IŠ PRAEITOS PASKAITOS

- jeigu java serveryje įkėlus projektą rodomas tuščias baltas langas, bent ant kito port'o aplikacija veikia
  - pasiūlymas užkomentuoti index.js service worker import'ą bei paskutine eilutę, kur service worker atlieka unregister
  - aplikacijos kūrimo etape užsikešuoja ir neteisingai iš cache nuskaito

IŠ PRAEITOS PASKAITOS

- jeigu neteisingai suderinote aplinką, tuomet dar nors ir java -version rodo 8 java versiją, tačiau:

```
$ mvn -version
> Apache Maven 3.5.2
> Java version: 10.0.2, vendor: Oracle Corporation
> Java home: /usr/lib/jvm/java-10-oracle
```

- turite pakeisti java ir maven'ui. Ubuntu sistemoje versiją pakeisti galima į .bashrc isidėjus:

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre
```

IŠ PRAEITOS PASKAITOS

- praeitą kartą generavote keletą aplikacijų
- toliau dirbsite su dviem projektais
  - su įprastu quickstart maven projektu
  - vis atnaujinsite Spring Boot projektą
- iš pradžių spring prijungsite ir atliksite užduotis iš quickstart šablono pasidarytoje aplikacijoje/projekte

AKADEMIJA.IT

INFOBALT IR TECH CITY

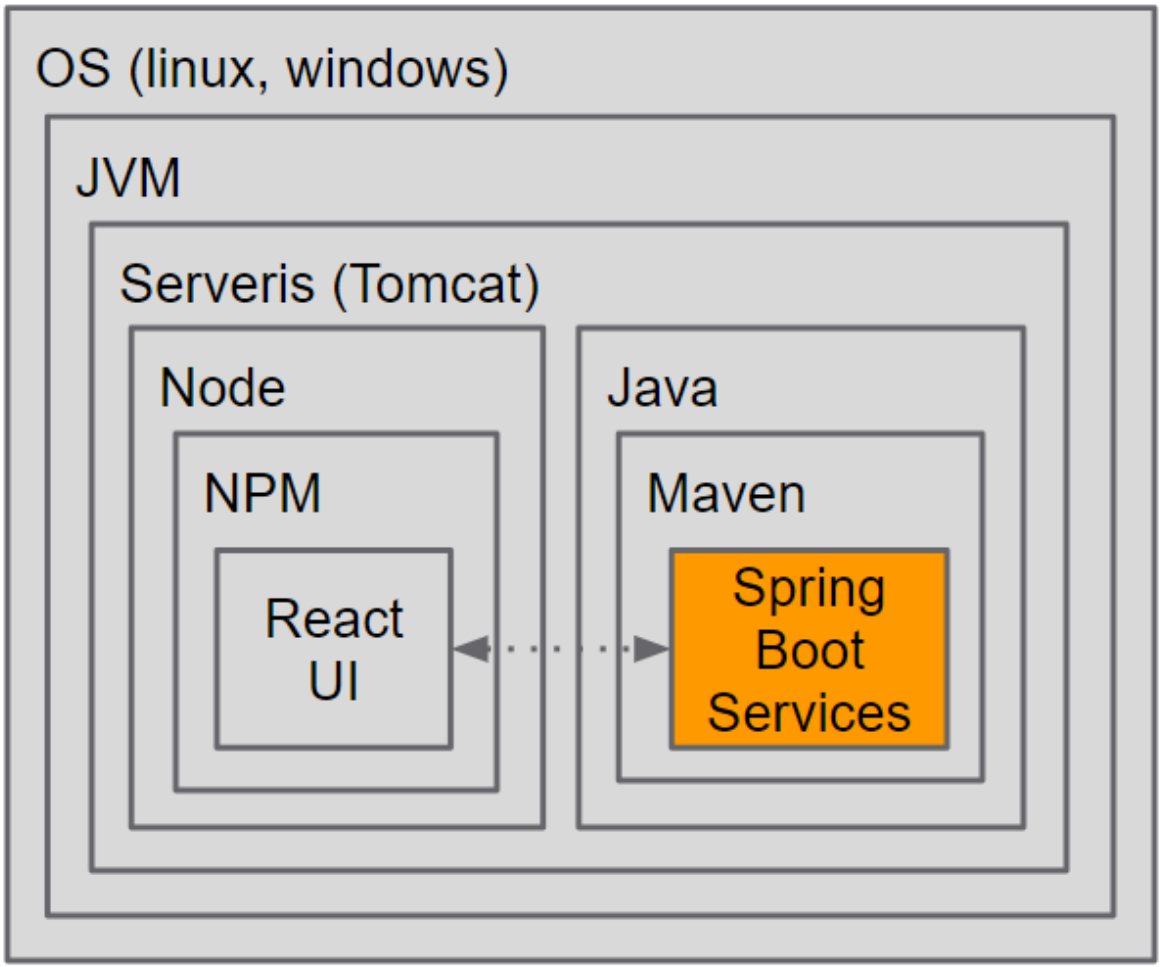
KAS YRA SPRING. KONFIGŪRACIJA PER XML. MAVEN MIGRACIJA

Andrius Stašauskas

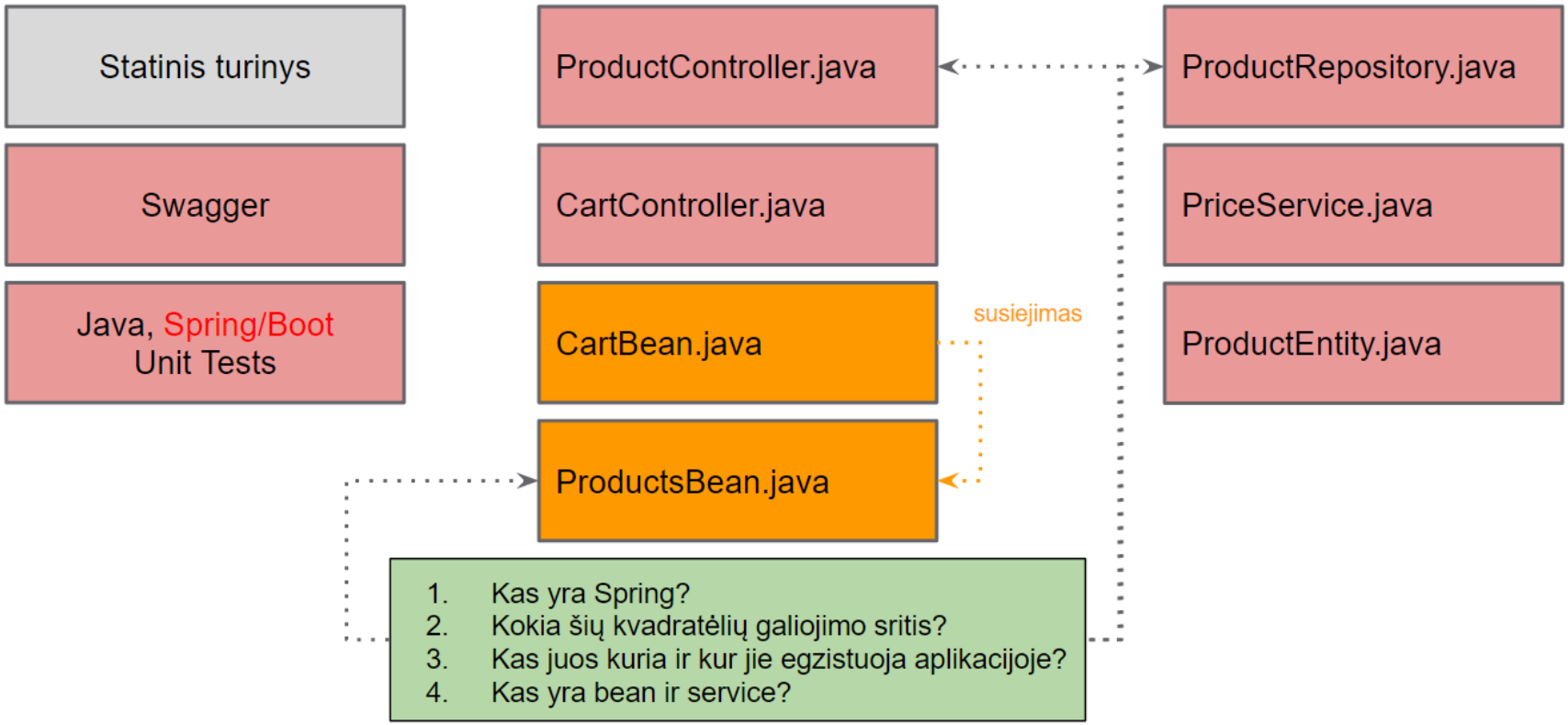
andrius@stasauskas.lt

http://stasauskas.lt/itpro2018/

KĄ JAU MOKAME IR KO DAR NE



KĄ JAU MOKAME IR KO DAR NE



TURINYS

- Kas yra Spring
  - IoC
  - DI
  - Container ir bean
- Spring konfigūracija:
  - XML
  - priklausomybės
- Maven projekto migracija
  - Spring Boot migracija

KAS YRA SPRING?

- Vienas populiariausių atviro kodo programinės įrangos kūrimo karkasų Java platformai
- Spring yra lengvasvoris ir pagrindiniai karkaso komponentai užima apie 2MB
- Pagrindinės Spring karkaso savybės gali būti naudojamos kuriant bet kokią Javą aplikaciją, tačiau papildomi karkaso moduliai ir išplėtimai leidžia kurti žiniatinklio aplikacijas naudojant Java EE platformą

SPRING TIKSLAI

- Supaprastinti Java EE PĮ kūrimo procesą.
- Spręsti problemas, kurių nepadengė Java EE.
- Integracija su populiariausiomis technologijomis.
- Pateikti modulinę architektūrą
  - galima pasirinkti ką naudoti, o ko ne

MAVEN SPRING PRIKLAUSOMYBIŲ PAVYZDŽIAI

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>4.3.13.RELEASE</version>
</dependency>
```

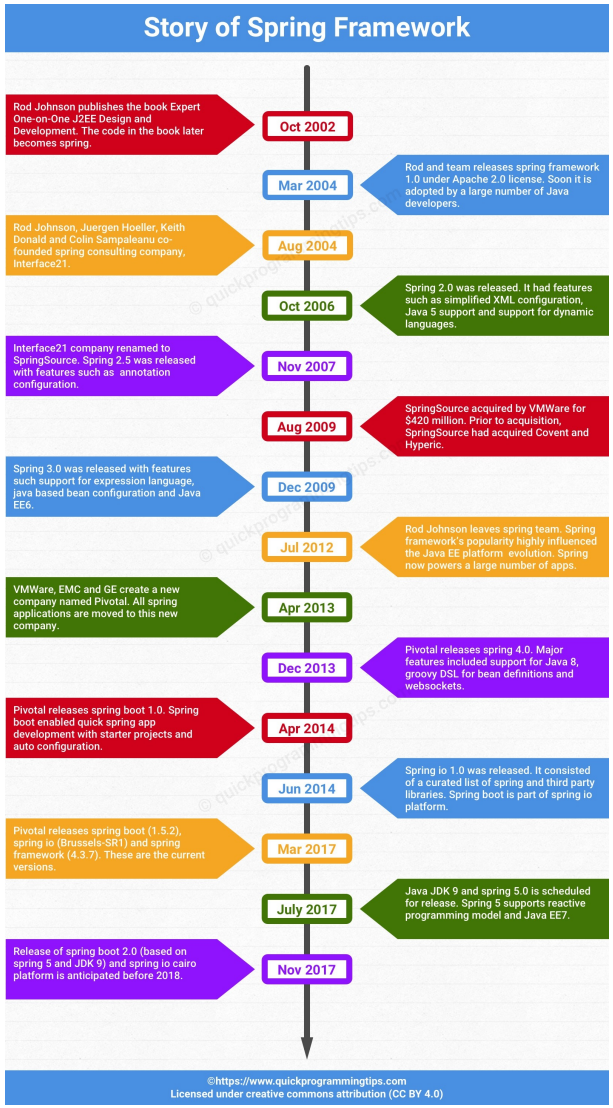
- Stabilios Spring Context versijos priklausomybė

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <version>1.5.9.RELEASE</version>
</dependency>
```

- Spring Boot

SPRING RAIDA

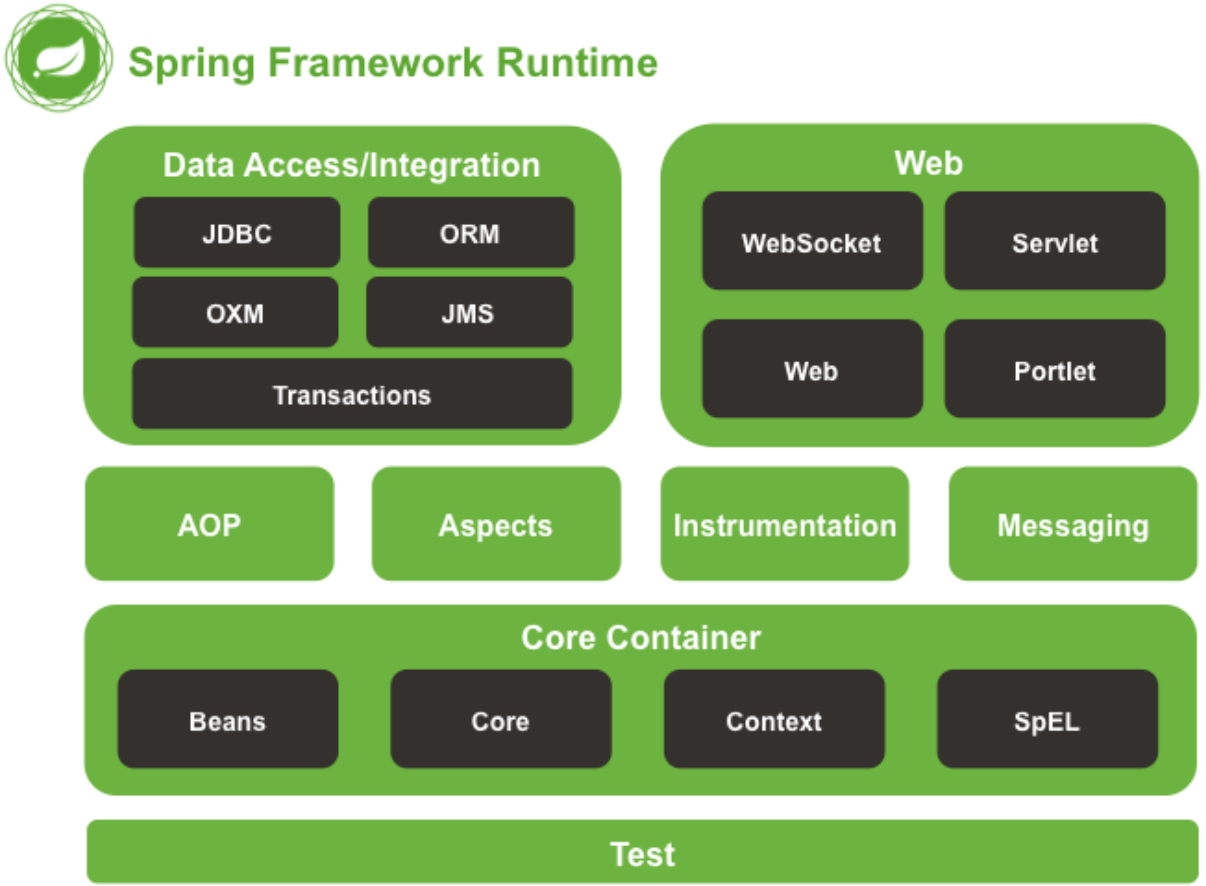
- Spring 1.0 (2001) - priklausomybių injekcija (DI - dependency injection), AOP, žiniatinklio karkasas.
- Spring 2.0 (2006) - Išplečiama konfigūracija (angl. extensible config), bean galiojimo sritis, dinaminių kalbų palaikymas (pvz. Groovy, Ruby), nauja žymių (angl. tag) biblioteka.
- Spring 2.5 (2007) - valdymas anotacijomis (angl. annotation-driven), automatinis bean suradimas (angl. discovery), naujas žiniatinklio karkasas, JUnit 4 integracija.
- Spring 3.0 (2009) - REST, SpEL, deklaratyvi validacija, ETag palaikymas, konfigūracija Java pagrindu (angl. Java-based).
- Spring 4.0 (2013) - Java 8, groovy DSL for beans and websockets
- Spring Boot 1.0 (2014) - quick Spring app development
  - Spring Boot 1.5.2 (2017)
  - Spring 4.3.7 (2017) - Java 6-8
- Spring 5.0 (2017-2018) - Java 8-10, reactive/functional programming, Java EE7
  - Spring Boot 2.0.6 (2018-10-16) Min Tomcat is 8.5.x
  - Spring 5.0.10 (2018-10-15)
- Spring 5.1 (2018-09-21) - Java 8-12 - Java 18.3 (10; no support), 18.9 (11; long support)
  - Spring Boot 2.1 (2018-10-30)


Story of Spring Framework
Rod Johnson publishes the book Expert One-on-One J2EE Design and Development. The code in the book later becomes spring.
Oct 2002
Mar 2004
Rod and team releases spring framework 1.0 under Apache 2.0 license. Soon it is adopted by a large number of Java developers.
Rod Johnson, Juergen Hoeller, Keith Donald and Colin Sampaleanu co-founded spring consulting company, Interface21.
Aug 2004
Oct 2006
Spring 2.0 was released. It had features such as simplified XML configuration, Java 5 support and support for dynamic languages.
Interface21 company renamed to SpringSource. Spring 2.5 was released with features such as annotation configuration.
Nov 2007
Aug 2009
SpringSource acquired by VMWare for $420 million. Prior to acquisition, SpringSource had acquired Covent and Hyperic.
Spring 3.0 was released with features such support for expression language, java based bean configuration and Java EE6.
Dec 2009
Jul 2012
Rod Johnson leaves spring team. Spring framework's popularity highly influenced the Java EE platform evolution. Spring now powers a large number of apps.
VMWare, EMC and GE create a new company named Pivotal. All spring applications are moved to this new company.
Apr 2013
Dec 2013
Pivotal releases spring 4.0. Major features included support for Java 8, groovy DSL for bean definitions and websockets.
Pivotal releases spring boot 1.0. Spring boot enabled quick spring app development with starter projects and auto configuration.
Apr 2014
Jun 2014
Spring io 1.0 was released. It consisted of a curated list of spring and third party libraries. Spring boot is part of spring io platform.
Pivotal releases spring boot (1.5.2), spring io (Brussels-SR1) and spring framework (4.3.7). These are the current versions.
Mar 2017
July 2017
Java JDK 9 and spring 5.0 is scheduled for release. Spring 5 supports reactive programming model and Java EE7.
Release of spring boot 2.0 (based on spring 5 and JDK 9) and spring io cairo platform is anticipated before 2018.
Nov 2017
©https://www.quickprogrammingtips.com
Licensed under creative commons attribution (CC BY 4.0)

PAPILDOMI SPRING PROJEKTAI

- Spring Web Flow
- Spring-WS
- Spring Boot
- Spring Web
- Spring Security
- Spring Batch, Spring Integration, Spring LDAP, Spring IDE / STS
- Spring Rich Client, Spring BeanDoc, BlazeDS Integration, Spring-DM, dmServer, Bundlor, tcServer

SPRING ARCHITEKTŪRA



ŽINIATINKLIS

- **Web** pagrindinis žiniatinklio funkcionalumas. Bylų nusiuntimas ir IoC konteinerio inicializavimas naudojant servlet listener ir web aplikacijų kontekstas.
- **Web-Servlet** modulis suteikia Spring MVC (modelis, vaizdas, kontroleris) implementaciją, skirtą žiniatinklio aplikacijoms.
- **WebSocket** modulis realizuoja integraciją su WebSocket ir SockJS, taip pat STOMP
- **Web-Portlet** modulis suteikia MVC implementaciją, skirtą portalo komponentams (angl. portlet).

DUOMENŲ PRIEIGA / INTEGRACIJA

- **JDBC** (Java Database Connectivity) šablonų abstrakcijos sluoksnis, skirtą pakeisti tiesioginį JDBC naudojimą.
- **ORM** (Object Relational Mapping) integracija su populiariausiais objektų į realiacijnę DB susiejimo karkasais (pvz. JPA, Hibernate, JDO, iBatis).
- **OXM** Object/XML susiejimo abstrakcijos sluoksnis, skirts JAXB/Castor/XMLBeans/JiBX/XStream palaikymui.
- **JMS** (Java Messaging Service) - siųsti ir gauti žinutes.
- Tranzakcijų modulis realizuoja deklaratyvų ir programinį tranzakcijų valdymą.

TESTAVIMAS

- Testavimo modulis turi integraciją su testavimo karkasais:
  - JUnit,
  - TestNG.
- Suteikia galimybę užkrauti testavimui skirtą aplikacijos kontekstą (`ApplicationContext`).
- Turi testavimui naudingus netikrus (angl. mock) objektus.

KITI MODULIAI

- **AOP** modulis suteikia aspektais orientuoto programavimo implementaciją, skirtą apibrėžti metodų perėmėjus (angl. method-interceptors), įterpimo taškus (angl. pointcuts).
- **Aspects** modulis realizuoja integraciją su AspectJ aspektais paremto programavimo karkasu.
- **Instrumentation** modulis suteikia klasių instrumentavimo ir klasių užkrovimo palaikymą specifiniams aplikacijų serveriams.
- **Messaging** modulis realizuoja abstrakcijas žinučių pagrindu veikiančioms aplikacijoms.

PAGRINDINIS KONTEINERIS

- **Core** modulis realizuoja kertinį Spring karkaso funkcionalumą:
  - kontrolės inversija (IoC - Inversion of Control),
  - priklausomybių injekcija (DI - Dependency Injection).
- **Bean** modulis suteikia `BeanFactory` fabriko šablono (angl Factory Pattern) implementaciją, skirtą Java objektų sukūrimui.

PAGRINDINIS KONTEINERIS

- **Context** modulis, naudodamas Core ir Bean modulius, realizuoja objektų aprašymo ir konfigūravimo funkcionalumą. `ApplicationContext` sąsaja yra centrinis Context modulio elementas.
- **SpEL** Spring Expression Language (išraiškų kalbų) modulis suteikia galingą užklausų ir ojektų grafo manipuliavimo, vykdymo metu, funkcionalumą.

KAS YRA IOC?

- Įprastoje programoje objektų gyvavimo ciklą kontroliuoja parašytas programinis kodas.
- IoC paremtoje sistemoje objektų gyvavimo ciklą valdo programinis konteineris.
- Jums reikia sukurti tik patį pirminį objektą `BeanFactory`, o visus kitus objektus, pagal poreikį, sukurs konteineris.

KAIP REALIZUOJAMAS IOC?

- Konteineris, valdydamas objektų gyvavimo ciklą, taip pat turi valdyti ir ryšius bei priklausomybes tarp objektų. Šiam tikslui naudojamos dvi strategijos:
  - Priklausomybės paieška (angl. Dependency lookup) - komponentas kitus jam reikalingus komponentus susiranda pats.
  - Priklausomybės injekcija (angl Dependency injection) - konteineris perduoda reikalingus komponentus per:
    - konstruktorių,
    - `set*` metodus (JavaBeans properties).
- Spring IoC naudoja priklausomybės injekcijos strategiją.

INJEKCIJA PER KONSTRUKORIŲ AR SET* METODUS?

- Injekcija per konstruktorių paprastai naudojama komponento parametrams, kurie yra būtini jo darbui.
- Injekcija per `set*` metodus paprastai naudojama, kai komponentas turi parametrų reikšmes pagal nutylėjimą arba norima leisti reikšmes perrašyti konteineriui.
- Praktikoje dažniausiai naudojama injekcija per `set*` metodus.

SPRING IOC KONTEINERIAI

- `BeanFactory` konteineris - paprasčiausias konteineris, realizuojantis bazinį priklausomybių injekcijos palaikymą ir apibrežiamas `org.springframework.beans.factory.BeanFactory` interfeisu. `BeanFactory` ir kiti susiję interfeisai (pvz. `BeanFactoryAware, InitializingBean, DisposableBean`) yra vis dar laikomi Spring karkase dėl atgalinio suderinanumo su daugeliu trečių šalių karkasų.

SPRING IOC KONTEINERIAI

- `ApplicationContext` konteineris - dažniausiai naudojamas konteineris, apibrėžiamas `org.springframework.context.ApplicationContext` interfeisu. Pagrindinės savybės:
  - I18N palaikymas.
  - Įvykių skleidimas (angl. Event Propagation): `ContextRefreshedEvent, ContextStartedEvent, ContextStoppedEvent, ContextClosedEvent, RequestHandledEvent.`
  - Resursų užkrovimas.

APPLICATIONCONTEXT IMPLEMENTACIJOS

- `FileSystemXmlApplicationContext` - šis konteineris nuskaito bean aprašymo XML bylas naudodamas į konstruktorių perduotą pilną bylos kelią.
- `ClassPathXmlApplicationContext` - šis konteineris nuskaito bean aprašymo XML bylas iš aplikacijos CLASSPATH.
- `WebXmlApplicationContext` - šis konteineris nuskaito bean aprašymo XML bylas iš žiniatinklio aplikacijos.

SPING IOC PAVYZDYS - BEAN

```
package com.tutorialspoint;
public class HelloWorld {
    private String message;
    public void setMessage(String message){
        this.message = message;
    }
    public void getMessage(){
        System.out.println("Your Message : " + message);
    }
}
```

SPING IOC PAVYZDYS - BEANS.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"
    <bean id="helloWorld" class="com.tutorialspoint.HelloWorld">
        <property name="message" value="Hello World!"/>
    </bean>
</beans>
```

SPING IOC PAVYZDYS - BEANFACTORY

```
package com.tutorialspoint;
import org.springframework.beans.factory.InitializingBean;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
public class MainApp {
    public static void main(String[] args) {
        XmlBeanFactory factory = new XmlBeanFactory
            (new ClassPathResource("Beans.xml"));
        HelloWorld obj = (HelloWorld) factory.getBean("helloWorld");
        obj.getMessage();
    }
}
```

SPING IOC PAVYZDYS - APPLICATIONCONTEXT

- vieno iš galimų kontekstų FileSystemXmlApplicationContext pavyzdys

```
package com.tutorialspoint;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
public class MainApp {
  public static void main(String[] args) {
      ApplicationContext context = new FileSystemXmlApplicationContext
          ("C:/Users/ZARA/workspace/HelloSpring/src/Beans.xml");
      HelloWorld obj = (HelloWorld) context.getBean("helloWorld");
      obj.getMessage();
  }
}
```

UŽDUOTIS 1 - SUKURTI PROJEKTO STRUKTŪRĄ

- Pagrindiniame projektų kataloge sukurti naują `FirstSpringProject` panaudojant Maven archetipą:

```
$ mvn archetype:generate -DgroupId=lt.itmokymai.spring \
> -DartifactId=FirstSpringProject \
> -DarchetypeArtifactId=maven-archetype-quickstart \
> -DinteractiveMode=false
```

```
# Pasirinkti projekto katalogą:
$ cd FirstSpringProject/
# Sukurti src/main/resources katalogą:
$ mkdir src/main/resources
```

- Sukurti Eclipse projekto konfigūraciją:

```
$ mvn eclipse:eclipse
```

UŽDUOTIS 1 - POM.XML PRIDĖTI SPRING PRIKLAUSOMYBĘ

- Taigi Spring Boot archetipą kol kas palikome nuošalyje ir pasidarėme naują quickstart paremtą `FirstSpringProject`
- Pridėkime į jo `pom.xml` Spring

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>5.0.10.RELEASE</version>
</dependency>
```

UŽDUOTIS 1 - POM.XML PRIDĖTI EXEC:JAVA PAPILDINĮ

```
<build><plugins><plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>exec-maven-plugin</artifactId>
  <version>1.3.2</version>
  <executions>
    <execution>
    <goals>
      <goal>java</goal>
    </goals>
    </execution>
  </executions>
  <configuration>
    <mainClass>lt.itmokymai.spring.App</mainClass>
  </configuration>
  </plugin></plugins></build>
```

UŽDUOTIS 1 - PRIDĖTI APLIKACIJŲ KONTEKSTO BYLĄ

- Sukurti pradinę src/main/resources/ **`application-context.xml`** bylą:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/spring-beans.xsd">

  <!-- Bean definitions goes there -->

</beans>
```

UŽDUOTIS 1 - SUKURTI BEAN SERVICEA

- Sukurti `ServiceA` klasę `lt.itmokymai.spring` pakete:

```
package lt.itmokymai.spring;
public class ServiceA {
  private String message;
  public String getResult() { return getMessage(); }
  public String getMessage() { return message; }
  public void setMessage(String message) {
      this.message = message;
  }
}
```

- Užregistruoti `ServiceA` bean `application-context.xml` byloje:

```
<bean id="serviceABean" class="lt.itmokymai.spring.ServiceA">
  <property name="message" value="ServiceA message" />
</bean>
```

UŽDUOTIS 1 - PAKEISTI LT.ITMOKYMAI.SPRING.APP KLASĘ

- `App.main()` metode sukurti IoC konteinerį.
- iš konteinerio gauti `ServiceA` bean.
- atspausdinti `ServiceA.getResult()` rezultatą

```
package lt.itmokymai.spring;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class App {
  public static void main( String[] args ) {
    ApplicationContext context = new ClassPathXmlApplicationContext(
      "application-context.xml");
    ServiceA serviceA = (ServiceA) context.getBean("serviceABean");
    System.out.println(serviceA.getResult());
    ((ConfigurableApplicationContext) context).close();
  }
}
```

UŽDUOTIS 1 - ĮVYKDYTI LT.ITMOKYMAI.SPRING.APP KLASĘ

- Pagaminti projekto darinį:

```
$ mvn clean package
```

- Įvykdyti `lt.itmokymai.spring.App` klasę naudojant `exec:java` papildinį:

```
$ mvn exec:java
```

- Įvykdyti `lt.itmokymai.spring.App` klasę Eclipse priemonėmis.

SPRING BEAN (PUPOS)

AKADEMIJA.IT
INFOBALT IR TECH CITY

KAS YRA SPRING BEAN?

- Objektai, kurie sudaro aplikacijos pagrindą ir kurių gyvavimo ciklas (inicializavimas, surinkimas ir pan.) yra valdomas Spring IoC konteinerio.
- Paprastai tai Java klasė, realizuojanti tam tikrą interfeisą ir JavaBean specifikaciją.
- Bean sukūrimui konteineris naudoja konfigūracijos metaduomenis:
  - XML paremta konfigūracija,
  - anotacijomis paremta konfigūracija,
  - Java paremta konfigūracija.

SPRING BEAN APRAŠAS

- `class` - privalomas atributas nurodantis bean sukūrimui naudojama Java klasę.
- `name` - šis atributas nurodo unikalų bean identifikatorių. XML konfigūracijoje galima naudoti `id` ir / arba `name` atributus bean identifikatorių nurodymui.
- `scope` - šis atributas nurodo bean galiojimo sritį.
- `constructor-arg` - naudojamas priklausomybių injekcijai į bean konstruktorių.
- `properties` - naudojamas priklausomybių injekcijai į `set*` metodus.

SPRING BEAN APRAŠAS

- `autowire` - naudojamas automatinei priklausomybių injekcijai.
- `lazy-init` - nurodo konteineriui sukurti bean pagal poreikį, o ne konteinerio paleidimo metu.
- `init-method` - grįžtamasis iškvietimas po to kai konteineris priskyrė visas privalomas bean savybes.
- `destroy-method` - grįžtamasis iškvietimas po to, kai bean valdantis konteineris yra sunaikimamas.

SPRING BEAN APRAŠYMO PAVYZDYS 1

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"
    <!-- A simple bean definition -->
    <bean id="..." class="...">
        <!-- collaborators and configuration for this bean go here --
    </bean>
    <!-- A bean definition with lazy init set on -->
    <bean id="..." class="..." lazy-init="true">
        <!-- collaborators and configuration for this bean go here --
    </bean>
</beans>
```

SPRING BEAN APRAŠYMO PAVYZDYS 2

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"
    <!-- A bean definition with initialization method -->
    <bean id="..." class="..." init-method="...">
        <!-- collaborators and configuration for this bean go here --
    </bean>
    <!-- A bean definition with destruction method -->
    <bean id="..." class="..." destroy-method="...">
        <!-- collaborators and configuration for this bean go here --
    </bean>
    <!-- more bean definitions go here -->
</beans>
```

SPRING BEAN GALIOJIMO SRITIS (SCOPE)

- `singleton` - konteineris sukurs tik vieną bean esybę (naudojama pagal nutylėjimą).
- `prototype` - IoC kiekvieną kartą kurs naują bean esybę.
- `request` - bean galiojimo sritis yra HTTP užklausa. Galimas tik su žiniatinklio `ApplicationContext`.
- `session` - bean galiojimo sritis yra HTTP sesija. Galimas tik su žiniatinklio `ApplicationContext`.
- `application` - bean galiojimo sritis yra aplikacija. Galimas tik su žiniatinklio `ApplicationContext`.
- `websocket` - bean galiojimo sritis yra websocket. Galimas tik su žiniatinklio `ApplicationContext`.

SPRING BEAN GALIOJIMO SRITIS

- Pavyzdys:

```
<!-- A bean definition with singleton scope -->
<bean id="..." class="..." scope="singleton">
    <!-- collaborators and configuration for this bean go here -->
</bean>
```

SPRING BEAN GALIOJIMO SRITIS

- Norint panaduoti mažesnės galiojimo srities bean didesnėje, reikia naudoti aop:proxy:

```
<!-- HTTP sesijos bean -->
<bean id="userPreferences"
 class="com.something.UserPreferences" scope="session">
    <!-- nurodo IoC is userPreferences padaryti proxy bean -->
    <aop:scoped-proxy/>
</bean>

<!-- singleton bean kuris gauna proxy bean [userPreferences] -->
<bean id="userService" class="com.something.SimpleUserService">
    <property name="userPreferences" ref="userPreferences"/>
</bean>
```

SPRING BEAN GYVAVIMO CIKLAS - INICIALIZAVIMAS

- Spring bean inicializavimo grįžtamasis iškvietimas naudojant `InitializingBean` interfeisą:

```
import org.springframework.beans.factory.InitializingBean;
public class MyBean implements InitializingBean {
 public void afterPropertiesSet() { /* do some initialization work */
}
```

- arba `init-method` attributą XML Spring bean konfigūracijoje:

```
<bean id="myBean" class="pvz.MyBean" init-method="init"/>
public class MyBean {
 public void init() { /* do some initialization work */ }
}
```

SPRING BEAN GYVAVIMO CIKLAS - SUNAIKINIMAS

- Spring bean sunaikinimo grįžtamasis iškvietimas naudojant `DisposableBean` interfeisą:

```
import org.springframework.beans.factory.DisposableBean;
public class MyBean implements DisposableBean {
 public void destroy() { /* do some destruction work */ }
}
```

- arba `destroy-method` attributą XML Spring bean konfigūracijoje:

```
<bean id="myBean" class="pvz.MyBean" destroy-method="destroy"/>
public class MyBean {
 public void destroy() { /* do some destruction work */ }
}
```

SPRING BEAN GYVAVIMO CIKLAS - DEFAULT

- Spring bean pagal nutylėjimą inicializavimo ir sunaikinimo grįžtamieji iškvietimai gali būti nurodomi XML konfigūracijos `default-init-method` and `default-destroy-method` atributais:

```
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"
 default-init-method="init"  default-destroy-method="destroy">
 <bean id="..." class="...">
 <!-- collaborators and configuration for this bean go here -->
 </bean>
</beans>
```

SPRING BEAN BAIGIAMOJI DOROKLĖ

- Spring bean baigiamosios doroklės (angl. post processor) grįžtamasis iškvietimas yra nurodomas `BeanPostProcessor` interfeisu.
- BeanPostProcessor intefeisas deklaruoja `postProcessBeforeInitialization` or `postProcessAfterInitialization` metodus, skirtus bean inicializavimo papildymui.
- Galima užregistruoti daugiau nei vieną `BeanPostProcessor`.

SPRING BEAN BAIGIAMOJI DOROKLĖ

- `BeanPostProcessor` implementuoja the `Ordered` interfeisą, tam kad galima būtų pakeisti baigiamųjų doroklių iškvietimo seką.
- `ApplicationContext` automatiškai randa visus bean kurie implementuoja `BeanPostProcessor` intefeisą ir užregistruoja konteineryje.
- Konteineris sukurtą bean objektą perduoda `BeanPostProcessor`tolimesniam inicializavimui.

BAIGIAMOJI DOROKLĖ PAVYZDYS - BEAN

```
package lt.itmokymai.spring;
import org.springframework.beans.factory.config.BeanPostProcessor;
import org.springframework.beans.BeansException;
public class InitHelloWorld implements BeanPostProcessor {
  public Object postProcessBeforeInitialization(Object bean,
      String beanName) throws BeansException {
    System.out.println("BeforeInitialization : " + beanName);
    return bean; // you can return any other object as well
  }
  public Object postProcessAfterInitialization(Object bean,
      String beanName) throws BeansException {
    System.out.println("AfterInitialization : " + beanName);
  return bean; // you can return any other object as well
  }
}
```

BAIGIAMOJI DOROKLĖ PAVYZDYS - XML

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"
  <bean id="helloWorld" class="com.tutorialspoint.HelloWorld"
      init-method="init" destroy-method="destroy">
    <property name="message" value="Hello World!"/>
  </bean>
  <bean class="com.tutorialspoint.InitHelloWorld" />
</beans>
```

UŽDUOTIS 2

- prieš bean kuriantis, išvesti bean vardą
- bean susikūrus, pranešti apie tai, kad bean sukurtas
  - pranešime turi figūruoti bean vardas
- po bean sunaikinimo išvesti tekstą, kad bean susinaikino
  - pranešime turi figūruoti bean vardas

SPRING PRIKLAUSOMYBĖS

AKADEMIJA.IT
INFOBALT IR TECH CITY

SPRING BEAN APRAŠYMO PAVELDĖJIMAS

- Bean gali turėti įvairios konfigūracijos: konstuktorių argumentai, atributai, inicializavimo metodai ir kt.
- Vaiko (angl. child) bean paveldi konfigūracijos duomenis iš tėvo (angl. parent) bean aprašymo ir gali perkrauti reikiamas reikšmes ir / arba pridėti naujas.
- Bean aprašymo paveldėjimas yra nesusijęs su Java klasių paveldėjimu ir bean aprašymą galima naudoti, kaip šabloną kitų bean aprašymui.
- XML paremtoje konfigūracijoje vaiko bean indikuoja `parent` atributas, nurodantis tėvo bean identifikatorių (bean `name` arba `id` atributo reikšmė).

BEAN APRAŠYMO PAVELDĖJIMO PAVYZDYS

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"
  <bean id="helloWorld" class="com.tutorialspoint.HelloWorld">
    <property name="message1" value="Hello World!"/>
    <property name="message2" value="Hello Second World!"/> <!-- inhe
  </bean>
  <bean id="helloIndia" class="com.tutorialspoint.HelloIndia"
      parent="helloWorld">
    <property name="message1" value="Hello India!"/> <!-- override --
    <property name="message3" value="Namaste India!"/> <!-- add -->
  </bean>
</beans>
```

SPRING BEAN PRIKLAUSOMYBIŲ INJEKCIJA

- Per konstruktorių - priklausomybių injekcija, kai konteineris iškviečia klasės konstruktorių su argumentais, atstovaujančius kitos klasės priklausomybę.
- Per `set*` - konteineris iškvietęs klasės konstruktorių be parametrų, po to kviečia `set*` metodus, atstovaujančius kitų klasių priklausomybes.
- Vienu metu galima naudoti abu priklausomybių injekcijos būdus. Rekomenduojama privalomoms priklausomybėms naudoti injekciją per konstruktorių, o neprivalomoms injekciją per `set*` metodus.

PRIKLAUSOMYBIŲ INJEKCIJA PER KONSTRUKTORIŲ 1

```
public class SpellChecker {
  public SpellChecker(){
    System.out.println("Inside SpellChecker constructor." );
  }
  public void checkSpelling() {
    System.out.println("Inside checkSpelling." );
  }
}
public class TextEditor {
  private SpellChecker spellChecker;
  public TextEditor(SpellChecker spellChecker) {
    System.out.println("Inside TextEditor constructor." );
    this.spellChecker = spellChecker;
  }
  public void spellCheck() { spellChecker.checkSpelling(); }
}
```

PRIKLAUSOMYBIŲ INJEKCIJA PER KONSTRUKTORIŲ 2

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"
  <!-- Definition for textEditor bean -->
  <bean id="textEditor" class="com.tutorialspoint.TextEditor">
    <constructor-arg ref="spellChecker"/>
  </bean>
  <!-- Definition for spellChecker bean -->
  <bean id="spellChecker" class="com.tutorialspoint.SpellChecker"/>
</beans>
```

PRIKLAUSOMYBIŲ INJEKCIJA PER SET* 1

```
public class TextEditor {
    private SpellChecker spellChecker;
    // a setter method to inject the dependency.
    public void setSpellChecker(SpellChecker spellChecker) {
        System.out.println("Inside setSpellChecker." );
        this.spellChecker = spellChecker;
    }
    // a getter method to return spellChecker
    public SpellChecker getSpellChecker() {
        return spellChecker;
    }
    public void spellCheck() {
        spellChecker.checkSpelling();
    }
}
```

PRIKLAUSOMYBIŲ INJEKCIJA PER SET* 2

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"
  <!-- Definition for textEditor bean -->
  <bean id="textEditor" class="com.tutorialspoint.TextEditor">
    <property name="spellChecker" ref="spellChecker"/>
  </bean>
  <!-- Definition for spellChecker bean -->
  <bean id="spellChecker" class="com.tutorialspoint.SpellChecker"/>
</beans>
```

VIDINIO SPRING BEAN DEKLARACIJA

- Kaip Java klasėje galima deklaruoti vidinę klasę, taip pat vidiniai bean yra deklaruojami kito bean `<property/>` arba `<constructor-arg/>` deklaracijos apimtyje:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"
  <bean id="outerBean" class="...">
    <property name="target">
      <bean id="innerBean" class="..."/>
    </property>
  </bean>
</beans>
```

UŽDUOTIS 3 - SUKURTI BEAN SERVICEB

- Sukurti `ServiceB` klasę `lt.itmokymai.spring` pakete ir realizuoti `ServiceA` priklausomybę naudojant `set*` priklausomybės injekciją:

```
package lt.itmokymai.spring;
public class ServiceB {
  private ServiceA serviceA;
  public void setServiceA(ServiceA serviceA) {
      this.serviceA = serviceA;
  }
  public String getResult () {
      return "ServiceB result:" + serviceA.getResult();
  }
}
```

UŽDUOTIS 3 - SUKURTI BEAN SERVICEB

- `src/main/resources/application-context.xml` užregistruoti `ServiceB`

```
<bean id="serviceBBean" class="lt.itmokymai.spring.ServiceB">
  <property name="serviceA" ref="serviceABean" />
</bean>
```

- `lt.itmokymai.spring.App.main()` atspausdinti `ServiceB.getResult()` metodo rezultatą
- Įvykdyti `lt.itmokymai.spring.App` klasę.

UŽDUOTIS 3 - PAKEISTI BEAN SERVICEB

- Pakeisti `lt.itmokymai.spring.ServiceB` klasę, kad naudotų `ServiceA` priklausomybės injekciją per konstruktorių.
- Pakeisti `ServiceB` bean deklaraciją `src/main/resources/application-context.xml` byloje.

UŽDUOTIS 3 - SUKURTI BEAN SERVICEC

- Sukurti `lt.itmokymai.spring.ServiceC` klasę, kuri paveldėtų `lt.itmokymai.spring.ServiceA` klasę.
- Pridėti `ServiceC` bean deklaraciją `src/main/resources/application-context.xml` byloje taip, kad paveldėtų `ServiceA` bean `message` atributo priskirtą reikšmę.

UŽDUOTIS 3 - SUKURTI BEAN SERVICEC

- Perkrauti `ServiceC.getResult()` metodą:

```
public String getResult() {
  return "ServiceC result:"+getMessage();
}
```

- `lt.itmokymai.spring.App.main()` atspausdinti `ServiceC.getResult()` metodo rezultatą į komandinę eilutę.

SPRING BEAN KOLEKCIJOS

AKADEMIJA.IT
INFOBALT IR TECH CITY

SPRING BEAN KOLEKCIJŲ INJEKCIJA

- Spring bean XML aprašas leidžia deklaruoti kolekcijos tipo atributus:
  - `<list>` - skirtas apibrėžti primitivių tipų arba bean sąrašą. Leidžia pasikartojančias reikšmes.
  - `<set>` - skirtas apibrėžti primitivių tipų arba bean aibę, be pasikartojančių reikšmių.
  - `<map>` - skirtas rakto - reikšmės kolekcijai, kai raktas ir reikšmė gali būti bet koks tipas.
  - `<props>` - skirtas rakto - reikšmės kolekcijai, kai raktas ir reikšmė yra String tipo.

KOLEKCIJŲ INJEKCIJOS PAVYZDYS - KLASĖ

```
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;
public class Customer
{
  private List<Object> lists;
  private Set<Object> sets;
  private Map<Object, Object> maps;
  private Properties pros;
  //...
}
```

KOLEKCIJŲ INJEKCIJOS PAVYZDYS - LIST

```
<bean id="..." class="Customer">
  <property name="lists">
    <list>
      <value>1</value>
      <ref bean="PersonBean" />
      <bean class="com.mkyong.common.Person">
        <property name="name" value="mkyongList" />
        <property name="address" value="address" />
        <property name="age" value="28" />
      </bean>
    </list>
  </property>
</bean>
```

KOLEKCIJŲ INJEKCIJOS PAVYZDYS - SET

```
<bean id="..." class="Customer">
  <property name="sets">
    <set>
      <value>1</value>
      <ref bean="PersonBean" />
      <bean class="com.mkyong.common.Person">
        <property name="name" value="mkyongSet" />
        <property name="address" value="address" />
        <property name="age" value="28" />
      </bean>
    </set>
  </property>
</bean>
```

KOLEKCIJŲ INJEKCIJOS PAVYZDYS - MAP

```
<bean id="..." class="Customer">
  <property name="maps">
    <map>
      <entry key="Key 1" value="1" />
      <entry key="Key 2" value-ref="PersonBean" />
      <entry key="Key 3">
        <bean class="com.mkyong.common.Person">
          <property name="name" value="mkyongMap" />
          <property name="address" value="address" />
          <property name="age" value="28" />
        </bean>
      </entry>
    </map>
  </property>
</bean>
```

KOLEKCIJŲ INJEKCIJOS PAVYZDYS - PROPS

```
<bean id="..." class="Customer">
    <property name="props">
      <props>
        <prop key="admin">admin@nospam.com</prop>
        <prop key="support">support@nospam.com</prop>
      </props>
    </property>
</bean>
```

UŽDUOTIS 4

- Aprašyti spring bean kolekciją, kurioje būtų produktų sąrašas (kaip React)
- Kiekvienas produktas taip pat savaime turi būti bean
- Produktas (produkto klasė) turi turėti title, image ir kitus parametrus (kaip React)
- ServiceC klasėje sukurti produktų sąrašą
- Nuskaityti tokią kolekciją iš App main klasės ir išvesti į ekraną (konsolę) produktų pavadinimus

SPRING BEAN AUTOMATINIS SURIŠIMAS

Autowire

SPRING BEAN AUTOMATINIS SURIŠIMAS

- Spring konteineris gali automatiškai surišti (angl. autowire) bendradarbiaujančius bean, nenaudodamas tiesiogiai `<constructor-arg>` ir / arba `<property>` elementuose nurodytos priklausomybių injekcijos.
- Automatinio surišimo būdai:
  - no - Naudojamas pagal nutylėjimą. Nurodo kad automatinis surišimas nebus atliekamas ir surišimas turi būti deklaruotas tiesiogiai nurodant priklausomybių injekcijas.

SPRING BEAN AUTOMATINIS SURIŠIMAS

- byName - surišimas pagal savybės vardą. Spring konteineris bean attributų vardus, kaip identifikatorius, naudos kitų užregistruotų bean paieškai ir priklausomybės injekcijai.
- byType - surišimas pagal bean klasės tipą. Spring konteineris bean attributų tipus naudos kitų to paties tipo užregistruotų bean paieškai ir priklausomybės injekcijai. Konteineris meta klaidą jei randamas daugiau nei vienas atributo tipo bean.

SPRING BEAN AUTOMATINIS SURIŠIMAS

- `constructor` - panašus į byType, tačiau naudojamas tik konstruktoriaus parametrams. Konteineris meta klaidą jei randamas daugiau nei vienas parametro tipo bean
- pavyzdžiui, žymime bean, kad Spring pats suieškotų priklausomybes pagal tipą:

```
<bean id="..." class="..." autowire="byType">
```

- tačiau norint naudoti kelis režimus, reikia naudoti anotacijas

UŽDUOTIS 5 - PRIKLAUSOMYBĖS

- ServiceB naudoja ServiceA. Todėl pašalinkite xml konfigūracijoje nuorodas į ServiceA
  - tiek constructor-arg parametrus, tiek property
- Nurodykite xml konfigūracijoje ServiceB parametrą autowire
  - pabandykite parinkti skirtingas reikšmes
- atkreipkite dėmesį kad constructor ir byType gali ieškoti tipo, o to tipo bean gali egzistuoti keli, ir jis nežinos, kurį pasirinkti

SPRING BOOT PRIKLAUSOMYBIŲ MIGRACIJA

UŽDUOTIS 6 - PRIKLAUSOMYBIŲ MIGRACIJA

- Trumpam grįžkime prie savo Spring Boot Starter archetipu paremto projekto
- Norėtumėme naudoti ne 4.1.8, o 5.0.10 Spring versiją
- Spring Boot kartu atsineša (transitive) ir Spring [Core], todėl reikia migruoti Spring Boot į naujesnę versiją
- migruoti lengviausia po truputį per minor versijas, o ne iškart į naujausią
- `pom.xml` pakeisti versiją:

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.4.2.RELEASE</version>
</parent>
```

UŽDUOTIS 6 - PRIKLAUSOMYBIŲ MIGRACIJA

- Kadangi pasikeitė Spring Boot API, tai ir DataSource Spy nebeveikia
- Iš tikrųjų, reikia ne tik `pom.xml` pakeisti versiją, bet ir pridėti dependency:

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.4.2.RELEASE</version>
</parent>
...
<dependency>
  <groupId>com.integralblue</groupId>
  <artifactId>log4jdbc-spring-boot-starter</artifactId>
  <version>1.0.2</version>
</dependency>
```

UŽDUOTIS 6 - PRIKLAUSOMYBIŲ MIGRACIJA

- `application.properties` pridėti:

```
logging.level.jdbc.sqlonly=DEBUG
```

- `AppConfig.java` ištrinti visą klasės turinį

```
@Configuration
public class AppConfig { }
```

- Pakeitus pom.xml, reikia pergeneruot `mvn eclipse:eclipse` ir Eclipse projektą atnaujinti Refresh
- Eclipse galima atsidaryti `pom.xml` ir Dependencies Hierarchy turi būti nauja `spring-core` versija
- Paleisti aplikaciją

UŽDUOTIS 6 - PAKETŲ MIGRACIJA

- App.java reikia migruoti SpringBootServletInitializer, nes jis buvo perkeltas į naują package ir yra pažymėtas kaip @Deprecated
- Taip žymima minor versijose tai, kas bus pašalinta pasikeitus major versijai. Taigi šį `import`

```
import org.springframework.boot.context.web.SpringBootServletInitializer;
```

- migruojame (pakeičiame) į

```
import org.springframework.boot.web.support.SpringBootServletInitializer;
```

- Paleisti aplikaciją ir pasitikrinti, kad ji veikia tiek tomcat, tiek spring-boot

UŽDUOTIS 7 - MIGRACIJA 1.4.2 -> 1.4.7

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.4.7.RELEASE</version>
</parent>
```

- mvn clean install, mvn eclipse:eclipse
- Eclipse atsidaryti `pom.xml` ir Dependencies Hierarchy turi būti 4.3.9 `spring-core` versija
- Paleisti aplikaciją ir pasitikrinti, kad ji veikia tiek tomcat, tiek spring-boot
  - t.y. /calc servisas turi veikti

UŽDUOTIS 7 - MIGRACIJA 1.4.7 -> 1.5.0

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.0.RELEASE</version>
</parent>
```

- pasitikrinti, kad `spring-core` versija 4.3.6
- HelloControllerTest.java ištrinti (-) / pridėti (+) eilutes

```
-import org.springframework.boot.test.IntegrationTest;
-import org.springframework.boot.test.SpringApplicationConfiguration;
-import org.springframework.test.context.web.WebAppConfiguration;
+import org.springframework.boot.test.context.SpringBootTest;
-@SpringApplicationConfiguration(classes = App.class)
-@WebAppConfiguration
-@IntegrationTest({"server.port:0",
-"spring.datasource.url:jdbc:h2:mem:hello-world-calc;DB_CLOSE_ON_EXIT=FALSE"})
+@SpringBootTest(classes = App.class, webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
```

UŽDUOTIS 7 - MIGRACIJA 1.5.0 -> 1.5.17

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.17.RELEASE</version>
</parent>
```

- pasitikrinti, kad `spring-core` versija 4.3.20

UŽDUOTIS 7 - MIGRACIJA 1.5.17 -> 2.0.0

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.17.RELEASE</version>
</parent>
...
<!-- pasalinti spring loaded -->
-<dependencies>
-  <dependency>
-    <groupId>org.springframework</groupId>
-    <artifactId>springloaded</artifactId>
-    <version>${spring-loaded.version}</version>
-  </dependency>
-</dependencies>
```

- pasitikrinti, kad `spring-core` versija 5.0.4

UŽDUOTIS 7 - MIGRACIJA 1.5.17 -> 2.0.0

- App.java pakeisti servlet import

```
-import org.springframework.boot.web.support.SpringBootServletInitializer;
+import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;
```

- nuo šios versijos keičiasi ir aplikacijos paleidimo komandos, nes serveris keičiasi iš tomcat7 į tomcat8
- paleisti spring-boot

```
mvn clean install spring-boot:run -Dspring-boot.run.arguments=--server.port=8081
```

- paleisti įdėtinį tomcat8

```
$ mvn clean install org.codehaus.cargo:cargo-maven2-plugin:1.7.0:run \
> -Dcargo.maven.containerId=tomcat8x -Dcargo.servlet.port=8081 \
> -Dcargo.maven.containerUrl=http://repo1.maven.org/maven2/org/apache/tomcat/tomcat/8.5.35/tomcat-8.5.35.zip
```

UŽDUOTIS 7 - MIGRACIJA 2.0.0 -> 2.0.6

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.6.RELEASE</version>
</parent>
```

- pasitikrinti, kad `spring-core` versija 5.0.10

KITOJE PASKAITOJE

Spring konfigūracija anotacijomis. Spring Junit testai. Spring Boot. Rest Servisai. Swagger. Spring Boot Junit testai