

Razvoj informacijskih sistemov

Informatika in podatkovne tehnologije

UNI, 2. letnik

Študijsko leto 2025 / 2026

Luka Pavlič, UM, FERI

Zaledni sistemi v Javi

03. in 10. 10. 2025

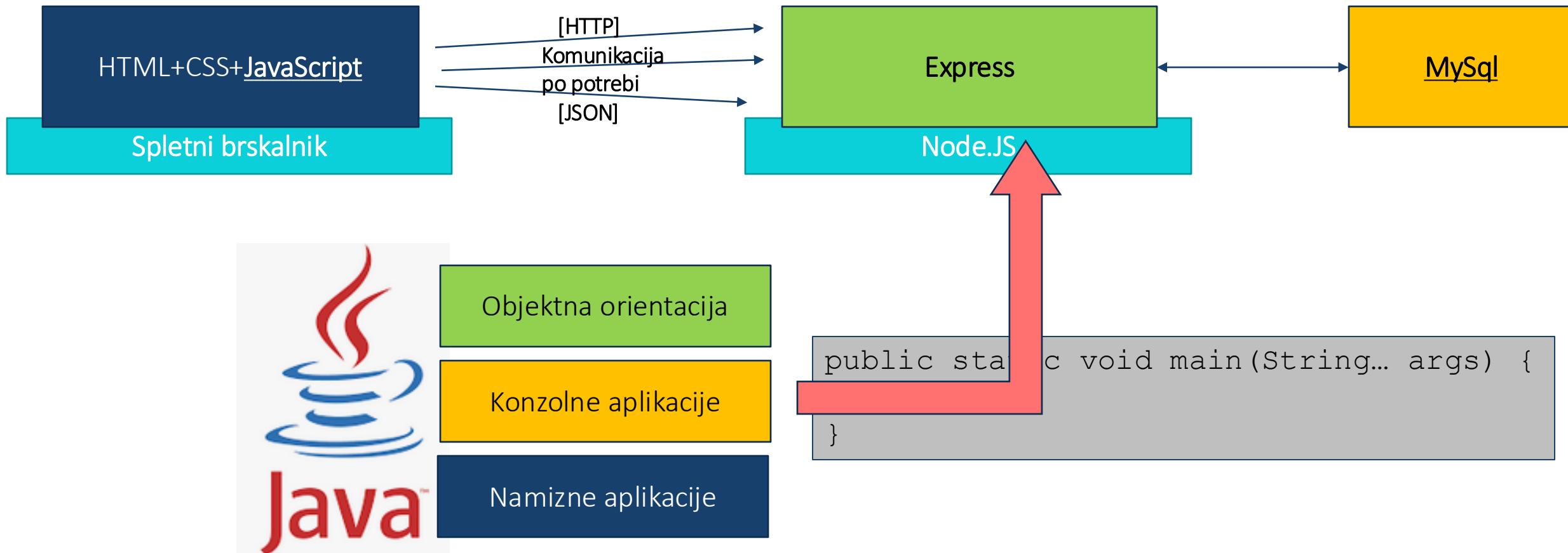
Sodobna Java

Maven / Gradle

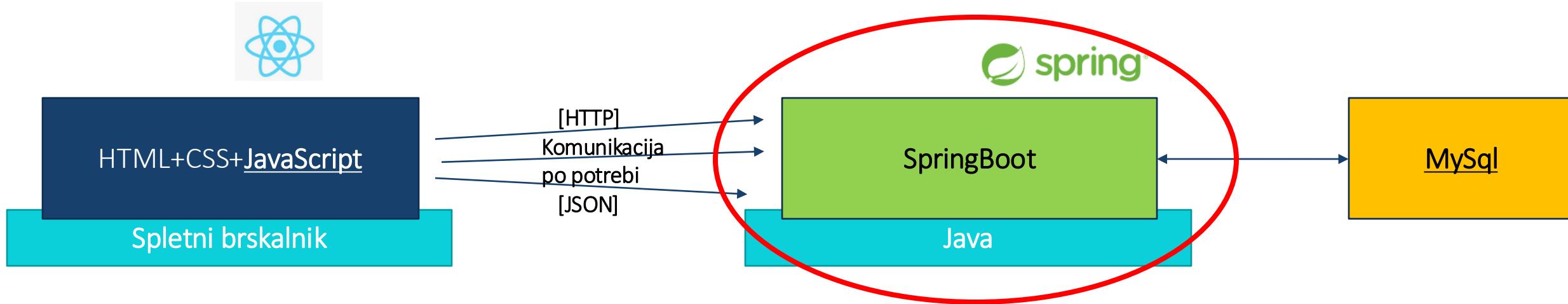
SpringBoot

Sodobna Java na primeru **SpringBoot**

Preden začnemo – 1. letnik...



Preden začnemo – 1. letnik → RIS...

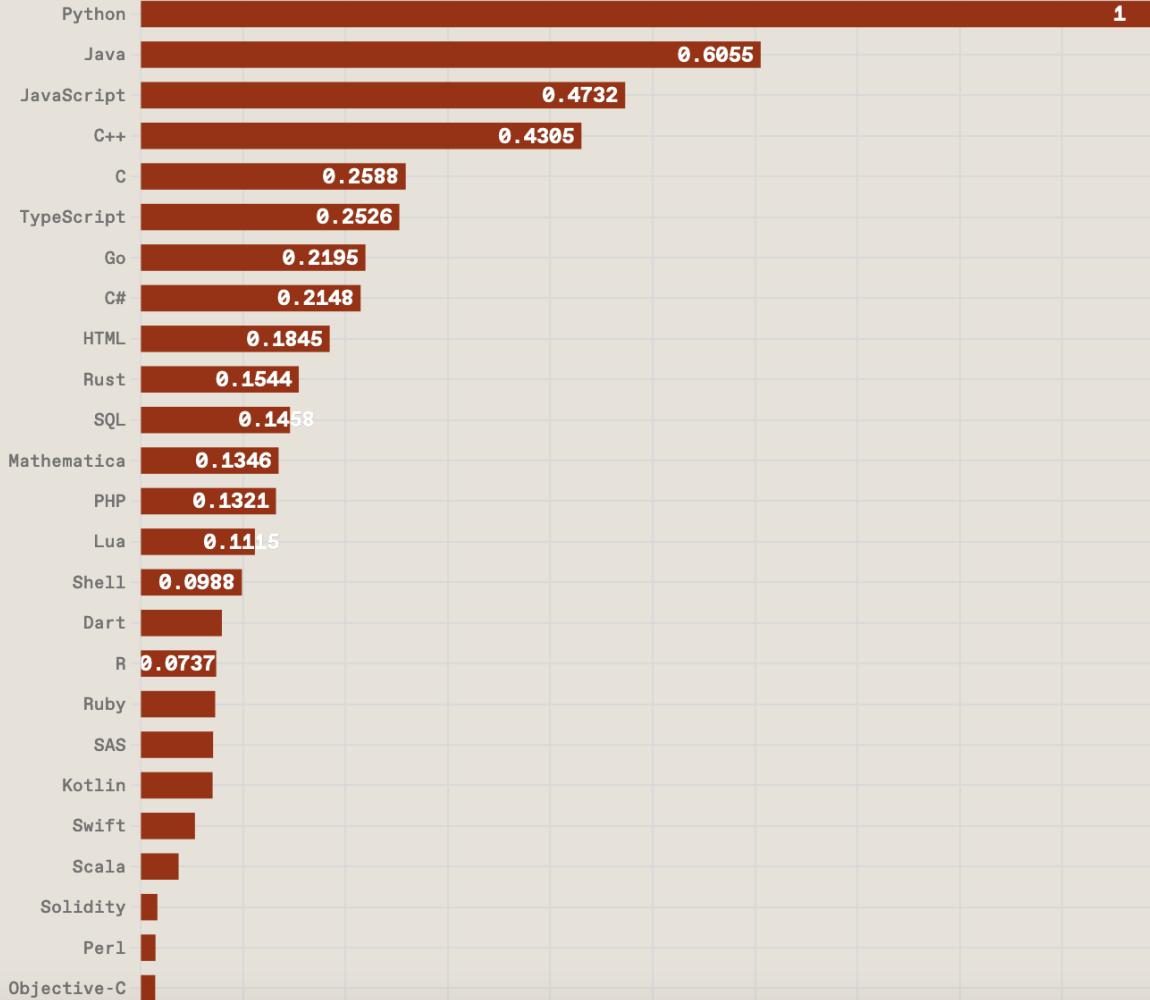


- Objektna orientacija
- Konzolne aplikacije
- Namizne aplikacije

#1 HITER pregled nekaterih novejših konceptov Java
#2 HITER pregled namena in zmožnosti ogrodja Spring
“Civiliziranega” bratranca SpringBoot
NA PRIMERIH
Podrobnosti boste znali raziskati sami

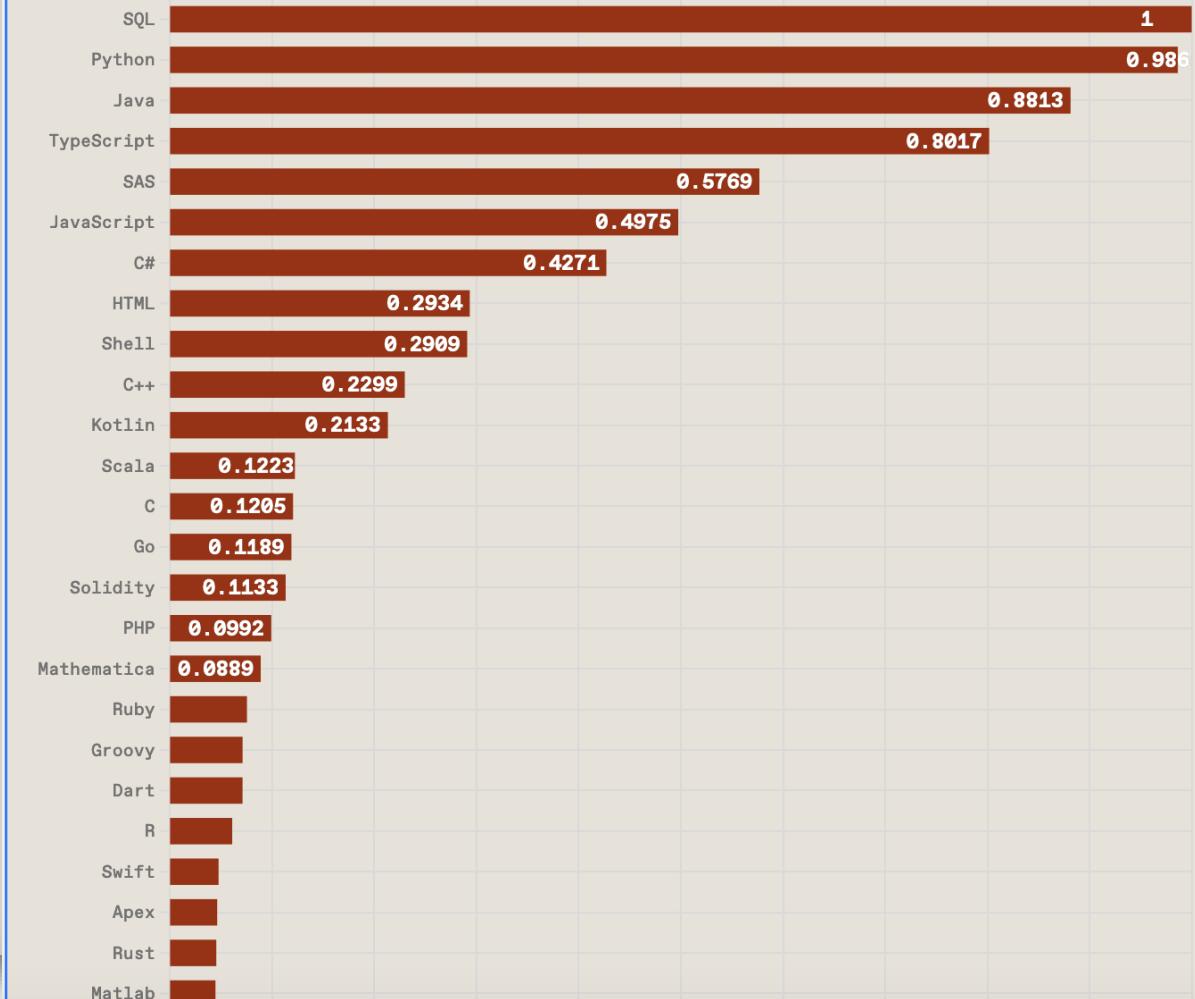
Top Programming Languages 2024

Click a button to see a differently weighted ranking



Top Programming Languages 2024

Click a button to see a differently weighted ranking



2025: Java SE 25-LTS / 23

<https://docs.oracle.com/en/java/javase/25/index.html>

<https://docs.oracle.com/en/java/javase/25/docs/api/index.html>

<https://docs.oracle.com/javase/tutorial/index.html>

<https://docs.oracle.com/javase/tutorial/java/concepts/>

<https://www.oracle.com/java/>

<https://www.oracle.com/java/technologies/java-se-glance.html>

<https://www.java.com/en/>

<https://openjdk.java.net/>

<https://blogs.oracle.com/javamagazine/>

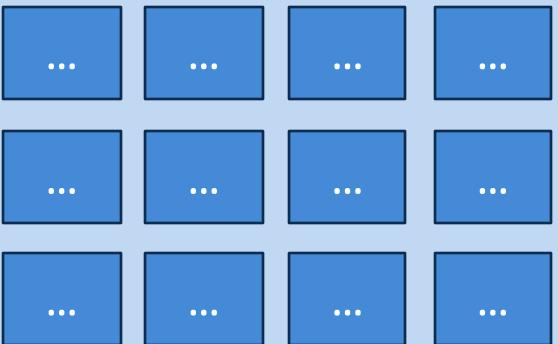
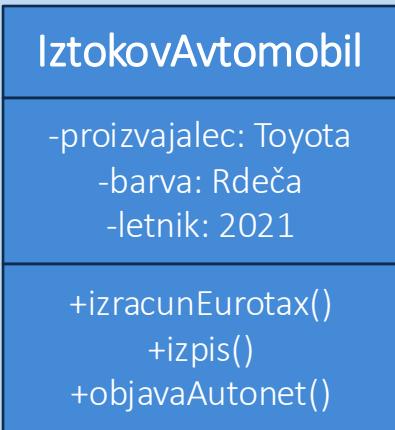
www.jcp.org

www.oracle.com/java



Osnove so jasne...

Avtomobil



+ izracunPorabe(
porabaNa100 : double, prevozenihKm: double
) : double

+ OBDOBJE_AMORTIZACIJE : int = 10

ref1

ref2

```

public class Oseba {
    public Oseba() {
    }

    public Oseba(String ime, String priimek) {
        this.ime = ime;
        this.priimek = priimek;
    }

    private String ime;

    private String priimek;

    public String getIme() {
        return ime;
    }

    public void setIme(String ime) {
        this.ime = ime;
    }

    public String getPriimek() {
        return priimek;
    }
}

```

Ograjevanje

public
protected
private
(privzeto)

Dedovanje

enkratno
poljubna globina
implementacija vmesnikov
referanca super
privzet "override"!

Polimorfizem

"Različno odzivanje na isto sporočilo"

V povezavi z dedovanjem:
1 podrazred JE 1 nadrazred!

```

Oseba peterKlepec=new Oseba("Peter","Klepec");
Oseba peterDrugi=peterKlepec;
Bogat peterBogatas=peterDrugi;
Bogat bogatunPeter=new Oseba("Peter","Klepec");

```

```
public class ZlatiRacun extends BancniRacun {
```

```
    public ZlatiRacun() {
    }
```

```
    public ZlatiRacun(String iban) {
        super(iban);
    }
}
```

```
private double dovoljenLimit=Constants.PRIVZETO_DOVOLJEN_LIMIT_ZLATI_RACUN;
```

```

public class Oseba implements Bogat {
    @Override
    public void doniraj(String namen, double znesek) {
        System.out.println("Ja vešda.... nič neo!");
    }
}

```

https://github.com/lukapavlic/primeri/tree/master/java_lambda





<https://maven.apache.org>

<https://mvnrepository.com>

Kako razvijamo?

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>si.um.feri</groupId>
<artifactId>java_lambda</artifactId>
<version>1.0-SNAPSHOT</version>

<properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
</properties>

</project>
```

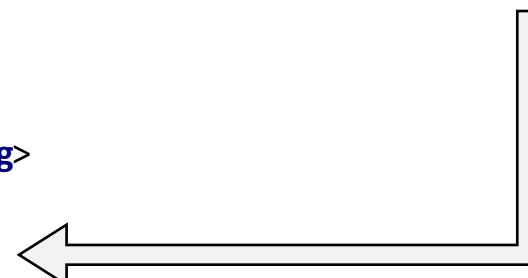
https://github.com/lukapavlic/primeri/blob/master/java_lambda/pom.xml

Alternativa:



<https://gradle.org>

```
<!-- https://mvnrepository.com/artifact/org.apache.httpcomponents/httpclient -->
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
    <version>4.5.13</version>
</dependency>
```



Project Object Model (pom.xml)

Metapodatki o projektu

| Projekt:

groupId

artifactId

version

packaging

| Dependencies – odvisnosti aplikacije

| Build – vtičniki, struktura projekta,...

| Repositories – združuje odvisnosti in artefakte

Tudi kreiranje / uporaba t.i.
Archetype-ov

Dependencies Scope

Odkrivanje oddaljenih odvisnosti je lahko omejeno z določitvijo okvira:

Compile - odvisnost je na voljo v projektu, privzet okvir

Provided - odvisnost bo zagotovljena s strani JDK ali aplikacijskega strežnika/zabojnika med izvajanjem

Runtime - odvisnost ni potrebna za prevajanje, vendar je potrebna med izvajanjem

Test – odvisnost je na voljo samo za faze preizkusa in izvajanje testov

System – potrebno je podati sistemsko pot

Import – uporabimo, kadar je odvisnost tipa pom, potrebna uvedba modulov in razdelek <dependencyManagement>

Default (Build) Lifecycle

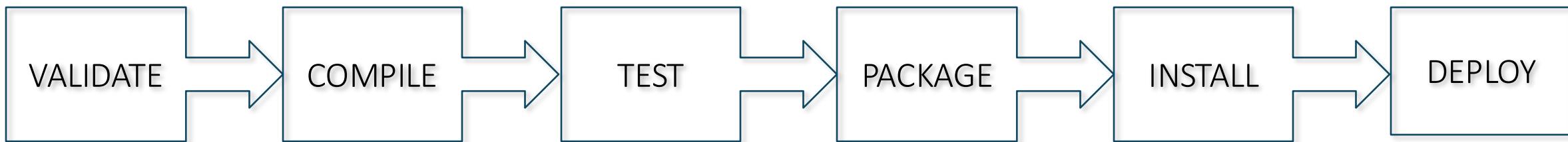
Maven™

Primarni cikel graditve aplikacije

Sestoji iz 21 faz

<https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html>

Glavne faze, ki se po navadi izvedejo:



Primer ukaza za zagon posamezne faze: ***mvn test***

izvedejo se vse faze do tiste, ki smo jo zagnali z ukazom:



Lambda izrazi v Javi - Za motivacijo

```
public static void main(String[] args) throws Exception {  
    new Thread(new Runnable() {  
        public void run() {  
            for (int i=0;i<100;i++) {  
                System.out.println(".");
                try {  
                    Thread.sleep(5);  
                } catch (InterruptedException e) {  
                    e.printStackTrace();  
                }  
            }  
        }.start();  
}
```

```
public static void main(String[] args) throws Exception {  
  
    new Thread(() -> {  
        for (int i=0;i<100;i++) {  
            System.out.println(".");
            try {  
                Thread.sleep(5);  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }).start();  
}
```

```
btnPozeni.addActionListener(new java.awt.event.ActionListener()  
{  
    public void actionPerformed(java.awt.event.ActionEvent evt) {  
        btnPozeniActionPerformed(evt);  
    }  
});
```

```
btnPozeni.addActionListener(evt-> {  
    btnPozeniActionPerformed(evt);  
});
```

“Closures”

So torej res “le” sintaktični sladkorček

Poslušalci

Klasični povratni klici

Anonimni notranji razredi

...

All Methods	Instance Methods	Abstract Methods	De
Modifier and Type	Method		
void	accept(T t)		
default Consumer<T>	andThen(Consumer<? super T> after)		

```
seznam.forEach(new Consumer<String>() {  
    public void accept(final String s) {  
        System.out.println(s);  
    }  
});
```

```
for (String s:seznam)  
    System.out.println(s);
```

```
seznam.forEach (s -> System.out.println(s));
```

```
seznam.forEach(System.out::println);
```

```
public void forEach(Consumer<? super E> action)
```

Module java.base
Package java.util.function

Interface Consumer<T>

Type Parameters:

T - the type of the input to the operation

Manj (bolj berljive) kode Ali: kako lambde najbolj izkoristiti

Primer

urejanje oseb v kolekciji po priimku

```
Arrays.sort(seznamOseb, (lhs, rhs) -> lhs.getPriimek().compareTo(rhs.getPriimek()));
```

Izpis urejenih oseb

```
seznamOseb.forEach (o -> System.out.println(o));
```

Ideal Use Case for Lambda Expressions

Approach 1: Create Methods That Search for Members That Match One Characteristic

Approach 2: Create More Generalized Search Methods

Approach 3: Specify Search Criteria Code in a Local Class

Approach 4: Specify Search Criteria Code in an Anonymous Class

Approach 5: Specify Search Criteria Code with a Lambda Expression

Approach 6: Use Standard Functional Interfaces with Lambda Expressions

Approach 7: Use Lambda Expressions Throughout Your Application

Approach 8: Use Generics More Extensively

Approach 9: Use Aggregate Operations That Accept Lambda Expressions as Parameters

Sintaksa lambda izraza

λ izraz z izrazom:

(niz1, niz2) -> niz1.compareTo(niz2);

parametri λ izraza \rightarrow izraz kot telo λ izraza ;

λ izraz z blokom programske kode:

```
(niz1, niz2) -> { return niz1.compareTo(niz2); }
```

parametri λ izraza → { Blok kode kot telo λ izraza zaključen z return stavkom }

Sintaksa lambda izraza

veljavni λ izrazi

$() \rightarrow \{\}$
 $(niz) \rightarrow \text{System.out.println}(niz)$
 $niz \rightarrow \text{System.out.println}(niz)$
 $(niz) \rightarrow \{\text{System.out.println}(niz);\}$
 $() \rightarrow \text{„rezultat“}$
 $() \rightarrow \{ \text{return } \text{„rezultat“}; \}$
 $(ime) \rightarrow \{ \text{return } \text{„Pozdravljen, “} + \text{ime}; \}$

neveljavni λ izrazi

$() \rightarrow \{ \text{„rezultat“} \}$
 $(ime) \rightarrow \text{„Pozdravljen, “} + \text{ime}$
 $(ime) \rightarrow \text{return } \text{„Pozdravljen, “} + \text{ime}$

Doseg lokalnih spremenjivk in atributov v Lambda izrazu

Veljajo enaka pravila, kot pri anonimnih notranjih razredih
(efektivno) final za lokalne spremenljivke v dosegu metode
Normalen dostop do atributov razreda

```
int st;
List<String> seznam=new ArrayList<>();

void main() throws Exception {
    seznam.forEach (s-> {
        System.out.println(st+" "+s);
        st++;
    });
}

List<String> seznam=new ArrayList<>();

void main() throws Exception {
    final int st=0;
    seznam.forEach (s-> {
        System.out.println(st+" "+s);
        st++; //NAPAKA
    });
}
```

Funkcijsko programiranje v Javi

Imperativni slog

```
public static void main(String[] args) {  
    List<Integer> seznam = Arrays.asList(1, 2, 3, 4, 5, 6, 7);  
    for (int i=0; i < seznam.size(); i++) {  
        System.out.println(seznam.get(i));  
    }  
}
```

Deklarativni slog

```
public static void main(String[] args) {  
    List<Integer> seznam = Arrays.asList(1, 2, 3, 4, 5, 6, 7);  
    seznam.forEach(System.out::println);  
}
```



Funkcijsko programiranje v Javi

Funkcije višjega razreda (higher-order functions)

Lastnost prvo-razredne funkcije omogoča definiranje funkcij višjega reda v funkcionalnem programiranju

Funkcije višjega reda lahko prejmejo funkcijo kot argument ali pa vrnejo funkcijo kot rezultat

```
// Funkcija višjega reda
Collections.sort(seznam, new Comparator<Integer>() {
    @Override
    public int compare(Integer o1, Integer o2) {
        return o1.compareTo(o2);
    }
});
Collections.sort(seznam, (o1, o2) -> o1.compareTo(o2));
System.out.println(seznam);
```

Funkcijsko programiranje v Javi

Dodatno lahko vmesniki vsebuje tudi privzete in statične metode, ki imajo ustrezeno implementacijo

```
public interface FizičnaOseba {  
    default void odpriRačun() { System.out.println("Odprt je bil račun za fizično osebo"); }  
  
    static void preveriStarost(int starost) {  
        System.out.println("Preverjanje starosti Fizične osebe: ");  
        System.out.println(starost > 18);  
    }  
}
```

Reference metod

Primer: izpis elementov seznama celih števil

```
List<Integer> seznam = Arrays.asList(1, 3, 2, 7, 6, 5, 4);

// imperativno
for (int i = 0; i < seznam.size(); i++) {
    System.out.println(seznam.get(i));
}

// funkcionalno
seznam.forEach((število) -> System.out.println(število));
// referenca metode
seznam.forEach(System.out::println);
```

Namesto klica metode lahko uporabimo referenco metode
Razrešitev ujemanja parametrov metod se izvede **avtomatsko!**

Razred Optional<T>

Uveden z namenom premoščanja težav, ki jih povzroča **referenca null** :

ovojni razred – ovije atribut razreda

koncept povzet po programskeh jezikih Haskell in Scala

Razred ponuja nabor metod, s katerimi se je lahko (bolj ali manj) elegantno izogniti preverjanju reference null

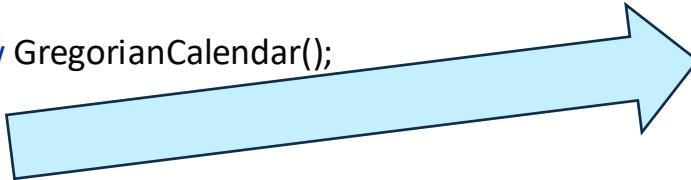
Razred smiselno uporabiti v situacijah, ko obstaja možnost, da metoda ne vrne pričakovanega rezultata

(npr. ni zadetkov, ker noben element ne ustreza zahtevanim pogojem)

All Methods	Static Methods	Instance Methods	Concrete Methods
Modifier and Type	Method		
static <T> Optional<T> empty()			
boolean	equals(Object obj)		
Optional<T>	filter(Predicate<? super T> predicate)		
<U> Optional<U>	flatMap(Function<? super T, ? extends Optional<? extends U>> mapper)		
T	get()		
int	hashCode()		
void	ifPresent(Consumer<? super T> action)		
void	ifPresentOrElse(Consumer<? super T> action, Runnable emptyAction)		
boolean	isEmpty()		
boolean	isPresent()		
<U> Optional<U>	map(Function<? super T, ? extends U> mapper)		
static <T> Optional<T> of(T value)			
static <T> Optional<T> ofNullable(T value)			
Optional<T>	or(Supplier<? extends Optional<? extends T>> supplier)		
T	orElse(T other)		
T	orElseGet(Supplier<? extends T> supplier)		
T	orElseThrow()		
<X extends Throwable>	orElseThrow(Supplier<? extends X> exceptionSupplier)		
T			
Stream<T>	stream()		
String	toString()		

Torej:

```
public class BancniRacun implements Bogat, Serializable {  
  
    private String iban;  
  
    private Calendar odprtOd=new GregorianCalendar();  
  
    private Oseba lastnik;  
  
    protected BigDecimal trenutnoStanje;  
  
    boolean aktiven;
```



```
public class BancniRacun implements Bogat, Serializable {  
  
    private String iban=Optional.empty();  
  
    private Calendar odprtOd=new GregorianCalendar();  
  
    private Optional<Oseba> lastnik;  
  
    protected BigDecimal trenutnoStanje;  
  
    boolean aktiven;  
  
    public Optional<Oseba> getLastnik() {  
        return lastnik;  
    }  
  
    public void setLastnik(Optional<Oseba> lastnik)  
    {  
        this.lastnik = lastnik;  
    }
```

```
System.out.println(jankovRacun.getLastnik().get().getPriimek());
```

Odpiramo nov bančni račun.

Odpiramo nov bančni račun z iban:SI56-0000-0000-2222

Exception in thread "main" java.lang.NullPointerException Create breakpoint :
at si.um.feri.banka.Aplikacija.main(Aplikacija.java:31)

Optional - Prirejanje vrednosti

Optional.empty()

```
BancniRacun jankovRacun=new BancniRacun("SI56-0000-0000-2222");
```

Optional.of()

```
jankovRacun.setLastnik(new Oseba("Janko","Bogataš"));
```

```
jankovRacun.setLastnik(Optional.of(new Oseba("Janko","Bogataš")));
```

```
if (jankovRacun.getLastnik().isPresent())
    System.out.println(jankovRacun.getLastnik().get().getPriimek());
```

Record-i

Ko objekt ustvarimo, njegove vsebine ne moremo več spremenjati
Uporablja se za **nespremenljive** objekte

Tradicionalno

```
class Oseba {  
  
    Oseba(string ime, string priimek) {  
        this.ime = ime;  
        this.priimek = priimek;  
    }  
  
    String ime() { return ime; }  
    String priimek() { return priimek; }  
  
    public boolean equals(Object o) {...}  
    public int hashCode() {...}  
    public String toString() {...}  
}
```

Z uporabo record

```
public record Oseba(String ime, String priimek){}
```

Record-i – dodatni atributi

Record lahko vsebuje le attribute, definirane v glavi razreda

```
//ni dovoljeno
public record Oseba(string ime, string priimek){
    String naslov;
}
```

Lahko pa vsebujejo dodatne **statične** attribute

```
//dovoljeno
public record Oseba(string ime, string priimek){
    static String NEZNANO = "Neznano";
}
```

Record-i – dedovanje in vmesniki

Record razredi ne morejo dedovati od drugih razredov

```
//ni dovoljeno
public record Oseba(string ime, string priimek) extends Clovek {
    ...
}
```

Lahko pa implementirajo vmesnike

```
//dovoljeno
public record Oseba(string ime, string priimek) implements Bogat, Serializable{
    @Override
    public void doniraj(String namen, double znesek) {
        ...
    }
}
```

Ogrodje Spring



...in njegov bolj “civiliziran” bratranec: SpringBoot

<https://spring.io/>

(MVC) (IoC) ogrodje za razvoj (strežniških) Javanskih aplikacij
Ekosistem projektov (modulov)

<https://spring.io/projects>

Ideja AOP

Filozofija Spring Boot

Generiranje osnovnega projekta s poenostavljenim upravljanjem odvisnosti

Izhod: izvršljiva datoteka JAR

Samodejna konfiguracija

```
mvnw clean package  
mvnw spring-boot:run
```



[productsMS-0.0.1-SNAPSHOT.jar](#)

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

```
@SpringBootApplication
public class MeasurementsApplication {

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

}
```

“Convention over configuration and fat JAR files”

```
2021-12-15 12:42:26.661 INFO 28820 --- [           main] s.u.f.m.MeasurementsApplication      : Starting MeasurementsApplication using Java 17.0.1 on LAPTOP-8075D with PID 28820 (C:\Users\Luka\SVN\informatika2021\primeri\demo_spring_measurements\target\classes) started by Luka in C:\Users\Luka\SVN\informatika2021\primeri\demo_spring_measurements)
2021-12-15 12:42:26.666 INFO 28820 --- [           main] s.u.f.m.MeasurementsApplication      : The following profiles are active: dev
2021-12-15 12:42:27.729 INFO 28820 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2021-12-15 12:42:27.790 INFO 28820 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 50 ms. Found 2 JPA repository interfaces.
2021-12-15 12:42:28.338 INFO 28820 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8280 (http)
2021-12-15 12:42:28.350 INFO 28820 --- [           main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2021-12-15 12:42:28.351 INFO 28820 --- [           main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.52]
2021-12-15 12:42:28.468 INFO 28820 --- [           main] o.a.c.c.C.[localhost].[/api/v1]        : Initializing Spring embedded WebApplicationContext
2021-12-15 12:42:28.468 INFO 28820 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1744 ms
```

The screenshot shows the Spring Initializr interface with the following configuration:

- Project:** Maven Project (selected)
- Language:** Java (selected)
- Dependencies:** Spring Data JPA (selected), SQL (selected)
- Spring Boot:** 2.6.1 (selected)
- Add Dependencies... CTRL + B** button

A modal dialog is open for creating a new project:

- Name: demo
- Location: ~\SVN\informatika2021\primeri\demo
- Language: Java, Kotlin, Groovy
- Type: Maven
- Group: com.example
- Artifact: demo
- Package name: com.example.demo
- Project SDK: Java 17 version 17.0.1
- Java: 11
- Packaging: Jar (selected)

3.2. Installing the Spring Boot CLI

The Spring Boot CLI (Command Line Interface) is a command line tool that you can use to quickly prototype with Spring. It lets you run Groovy scripts, which means that you have a familiar Java-like syntax without so much boilerplate code.

You do not need to use the CLI to work with Spring Boot, but it is a quick way to get a Spring application off the ground without an IDE.

3.2.1. Manual Installation

You can download the Spring CLI distribution from the Spring software repository:

- [spring-boot-cli-2.6.1-bin.zip](#)
- [spring-boot-cli-2.6.1-bin.tar.gz](#)

Cutting edge snapshot distributions are also available.

Once downloaded, follow the [INSTALL.txt](#) instructions from the unpacked archive. In summary, there is a `spring` script (`spring.bat` for Windows) in a `bin/` directory in the `.zip` file. Alternatively, you can use `java -jar` with the `.jar` file (the script helps you to be sure that the classpath is set correctly).

Kdaj bi uporabili Spring boot?

Hitro in enostavno začnemo z razvojem

Testiranje novih funkcionalnosti ogrodja

Ni potrebnega veliko predznanja (konfiguracija, odvisnosti ipd.)

Razvijamo majhne in avtonomne aplikacije (npr. mikrostoritve)

Razvijamo distribuirane aplikacije

Zaledni sistemi v oblaku

...

Oznaka @SpringBootApplication

Katerikoli razred, ki ima javno statično metodo "main"

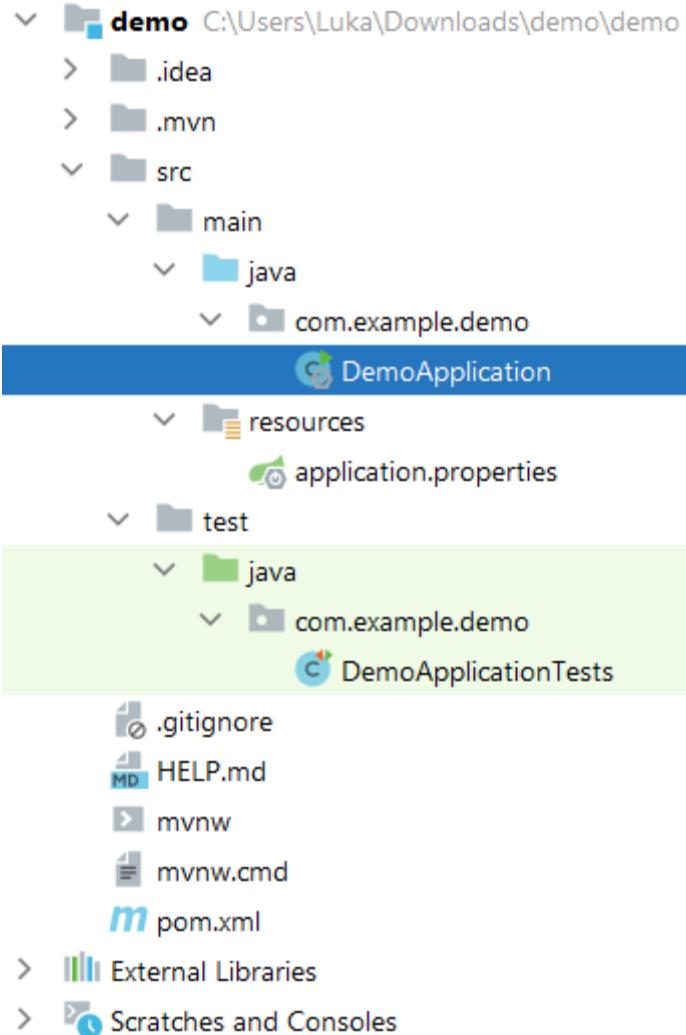
V projektu (paket aplikacije in pod-paketi) omogoči samodejno iskanje komponent (@Bean, @Component, @Controller, @Configuration ...)

Nosi vlogo (samodejno) konfiguracijskega razreda

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = {@org.springframework.context.annotation.ComponentScan.Filter(type = org.springframework.context.annotation.FilterType.CUSTOM,
classes = {org.springframework.boot.context.TypeExcludeFilter.class}),@org.springframework.context.annotation.ComponentScan.Filter(type =
org.springframework.context.annotation.FilterType.CUSTOM, classes = {org.springframework.boot.autoconfigure.AutoConfigurationExcludeFilter.class})})
public @interface SpringBootApplication
extends annotation.Annotation
```

Indicates a configuration class that declares one or more @Bean methods and also triggers auto-configuration and component scanning. This is a convenience annotation that is equivalent to declaring @Configuration, @EnableAutoConfiguration and @ComponentScan.

“Nulta aplikacija”



```
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);  
    }  
}
```

Najosnovnejší primer

≡



Project

- Gradle Project
- Maven Project

Language

- Java
- Kotlin
- Groovy

Spring Boot

- 3.0.0 (SNAPSHOT)
- 3.0.0 (RC1)
- 2.7.6 (SNAPSHOT)
- 2.7.5
- 2.6.14 (SNAPSHOT)
- 2.6.13

Project Metadata

Group	si.um.feri
Artifact	demo
Name	demo
Description	Demo project for Spring Boot
Package name	si.um.feri.demo
Packaging	<input checked="" type="radio"/> Jar
Java	<input type="radio"/> 19 <input checked="" type="radio"/> 17 <input type="radio"/> 11 <input type="radio"/> 8

Dependencies

[ADD DEPENDENCIES... CTRL + B](#)

Spring Web WEB

Build web, including RESTful, applications using Spring MVC.
Uses Apache Tomcat as the default embedded container.

Spring Data JPA SQL

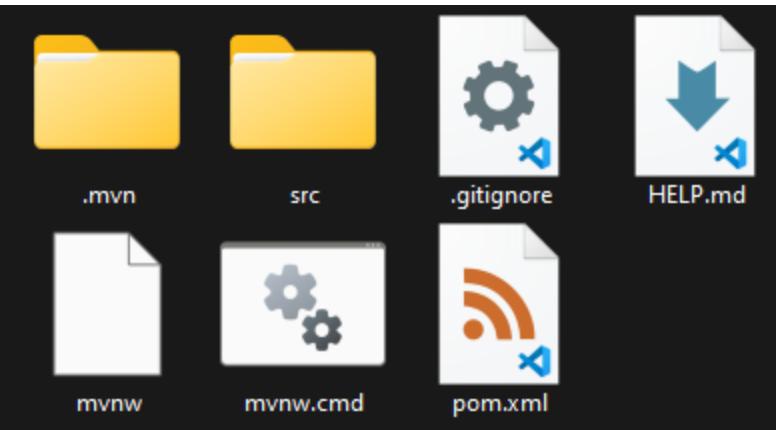
Persist data in SQL stores with Java Persistence API using
Spring Data and Hibernate.



[GENERATE CTRL + ↵](#)

[EXPLORE CTRL + SPACE](#)

[SHARE...](#)



application.properties:

```
server.port=8180  
server.servlet.context-path=/api/v1  
server.error.include-message=always
```

.\mvnw spring-boot:run

```
package si.um.feri.demo.rest;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class InfoController {
    @GetMapping("/info")
    public String pozdravi() {
        return "Zdravo";
    }
}
```

Sklad odvisnosti za spletnne zaledne sisteme

Nivo trajnih podatkov: JPA

Java Persistence API!

+odvisnosti do JDBC gonilnikov

+application.properties

Nivo poslovne logike

Lastni razredi

Lastna zrna

DAO komponente: Spring Repositories

API Nivo

Spring Web - RestController

Nivo trajnih podatkov

Podatkovni objekti

```
package si.um.feri.demo.vao;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Oseba {

    private String ime;

    private String priimek;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;

    public Oseba() {
    }

    ...
}
```

```
package si.um.feri.demo.dao;
```

```
import org.springframework.data.repository.CrudRepository;
import si.um.feri.demo.vao.Oseba;
```

```
public interface OsebaRepository extends CrudRepository<Oseba, Integer> { }
```

Dostop do trajnih podatkov

Uporaba v kodu

```
package si.um.feri.demo.rest;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import si.um.feri.demo.dao.OsebaRepository;
import si.um.feri.demo.vao.Oseba;
```

```
@RestController
public class InfoController {
```

```
    @Autowired
    OsebaRepository dao;
```

```
    @GetMapping("/info")
    public String pozdravi() {
        dao.save(new Oseba("Peter", "Klepec"));
        return "Zdravo. Dodali smo eno osebo.";
    }
}
```

Konfiguracija trajnih podatkov

application.properties (in različice)

```
spring.datasource.url=jdbc:mysql://localhost:3306/bazica
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.show-sql=true
spring.jpa.generate-ddl=true
```

JDBC gonilnik

```
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<scope>runtime</scope>
</dependency>
```

```
mysql> show tables;
+-----+
| Tables_in_bazica |
+-----+
| hibernate_sequence |
| oseba |
+-----+
2 rows in set (0.00 sec)

mysql> desc oseba;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id   | int       | NO  | PRI | NULL    |       |
| ime  | varchar(255) | YES |     | NULL    |       |
| priimek | varchar(255) | YES |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)

mysql> select * from oseba;
+---+---+---+
| id | ime | priimek |
+---+---+---+
| 1  | Peter | Klepec |
+---+---+---+
1 row in set (0.00 sec)

mysql>
```

Uporaba Jpa entitet v komponentah

Možna uporaba EntityManagerja – redkeje smiselno

Uporaba pripravljenih generičnih DAO “repositorijev”!

@Autowired

private EntityManager em;

Vmesnik CrudRepository

<https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html>

CRUD funkcionalnosti

Vmesnik PagingAndSortingRepository

<https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/PagingAndSortingRepository.html>

Dodane funkcionalnosti za sortiranje in delo s stranmi

Vmesnik JpaRepository

<https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html>

Dodaja dostop do naprednih JPA funkcionalnosti, npr. takojšnje uveljavnjanje sprememb (flush), paketno obdelavo objektov ipd.

CrudRepository

Method Summary

All Methods	Instance Methods	Abstract Methods
Modifier and Type		Method and Description
long		<code>count()</code> Returns the number of entities available.
void		<code>delete(T entity)</code> Deletes a given entity.
void		<code>deleteAll()</code> Deletes all entities managed by the repository.
void		<code>deleteAll(Iterable<? extends T> entities)</code> Deletes the given entities.
void		<code>deleteAllById(Iterable<? extends ID> ids)</code> Deletes all instances of the type T with the given IDs.
void		<code>deleteById(ID id)</code> Deletes the entity with the given id.
boolean		<code>existsById(ID id)</code> Returns whether an entity with the given id exists.
Iterable<T>		<code>findAll()</code> Returns all instances of the type.
Iterable<T>		<code>findAllById(Iterable<ID> ids)</code> Returns all instances of the type T with the given IDs.
Optional<T>		<code>findById(ID id)</code> Retrieves an entity by its id.
<S extends T> S		<code>save(S entity)</code> Saves a given entity.
<S extends T> Iterable<S>		<code>saveAll(Iterable<S> entities)</code> Saves all given entities.

API nivo

V Spring (Web) so to krmilniki

Oznaka @Controller

Sodelujejo v začetnem iskanju komponent Spring aplikacije

Podpirajo CDI (IoC) - @Autowired, @Value, @Inject, – in posredovanje instanc preko "set" metod in konstruktorjev

Namenski REST krmilniki - @RestController

= @Controller

= @ResponseBody (privzeto: JSON)

Označevanje metod krmilnika, ki se odzivajo na HTTP zahteve

```
@CrossOrigin  
@RestController  
public class ProductController {  
  
    @GetMapping("/products")  
    public @ResponseBody Iterable<Product> getAllProducts() {  
        return translateProductListToDtoList(dao.findAll());  
    }  
}
```

* Opcijske oznake

Oznake v krmilniku

Krovna oznaka: @RequestMapping

Value (=pot)

Method (=GET, POST, PUT, DELETE....)

...

Specializirane oznake:

@GetMapping

@PostMapping

@PutMapping

@PatchMapping

@DeleteMapping

...

Primer krmilnika

```
package si.um.feri.demo.rest;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import si.um.feri.demo.dao.OsebaRepository;
import si.um.feri.demo.vao.Oseba;
import java.util.logging.Logger;

@RestController
@RequestMapping("/osebe")
public class OsebaController {

    private static final Logger log = Logger.getLogger(OsebaController.class.toString());

    @Autowired
    OsebaRepository dao;

    @GetMapping("/{id}")
    public Oseba getOseba(@PathVariable int id) {
        log.info("Get osebe z id "+id);
        return dao.findById(id).get();
    }

    @PostMapping
    public Oseba postOseba(@RequestBody Oseba oseba) {
        log.info("Dodajanje nove osebe "+oseba);
        return dao.save(oseba);
    }
}
```

Swagger?

The screenshot shows the Swagger UI interface running at `http://127.0.0.1:8180/api/v1/swagger-ui/index.html`. The title bar indicates the URL is `127.0.0.1:8180/api/v1/swagger-ui/index.html` and the zoom level is 90%. The main content area displays an "OpenAPI definition" for version `v0` (OAS3) located at `/api/v1/v3/api-docs`. The interface includes a "Servers" dropdown set to `http://127.0.0.1:8180/api/v1 - Generated server url`.

```
<dependency>
<groupId>org.springdoc</groupId>
<artifactId>springdoc-openapi-ui</artifactId>
<version>1.6.6</version>
</dependency>
```

The screenshot shows the generated API documentation. It includes sections for the `oseba-controller` and `info-controller`. The `oseba-controller` section contains a `POST` endpoint for `/osebe` and a `GET` endpoint for `/osebe/{id}`. The `info-controller` section contains a `GET` endpoint for `/info`.

Celoten primer – korak po korak do delujoče REST storitve

1. Spring Initializr
2. Prenos projekta in uvoz v IDE
3. Dopolnitev odvisnosti – npr. JDBC gonilnik za podatkovno bazo
4. Dopolnitev konfiguracijske datoteke (application.properties)
5. Priprava razredov:
 1. JPA Entitete
 2. DAO vmesniki
 3. REST krmilniki
6. Dodajanje dodatnih funkcionalnosti preko odvisnosti
 1. Npr. Swagger
 2. Npr. Actuator
 3. ...
7. Zagon

SpringBoot primeri iz predavanj

https://github.com/lukapavlic/primeri/tree/master/spring_00_iz_predavanj

Primer na prosojnicah

https://github.com/lukapavlic/primeri/tree/master/spring_dummy_ms

Enostaven primer dveh povezanih entitet

https://github.com/lukapavlic/primeri/tree/master/spring_locations_ms

Primer izpostavitve trajnih podatkov skozi rest vmesnik

https://github.com/lukapavlic/primeri/tree/master/spring_measurements_backend

<https://github.com/lukapavlic/ris2025>

Eden celovitejših primerov

Primer, ki smo ga naredili skupaj