Spring Boot - podstawy

- Inversion Of Control
- Dependency Injection
- beans & collaborators
- @Primary, @Qualifier
- @SpringBootApplication
- **konfiguracja**
  - properties/yml
  - @ConfigurationProperties
  - profile
  - Java vs XML
- scope

# Zewnętrzna konfiguracja

- uruchamianie tej samej aplikacji w różnych środowiskach
- dokumentacja pokazuje 17 sposobów : )

[https://docs.spring.io/spring-boot/docs/current/reference/html/  
boot-features-external-config.html](https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-external-config.html)



# Spring Boot - podstawy

- Inversion Of Control
- Dependency Injection
- beans & collaborators
- @Primary, @Qualifier
- @SpringBootApplication
- **konfiguracja**
  - **properties/yml**
  - @ConfigurationProperties
  - profile
  - Java vs XML
- scope



# application.yml vs application.properties

pageController.msg: Hello from properties config

pageController.foo: foo

pageController.bar: bar

**vs**

pageController:

msg: Hello from YAML config

foo: foo

bar: bar





# **.properties & .yaml - ćwiczenie**

- projekt startowy : `_6_prop-and-yml-start`
- rozwiązanie : `_6_prop-and-yml-final`



# Spring Boot - podstawy

- Inversion Of Control
- Dependency Injection
- beans & collaborators
- @Primary, @Qualifier
- @SpringBootApplication
- konfiguracja
  - properties/yml
  - **@ConfigurationProperties**
  - profile
  - Java vs XML
- scope



# @ConfigurationProperties

- mapowanie props'ów do POJO
- wsparcie IDE
- @Valid problemy z YAML ; )
- użycie
  - komponent: @ConfigurationProperties,
  - dodanie zależności do maven'a ( IDE podpowiada),
  - setter'y dla ustawianych pól



# @ConfigurationProperties - ćwiczenie

- projekt startowy : `_7_configuration-properties-start`
- rozwiązanie : `_7_configuration-properties-final`



# Spring Boot - podstawy

- Inversion Of Control
- Dependency Injection
- beans & collaborators
- @Primary, @Qualifier
- @SpringBootApplication
- **konfiguracja**
  - properties/yml
  - @ConfigurationProperties
  - **profile**
    - Java vs XML
- scope



# Profile

- kontrola, czy dane ziarno jest ładowane do kontekstu aplikacji
- użyteczne np. dla: lokalna baza vs produkcyjna
- `@Profile("development")`
  - `spring.profiles.active=development`
  - `-Dspring.profiles.active=development`



# Profile - ćwiczenie

- projekt startowy : `_8_profiles-start`
- rozwiązanie : `_8_profiles-final`



# Spring Boot - podstawy

- Inversion Of Control
- Dependency Injection
- beans & collaborators
- @Primary, @Qualifier
- @SpringBootApplication
- **konfiguracja**
  - properties/yml
  - @ConfigurationProperties
  - **profile**
  - Java vs XML
- scope





# Dependency Injection - konfiguracja w Springu

- Spring pozwala konfigurować ziarna na różne sposoby
  - XML
  - Java Configuration
  - @dnotacje
  - Groovy Configuration



# Dependency Injection - konfiguracja w Springu

- nie zajmujemy się Groovy'm w tym kursie
- generalnie możemy mówić o podziale na konfigurację XML i Java
- adnotacje to XML +



# Dependency Injection - konfiguracja w Springu

```
@Configuration
@ComponentScan(basePackages = {"pl.altkom.di.controller"})
public class JavaConfig {
    @Bean
    @Profile("java")
    public GreetService greetService() {
        return new GreetServiceFirstImplementation( msg: " Bean configured in Java");
    }
}
```

```
<beans profile="xml">
    <bean id="secondImpl" class="pl.altkom.di.service.GreetServiceSecondImplementation">
        <constructor-arg value="${service.second.implementation.msg}"/>
    </bean>
    <context:component-scan base-package="pl.altkom.di.controller"/>
    <context:property-placeholder location="classpath:service.properties" />
</beans>
```



# Dependency Injection - ćwiczenie

- projekt startowy : `_9_configuration-start`
- rozwiązanie : `_9_configuration-final`



# Spring Boot - podstawy

- Inversion Of Control
- Dependency Injection
- beans & collaborators
- @Primary, @Qualifier
- @SpringBootApplication
- **konfiguracja**
  - properties/yml
  - @ConfigurationProperties
  - **profile**
  - Java vs XML
- **scope**



# Bean's scope

- **@Scope("<wybrany\_scope>")**
- singleton (default)
- prototype
- request
- session
- global session



# Bean's scope -ćwiczenie

- projekt startowy : \_10-scope-start
- rozwiązanie : \_10-scope-final



# Ćwiczenie końcowe

- Utrwalenie wiedzy:
  - wstrzykiwanie zależności
  - application.properties / yml
  - @SpringBootApplication
  - @Profiles





# TODO



- utwórz serwis REST:
  - GET - localhost:8080/foo
  - GET - localhost:8080/bar
  - GET - localhost:8080/x
- osobne pliki (yml lub properties), dla profili:
  - dev
  - prod
  - default
- HomeController korzystający z 2 implementacji interfejsu MessageService
- implementacje zwracają wartości z pliku: foo, bar i x
- jedna z implementacji używa @Value, druga @ConfigurationProperties
- o tym, która implementacja będzie wstrzykiwana do kontrolera decyduje profil



# I szkielet projektu



- utwórz projekt spring-boot (Web) di-exercise
- utwórz pakiet controller z klasą HomeController
- utwórz pakiet service z interfejsem MessageService
  - String foo(), String bar(), int x()
- implementory
  - (domyślne implementacje z IDE)
  - @Service
  - MessageServiceDevImpl
  - MessageServiceProdImpl



# Il property



- w resources dodaj trzy pliki (rozszerzenie properties lub yml)
  - application
  - application-dev
  - application-prod
- w każdym z plików dodaj wartości
  - `example.value.foo="jakis string"`
  - `example.value.bar = "jakis inny string"`
  - charakterystyczne dla pliku
- tylko w application dodaj `example.value.x=999` i `spring.profiles.active=prod`
- w implementorach `MessageService` dodaj pola `String foo`, `String bar`, `int x`.
- w `MessageServiceDevImpl` ustaw pola używając `@Value`.
- w `MessageServiceProdImpl` ustaw pola używając `@ConfigurationProperties`
  - (potrzebna zależność w mavenie)
- zaimplementuj metody implementorów
  - zwracają wartości pól



# III użycie serwisu



- wstrzyknij serwis
  - pole, setter lub konstruktor
- w kontrolerze zaimplementuj 3 metody
  - zwracają wartości propriety'ów
    - **/foo**
    - **/bar**
    - **/x**





# IV profile



- zdefiniuj statyczne, finalne pola dla profili
  - dev
  - prod
- użyj tych pól jako wartości w @Profile dla implementorów



# V test



- przetestuj działanie aplikacji w przeglądarce
- opcjonalnie napisz testy



# Opcjonalnie

- zmodyfikuj serwis
  - inkrementacja dla kolejnego requestu w sesji

