NACIONĀLAIS ATTĪSTĪBAS PLĀNS 2020
EIROPAS SAVIENĪBA
Eiropas Sociālais fonds
IEGULDĪJUMS TAVĀ NĀKOTNĒ

Java programmēšanas pamati

16. NODARBĪBA

SPRING


spring

SPRING

- Vispopulārākā programmatūras izstrādes ietvars (framework) priekš Enterprise Java
- Spring padara uzņēmumu programmu izstrādi ātru, vienkāršu un kvalitatīvu
- Spring ļauj veidot programmas, kuras sastāv no viegli testējam un pārizmantojama koda
- Spring atbalsta dažādas uz JVM laižamas valodas – Groovy, Kotlin
- Spring ļauj veidot programmas ar dažādu arhitektūru

SPRING
PAMATI

DIZAINA FILOZOFIJA

- Izvēles iespēja ik uz soļa
- Ļauj atlikt kritisku dizaina lēmumus līdz pēdējam brīdim (maini datu glabāšanas veidu, kad nepieciešams)
- Spring iedrošina elastību
- Spring neuzstāj uz viena konkrēta risinājuma
- Nodrošina stingru savietojamību
- Velta lielu uzmanību ērtam API dizainam
- Nosaka augstus koda kvalitātes standartus

ATKARĪBAS INJEKCIJA

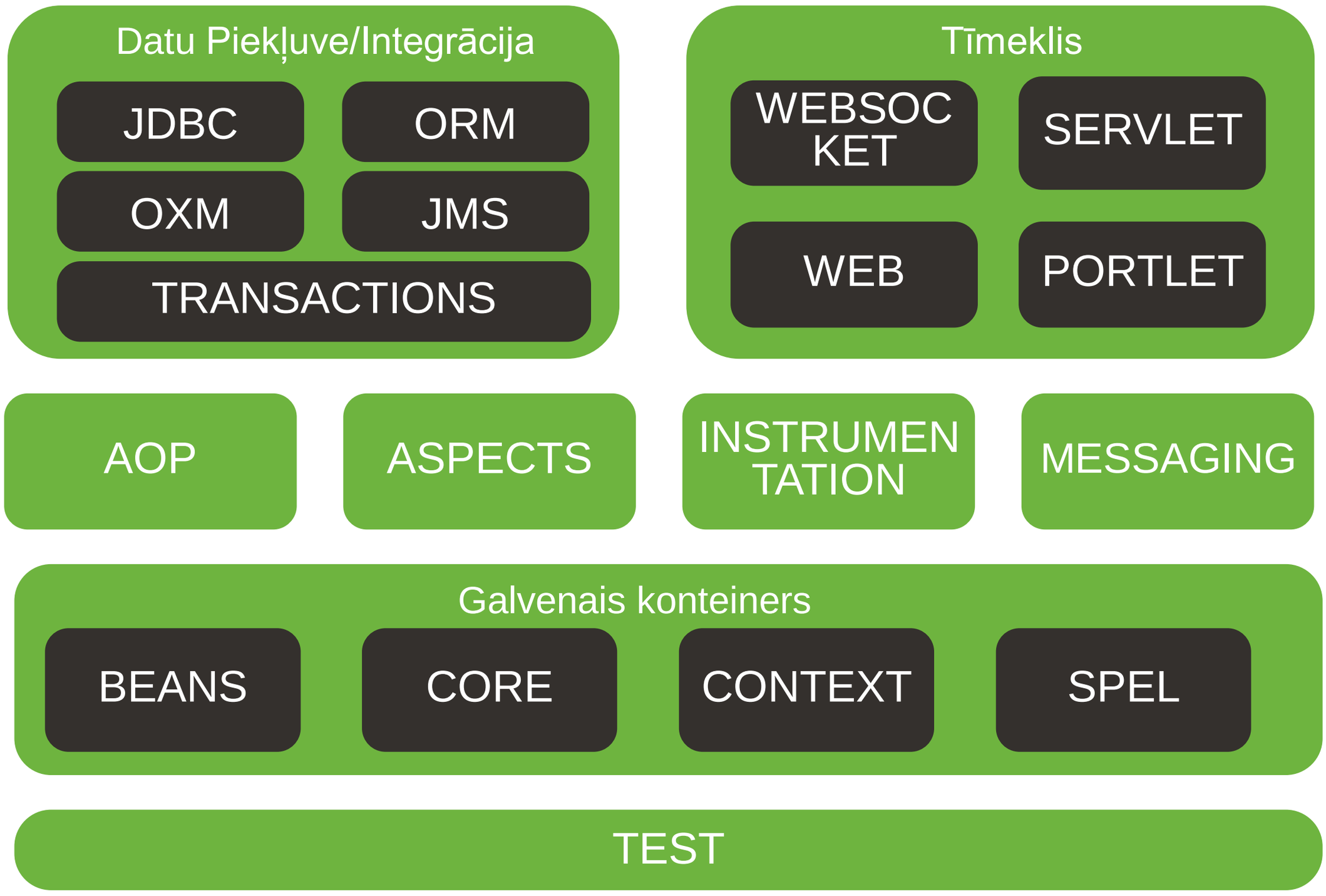
- Tehnoloģija ar kuru Spring asociējas ir Atkarības Injekcija (AI)
- AI palīdz būt klasēm neatkarīgām vienai no otras
- AI palīdz klases testēt neatkarīgi vienai no otras
- AI palīdz salīmēt šī klases vienā programmā un tai pat laikā turēt tās neatkarīgi vienai no otras

ASPEKTU ORIENTĒTA PROGRAMMĒŠANA (AOP)

- Viena no Spring pamata komponentēm ir aspektu orientēta programmēšana
- Fukcijas, kas aptver vairākus lietojumprogrammas punktus, sauc par transversālām bažām (cross-cutting concerns) un šīs bažas tiek nošķirtas no lietojumprogrammas
- Aspektus var izmantot: logošanā, drošības nodrošināšanai, kešatmiņai, u.c.
- AI pamatvienība ir klase, bet AOP tas ir aspekts
- AI atdala objektus vienu no otra, bet AOP atdala transversālās bažas
- Spring AOP modulis ļauj definēt metožu pārtvērējus (interceptors) un krustpunktus, lai atdalītu šo kodu

SPRING
MODUĻI

SPRING MODUĻI



GALVENAIS KONTAINERS

- **«Core»** modulis nodrošina sistēmas galvenās sastāvdaļas, tai skaitā Kontroles Inversijas mehānismu un AI
- **«Bean»** modulis nodrošina BeanFactory, kas ir Factory dizaina tehnikas realizācija
- **«Context»** modulis ir balstīts uz moduļiem Core un Beans. Tas ir līdzeklis, lai piekļūtu visiem definētajiem un konfigurētajiem objektiem. Interfeiss ApplicationContext ir moduļa Context centrālais punkts.
- **«SpEL»** modulis nodrošina spēcīgu izteiksmju valodu, lai manipulētu ar objekta grafu programmas izpildlaikā

DATU PIEKĻUVE/INTEGRĀCIJA

- **«JDBC»** modulis nodrošina JDBC abstrakcijas slāni, kas novērš nepieciešamību pēc garlaicīgas ar JDBC saistītu kodēšanu.
- **«ORM»** modulis nodrošina integrācijas slāņus populārām objektu-relāciju kartēšanas API, tostarp JPA, JDO, Hibernate un iBatis.
- **«OXM»** modulis nodrošina abstrakcijas slāni, kas atbalsta Object/XML kartēšanas ieviešanu JAXB, Castor, XMLBeans, JiBX un XStream.
- **«JMS»** (Java Messaging Service) modulis satur funkcijas ziņojumu sagatavošanai un lietošanai.
- **«Transaction»** modulis atbalsta programmētu un deklaratīvu darījumu pārvaldību (transaction management) klasēm, kurās tiek ieviestas īpašas interfeisi, kā arī visiem jūsu POJO.

TĪMEKLIS

- **«Web»** modulis nodrošina tīmekļa orientētas lietojumprogrammas izstrādei neieciešamās funkcijas, kā arī atbilstošā konteksta inicializāciju
- **«Web-MVC»** modulis saturs Spring(a) Model-View-Controller (MVC) implementāciju tīmekļa lietojumprogrammām
- **«Web-Socet»** modulis nodrošina divu virzienu komunikāciju starp klientu un serveri lietujummprogrammā, kas ir balstīta uz WebSocket tehnoloģijas
- **«Web-Portlet»** modulis nodrošina MVC ieviešanu, kas tiek izmantota portlet vidē, un atspoguļo Web-Servlet moduļa funkcionalitāti.

CITI

- **«AOP»** modulis nodrošina aspektu orientētu programmēšanas implementāciju, kas dod iespēju definēt metodes-pārtvērējus un ļauj atdalīt kodu no biznesa loģikas
- **«Aspects»** modulis nodrošina integrāciju ar AspectJ, kas ir jaudīgs AOP framevorks
- **«Instrumentation»** modulis atbalsta klašu instrumentāciju un klašu ielādētāju (class-loader) implementāciju, kas tiek izmantota noteiktiem aplikāciju serveriem
- **«Messaging»** modulis atbalsta STOMP kā WebSoket apakšprotokolu izmantošanai programmās
- **«Test»** modulis nodrošina Spring komponenšu testēšanu, izmantojot Junit vai TestNg ietvaram (framework)

SPRING
KONTEINERIS

(1) SPRING IOC KONTEINERIS

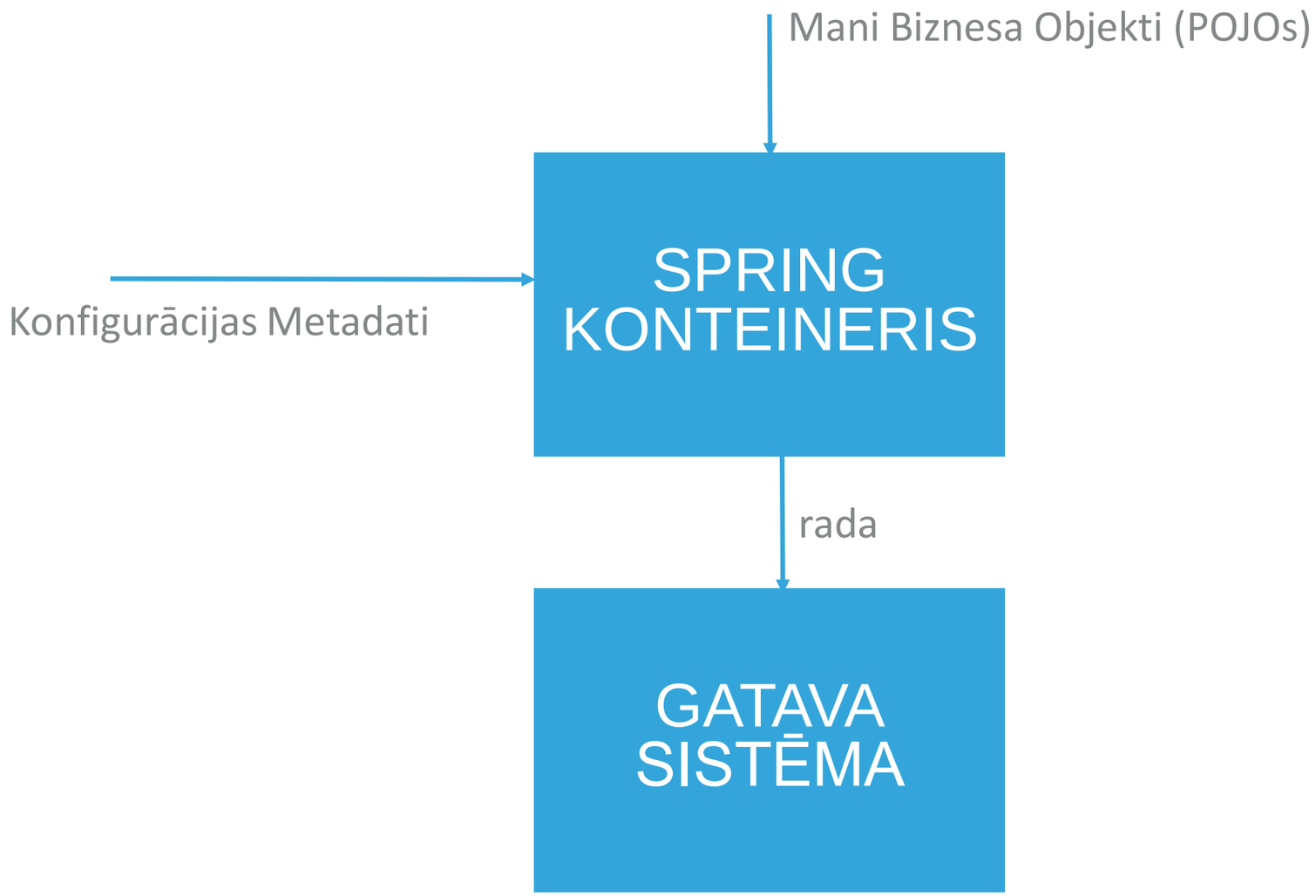
- Spring konteiners ir pamatā Spring ietvaram (framework):
  - izveido objektus
  - savieno objektus kopā
  - konfigurē objektus
  - pārvalda objektu pilnu dzīvesciklu
  - izmanto AI, lai pārvaldītu komponentes, kas veido programmu

(2) SPRING IOC KONTEINERIS

- Spring konteiners saņem instrukcijas:
  - kurus objektus instancēt (instantiate), kofigurēt un salikt kopā izmantojot dotos metadatus
  - meta datus var definēt izmantojot XML, Java anotācijas vai Java kodu

- Spring IoC konteiners izmanto Java POJO klases un konfigurācijas metadatus, lai izveidotu pilnībā nokonfigurētu un palaižamu sistēmu

SHĒMA


Mani Biznesa Objekti (POJOs)
Konfigurācijas Metadati
SPRING KONTEINERIS
rada
GATAVA SISTĒMA

KONFIGURACIJA

ANOTĀCIJU BĀZĒTA KONFIGURĀCIJA

- Centrālie elementi konfigurācijā ir:
  - Ar anotāciju **@Configuration** apzīmētas klases
  - Ar anotāciju **@Bean** apzīmētas metodes
- **@Bean** antoācija tiek izmantota, lai norādītu ka metode instancē, konfigurē un inicializē jaunu objektu, kuru pārvaldīs Spring IoC konteineris
- **@Bean** anotētas metodes var tik izmantotas jebkurā Spring **@Component** klasē, lai gan parasti tiek izmantotas **@Configuration** anotētās klasēs
- Ar **@Configuration** anotēta klase norāda, ka klases mērķis ir «bīna» definēšana
- **@Configuration** anotētās klasēs «bīnos» ir iespējams injicēt citu «bīnu» vienkārši izsaucot attiecīgo metodi

PIEMĒRS

```
@Configuration
public class AppConfig {

    @Bean
    public MyService myService() {
        return new MyServiceImpl();
    }
}
```

```
@Configuration
public class AppConfig {

    @Bean(name = "myThing")
    public Thing thing() {
        return new Thing();
    }
}
```

«BĪNU» INJICĒŠANA

- Kad «bīnam» ir atkarība vienam no otra, tad šādu atkarību var injicēt, vienkārši izsaucot injicējamā «bīna» metodi

```
@Configuration
public class AppConfig {

    @Bean
    public BeanOne beanOne() {
        return new BeanOne(beanTwo());
    }

    @Bean
    public BeanTwo beanTwo() {
        return new BeanTwo();
    }
}
```

KOMPONENTE

- **@Component** ir vispārēja anotācija jebkurai Spring komponentei
- **@Service, @Repository, @Controller** ir anotācija Spring komponenšu apzīmēšanai, tikai katrai no tām ir savs gadījums, kad to izmanto:
  - **@Service** servisa noteikšanai
  - **@Repository** krātuves noteikšanai
  - **@Controller** kontroliera noteikšanai

KOMPONENTE

- **@Component** ir vispārēja anotācija jebkurai Spring komponentei
- **@Service, @Repository, @Controller** ir anotācija Spring komponenšu apzīmēšanai, tikai katrai no tām ir savs gadījums, kad to izmanto:
  - **@Service** servisa noteikšanai
  - **@Repository** krātuves noteikšanai
  - **@Controller** kontroliera noteikšanai

AUTOMATIZĒTA KLAŠU ATRAŠANA UN «BĪNU» REĢISTRĒŠANA

- **@ComponentScan** anotācija nodrošina automātiski atrast «bīnus» un reģistrēt tos Spring sistēmā (šai anotācijai ir jāatrodas kopā ar **@Configuration**)
- Atribūts **basePackages** norāda pakotni, kurā «bīni» ir jāmeklē

```
@Configuration
@ComponentScan(basePackages = "org.example")
public class AppConfig  {
    ...
}
```

ATKARĪBU INJEKCIJA

(1) ANOTĀCIJA @Autowired

- **@Autowired** anotācija norāda, kur tieši injekcijai ir jānotiek
- **@Autowired** var novietot virs lauka vai metodes, tad Spring saprot, ka ir nepieciešams šeit injicēt «bīnu»

```
public class SimpleMovieLister {

    private MovieFinder movieFinder;

    @Autowired
    public void setMovieFinder(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }

    // ...
}
```

(1) ANOTĀCIJA @Autowired

- **@Autowired** var novietot virs konstruktora, kas arī mūsdienās ir labā prakse

```
@Component
public class SimpleMovieLister {

        private final MovieFinder movieFinder;

        @Autowired
        public SimpleMovieLister(MovieFinder movieFinder) {
                this.movieFinder = movieFinder;
        }
}
```

(1) ANOTĀCIJA @Autowired

- **@Autowired** anotācija norāda, kur tieši injekcijai ir jānotiek
- **@Autowired** var novietot virs lauka vai metodes, tad Spring saprot, ka ir nepieciešams šeit injicēt «bīnu»

```
public class SimpleMovieLister {

    private MovieFinder movieFinder;

    @Autowired
    public void setMovieFinder(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }

    // ...
}
```

(1) ANOTĀCIJA @Autowired

- **@Autowired** anotācija norāda, kur tieši injekcijai ir jānotiek
- **@Autowired** var novietot virs lauka vai metodes, tad Spring saprot, ka ir nepieciešams šeit injicēt «bīnu»

```
public class SimpleMovieLister {

    private MovieFinder movieFinder;

    @Autowired
    public void setMovieFinder(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }

    // ...
}
```

SPRING BOOT


spring
BOOT

PĀRSKATS

- Spring-Boot dod iespēju viegli un ātri izveidot produkcijas vides kvalitātes Spring bāzētu lietojumprogrammu
- Spring-Boot lietojumprogrammām Spring konfigurācija ir ļoti minimāla

IEZĪMES

- Izveido patstāvīgu (stand-alone) Spring lietojumprogrammu
- Iekļauts Tomcat, Jetty, vai Undertow aplikāciju serveris
- Piedāvā «starter» atkarības, lai vienkāršotu jūsu «build» konfigurāciju
- Automātiski nokonfigurē Spring un 3ās puses bibliotēkas
- Sagatavo produkcijai gatavas iezīmes – metrikas, veselības pārbaude un ārēja konfigurācija
- Izslēdz nepieciešamību pēc XML konfigurācijas

IESĀC AR SPRING INITUALIZR

- Izvēlies vēlamo konfigurāciju savam projektam no https://start.spring.io/
- Uzģenerē savu projekta mapi
- Importē to savā IDEA

BUILD.GRADLE

```
plugins {
	id 'org.springframework.boot' version '2.4.3'
	id 'io.spring.	' version '1.0.11.RELEASE'
	id 'java'
}

group = 'com.example'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '1.8'

repositories {
	mavenCentral()
}

dependencies {
	implementation 'org.springframework.boot:spring-boot-starter-web'
	testImplementation 'org.springframework.boot:spring-boot-starter-test'
}

test {
	useJUnitPlatform()
}
```

SPRING PALAIŠANA

- Metodē «main» tiek izsaukt SpringApplication metode run
- SpringApplication piestartē mūsu programmu un automātiski konfigurēto aplikāciju serveri (Tomcat)

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoApplication {

	public static void main(String[] args) {
		SpringApplication.run(DemoApplication.class, args);
	}

}
```

CLASSPATH AKTAKARĪBAS

- SpringBoot nodrošina vairākus «Startētājus» (starter), kas pievieno nepieciešamās bibliotēkas Jūsu classpath
- spring-boot-starter-web iekļauj sevī komponentes, kas ir nepieciešamas tīmekļa programmai, piemēram Tomcat serveris
- spring-boot-starter-test iekļauj nepieciešamās komponentes tīmekļa aplikācijas testēšanai
- Sparudnis dependency-management nodrošina šo atkarību pārvaldību

ATKARĪBU PĀRVALDĪBA

- Katrā Spring Boot laidienā ir apkopots to atbalstīto atkarību saraksts
- Praksē nav nepieciešams norādīt precīzu atkarības versiju, jo SpringBoot to pārvalda Jūsu vietā
- Kad tiek atjaunota SpringBoot versija, tad tiek atjaunotas arī atkarību versijas
- Bet ir atļauts arī norādīt sev vēlamo versiju atkarībai, ja tas ir nepieciešams

ATSAUCES

- https://spring.io/guides
- https://docs.spring.io/spring-framework/docs/current/reference/html/

LATVIJAS UNIVERSITATE
ANNO 1919

VADĪBAS UN UZŅĒMĒJDARBĪBAS