

Keretrendszer alapú programozás

Fazekas Csaba

fazekas.csaba@uni-eszterhazy.hu

Spring Boot alapok

Plain Old Java Object („jó öreg Java-objektum”)

A kifejezést Martin Fowler, Rebecca Parsons és Josh MacKenzie vezette be, 2000 szeptemberében:

„Azon tűnődtünk, miért ellenzik az emberek annyira a sima objektumok használatát a rendszereikben, és arra jutottunk, hogy azért, mert az egyszerű objektumoknak nem volt fantázianevekük. Így hát adtunk nekik egyet, és az szépen elterjedt.”

https://hu.wikipedia.org/wiki/Plain_Old_Java_Object

“Kiáltvány az Agilis szoftverfejlesztésért” aláírói:

Extreme
programming,
JUnit egységteszt,
Facebook-nál
dolgozik.

Kent Beck	James Grenning	Robert C. Martin
Mike Beedle	Jim Highsmith	Steve Mellor
Arie van Bennekum	Andrew Hunt	Ken Schwaber
Alistair Cockburn	Ron Jeffries	Jeff Sutherland
Ward Cunningham	Jon Kern	Dave Thomas
Martin Fowler	Brian Marick	

Kiáltvány az agilis szoftverfejlesztésért

A szoftverfejlesztés hatékonyabb módját tárjuk fel saját tevékenységünk és a másoknak nyújtott segítség útján. E munka eredményeképpen megtanultuk értékelni:

Az egyéneket és a személyes kommunikációt a módszertanokkal és eszközökkel szemben

A működő szoftvert az átfogó dokumentációval szemben

A megrendelővel történő együttműködést a szerződéses egyeztetéssel szemben

A változás iránti készséget a tervek szolgai követésével szemben

Azaz, annak ellenére, hogy a jobb oldalon szereplő tételek is értékkel bírnak, mi többre tartjuk a bal oldalon feltüntetetteket.

Kiáltvány az agilis szoftverfejlesztésért

12 elv:

1. Legfontosabbnak azt tartjuk, hogy az ügyfél elégedettségét a működő szoftver mielőbbi és folyamatos szállításával vívjuk ki.
2. Elfogadjuk, hogy a követelmények változhatnak akár a fejlesztés vége felé is. Az agilis eljárások a változásból versenyelőnyt kovácsolnak az ügyfél számára.
3. Szállíts működő szoftvert gyakran, azaz néhány hetenként vagy havonként, lehetőség szerint a gyakoribb szállítást választva.
4. Az üzleti szakértők és a szoftverfejlesztők dolgozzanak együtt minden nap, a projekt teljes időtartamában.
5. Építsd a projektet sikerorientált egyénekre. Biztosítsd számukra a szükséges környezetet és támogatást, és bízz meg bennük, hogy elvégzik a munkát.
6. A leghatásosabb és leghatékonyabb módszer az információ átadásának a fejlesztési csapaton belül, a személyes beszélgetés.
7. A működő szoftver az elsődleges mércéje az előrehaladásnak.
8. Az agilis eljárások a fenntartható fejlesztést pártolják. Fontos, hogy a szponzorok, a fejlesztők és a felhasználók folytonosan képesek legyenek tartani egy állandó ütemet.
9. A műszaki kiválóság és a jó terv folyamatos szem előtt tartása fokozza az agilitást.
10. Elengedhetetlen az egyszerűség, azaz az elvégezetlen munkamennyiség maximalizálásának művészete.
11. A legjobb architektúrák, követelmények és rendszertervek az önszerveződő csapatoktól származnak.
12. A csapat rendszeresen mérlegeli, hogy miképpen lehet emelni a hatékonyságot, és ehhez hangolja és igazítja az működését.

Plain Old Java Object („jó öreg Java-objektum”)

```
class Pojo {  
  
    private String name;  
  
    public String getName(){  
        return this.name;  
    }  
  
    public void setName(String name){  
        this.name = name;  
    }  
}
```

JavaBeans

“A **JavaBeans** egy újrafelhasználható, állapotot reprezentálni képes, átvitelt is segítő Java **tervezési minta**. A JavaBeans specifikációt a Sun dolgozta ki, eredetileg GUI komponensek adatrepresentációjához.”

Alapja egy POJO, de ennek felépítésekor bizonyos szabályokat be kell tartani:

- Szerializálhatónak kell lennie. -> implements Serializable
- Van publikus, paraméter nélküli konstruktora (default konstruktor).
- Privát mezői vannak, de publikus getter és setter metódusokkal el lehet érni ezeket.

JavaBeans

```
import java.io.Serializable;

public class PersonJavaBean implements Serializable {

    private String name;

    public PersonJavaBean() {
    }

    public PersonJavaBean(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```


JavaBeans

```
import java.io.*;

public class Main {
    public static void main(String[] args) {

        PersonJavaBean person = new PersonJavaBean("Hello World!");

        try {
            FileOutputStream fileOut = new FileOutputStream("data");
            ObjectOutputStream out = new ObjectOutputStream(fileOut);
            out.writeObject(person);
            out.close();
            fileOut.close();
            System.out.println("Serialization is done.");
        } catch (IOException e) {
            e.printStackTrace();
        }

        PersonJavaBean deserializedPerson = null;
        try {
            FileInputStream fileIn = new FileInputStream("data");
            ObjectInputStream in = new ObjectInputStream(fileIn);
            deserializedPerson = (PersonJavaBean) in.readObject();
            in.close();
            fileIn.close();
            System.out.println("Deserialization is ready.");
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }

        if (deserializedPerson != null) {
            System.out.println("Name: " + deserializedPerson.getName());
        }
    }
}
```

JavaBeans

data fájl tartalma:

```
"ÜsrPersonJavaBeanGŽ@r?Δ-kLnameLjava/lang/String;xpt  
Hello World!"
```

Fontos megjegyezni, hogy a “Serialisation” mindig implementáció függő. Lehet az egyszerű byte tömb, de lehet XML, JSON formátumú adat.

Enterprise JavaBeans

“Az **Enterprise JavaBeanek** (EJB) moduláris vállalati alkalmazásokban az üzleti logika megvalósításához használt szerveroldali komponensek. Működtetésükhöz egy EJB konténer szükséges, ami általában egy **Java EE alkalmazáserverben** fut. Az EJB specifikációt eredetileg az IBM adta ki 1997-ben. Később, 1999-ben, a **Sun Microsystems** is elfogadta, alkalmazta (EJB 1.0 és 1.1), és tovább terjesztették a **Java Community Process** alatt mint JSR19, JSR152, JSR220 és JSR318”

https://hu.wikipedia.org/wiki/Enterprise_JavaBeans

Különböző típusai vannak. Például: Session Beans, Entity Beans stb.

Enterprise JavaBeans

“Az EJB specifikációt arra tervezték, hogy egy alapvető módszer legyen a vállalati alkalmazásokban levő back-end üzleti kódok megvalósítására. **Ezekben a kódokban gyakran ugyanazok a hibák jelentek meg, és ezeket a problémákat a programozók újra és újra ugyanúgy oldották meg.** Az EJB-t arra hozták létre, hogy kezelje a perzisztenciát, a tranzakciók integritását és az adatvédelmet úgy, hogy a programozók szabadon koncentrálhassanak a megoldandó problémákra.”

https://hu.wikipedia.org/wiki/Enterprise_JavaBeans

Enterprise JavaBeans

Az EJB specifikáció részletezi egy alkalmazás szolgáltatásait:

- Tranzakciókezelés
- Integráció a Java Persistence API által nyújtott perzisztenciaszolgáltatással
- Konkurenciakezelés
- Események kezelése a Java Message Service-zel, és a Java EE Connector Architecture használatával
- Aszinkron metódushívás
- Feladatütemezés
- Elnevezési- és könyvtár szolgáltatások (JNDI)
- Távoli eljárashívás RMI-IIOP és webszolgáltatásokon keresztül
- Biztonság (Java Cryptography Extension (JCE) és JAAS)
- Szoftver komponensek telepítése alkalmazáserverre

Enterprise JavaBeans

```
import javax.ejb.Remote;

@Remote
public interface Calculator {
    int add(int a, int b);
    int subtract(int a, int b);
}
```

```
import javax.ejb.Stateless;

@Stateless
public class CalculatorBean implements Calculator {

    @Override
    public int add(int a, int b) {
        return a + b;
    }

    @Override
    public int subtract(int a, int b) {
        return a - b;
    }
}
```

In this example, **Calculator** is the remote interface that defines the business methods (add and subtract). The **CalculatorBean** class implements this interface and is annotated with @Stateless to indicate that it is a stateless session bean.

Enterprise JavaBeans

```
import javax.ejb.EJB;

public class CalculatorClient {

    @EJB
    private Calculator calculator;

    public void performCalculation() {
        int sum = calculator.add(1, 2);
        int difference = calculator.subtract(2, 1);
    }

    public static void main(String[] args) {
        CalculatorClient client = new CalculatorClient();
        client.performCalculation();
    }
}
```

The @EJB annotation is used to inject the Calculator EJB into the calculator field.

The performCalculation method then uses the injected EJB to perform addition and subtraction operations.

Enterprise JavaBeans

“A problémák gyorsan előkerültek, ennek következtében az EJB hírneve kopni kezdett. Néhány API fejlesztő úgy gondolta, hogy az EJB API-ja sokkal összetettebb, mint amihez hozzá voltak szokva. A rengeteg ellenőrzött kivétel, igényelt interfész és a bean osztályok absztrakt osztályként való implementálása szokatlan és természetellenes volt a legtöbb programozó számára. Ismert, hogy az EJB szabvány nagyon összetett problémákat próbált kezelni. Ilyen pl. az objektum-relációs leképezés vagy a tranzakció épség. Ennek ellenére sok programozó az API-t is hasonló, vagy még nehezebbnek vélte, így alakult ki az a **széles körű felfogás, miszerint az EJB által bemutatott összetettség nem nyújt igazi hasznot.**

A cégek egymás között megegyeztek abban, hogy az EJB specifikáció elsődleges előnye, az elosztott alkalmazások tranzakciós épségének biztosítását a legtöbb üzleti alkalmazás kevésbé használja. Sokkal hasznosabbak voltak a **könnyűsúlyú keretrendszerek** által nyújtott funkciók. Példa erre a [Spring](#), vagy a [Hibernate](#)."

https://hu.wikipedia.org/wiki/Enterprise_JavaBeans

Spring Framework

- 2003-ban Rod Johnson adta ki.
- XML-ben volt konfigurálható.
- Cél a Java alapú fejlesztés előmozdítása és olyan tevezési módszerek használata, amik segítik a munkát:
 - convention over configuration,
 - dependency injection,
 - aspektusorientált programozás.
- Ugyanakkor például a függőség kezelés különböző Spring modulokhoz kihívást jelentett.

Spring Boot

- 2012: A Spring 3.1 verziójában megjelent a Java alapú konfiguráció lehetősége, ami csökkentette az XML konfiguráció szükségességét.
- 2013: Megjelent a Groovy alapú konfigurálhatóság.
- **2014: Spring Boot 1.0**
 - Convention over configuration bevezetése.
 - Spring komponensek automatikus konfigurálása.
 - Beágyazott szerverek (pl Tomcat, Jetty, stb).

Groovy

Apache Groovy is a **powerful, optionally typed and dynamic** language, with **static-typing and static compilation** capabilities, for the Java platform aimed at improving developer productivity thanks to a concise, **familiar and easy to learn syntax**.


It integrates smoothly with any Java program, and immediately delivers to your application powerful features, including scripting capabilities, **Domain-Specific Language** authoring, runtime and compile-time **meta-programming** and **functional** programming.


Spring Boot




<https://start.spring.io/>

Spring Boot







Project

☒ Gradle - Groovy
☐ Gradle - Kotlin
☐ Maven

Language

☒ Java ☐ Kotlin
☐ Groovy

Spring Boot

☐ 3.2.0 (SNAPSHOT) ☐ 3.2.0 (M3)
☐ 3.1.5 (SNAPSHOT) ☒ 3.1.4 ☐ 3.0.12 (SNAPSHOT)
☐ 3.0.11 ☐ 2.7.17 (SNAPSHOT) ☐ 2.7.16

Project Metadata

Group

Artifact

Name

Description

Package name



Packaging ☒ Jar ☐ War

Java ☐ 21 ☒ 17 ☐ 11 ☐ 8

Dependencies

ADD DEPENDENCIES... ⌘ + B

No dependency selected



GENERATE ⌘ + ↵

EXPLORE CTRL + SPACE

SHARE...

Spring Boot

web

Press ⌘ for multiple adds

Spring Web

WEB

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

Spring Reactive Web

WEB

Build reactive web applications with Spring WebFlux and Netty.

Thymeleaf

TEMPLATE ENGINES

A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

Spring Web Services

WEB

Facilitates contract-first SOAP development. Allows for the creation of flexible web services using one of the many ways to manipulate XML payloads.

WebSocket

MESSAGING

Build Servlet-based WebSocket applications with SockJS and STOMP.

Jersey

WEB




Framework for developing RESTful Web Services in Java that provides support for JAX-RS APIs.

Vaadin

WEB

A web framework that allows you to write UI in pure Java without getting bogged down in JS, HTML, and CSS.

Spring Boot



Project

- ☒ Gradle - Groovy
- ☐ Gradle - Kotlin
- ☐ Maven

Language

- ☒ Java
- ☐ Kotlin
- ☐ Groovy

Spring Boot

- ☐ 3.2.0 (SNAPSHOT)
- ☐ 3.2.0 (M3)
- ☐ 3.1.5 (SNAPSHOT)
- ☒ 3.1.4
- ☐ 3.0.12 (SNAPSHOT)
- ☐ 3.0.11
- ☐ 2.7.17 (SNAPSHOT)
- ☐ 2.7.16

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging

- ☒ Jar
- ☐ War

Java

- ☐ 21
- ☒ 17
- ☐ 11
- ☐ 8

Dependencies

ADD DEPENDENCIES... ⌘ + B

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

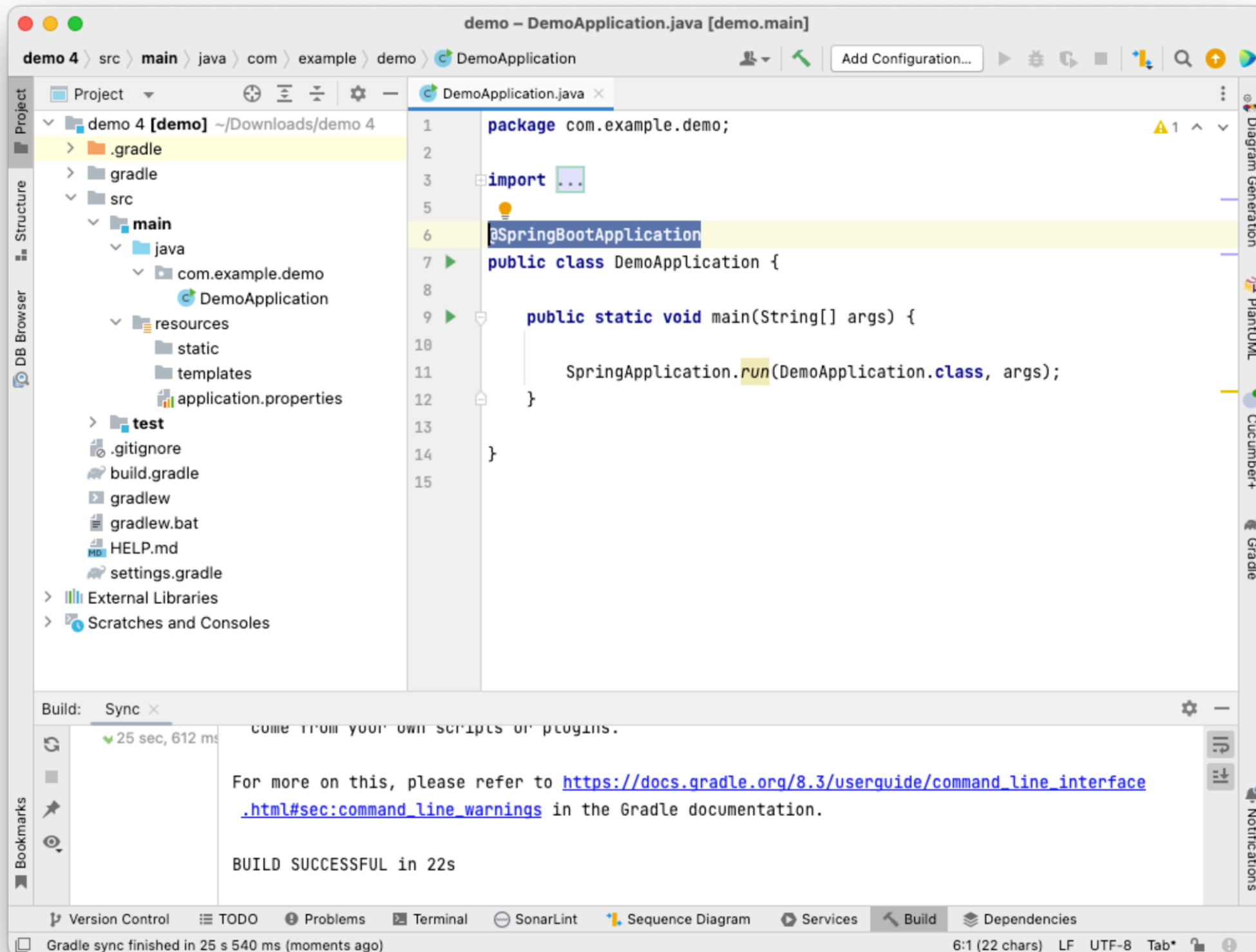


GENERATE ⌘ + ↵

EXPLORE CTRL + SPACE

SHARE...

Spring Boot



@SpringBootApplication

A következő 3 annotációt tartalmazza:

- @SpringBootConfiguration
- @EnableAutoConfiguration
- @ComponentScan

Egyéb annotációk

- Hozzunk létre egy RestApi osztályt -> Jobb kattintás a package néven, majd New / Java Class

```
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;
```

```
@RestController
```

```
public class RestApi {
```

```
    @RequestMapping("/")
```

```
    public String hello(){  
        return "Hello";  
    }
```

```
}
```

Egyéb annotációk

Bean-eket a Spring hozza létre, de a létrehozás kontrollálható annotációkkal az @Scope paramétereként:

- singleton - egyetlen példány
- prototype - mindig új példány
- request - request-enként egy példány
- session - felhasználónként egy példány

```
@Scope ( "session" )
```

Egyéb annotációk

```
@Scope("session")
public class CurrentTime {

    String currentTime =
        Calendar.getInstance().getTime().toString();

    public String getTime(){
        return currentTime;
    }
}
```

Egyéb annotációk

```
@RestController
public class RestApi {

    private CurrentTime time = new CurrentTime();

    @RequestMapping("/")
    public String hello(){
        return time.getTime();
    }
}
```

Application Context

- Inversion of Control (IoC)-hez szükséges
- Ebben tárolja a Spring a létrehozott object-eket.

```
@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) {

        ApplicationContext applicationContext =
            SpringApplication.run(DemoApplication.class, args);

        String[] beans = applicationContext.getBeanDefinitionNames();
        Arrays.sort(beans);
        for(String name : beans)
            System.out.println(name);
    }
}
```

De miért jönnek létre a Bean-ek?

Miért jönnek létre a Bean-ek?

- @SpringBootConfiguration
- @EnableAutoConfiguration
- **@ComponentScan** - keresse meg a szükséges bean-eket a jelen package-ben. (megadhatók package-ek)

@SpringBootApplication

De a CurrentTime nincs a listában!

```
applicationAvailability  
applicationTaskExecutor  
basicErrorController  
beanNameHandlerMapping  
beanNameViewResolver  
characterEncodingFilter  
conventionErrorViewResolver  
defaultServletHandlerMapping
```


@Component

```
@Scope("singleton")
@Component
public class CurrentTime {

    String currentTime =
        Calendar.getInstance().getTime().toString();

    public String getTime(){
        return currentTime;
    }
}
```

@Autowired

```
@RestController
public class RestApi {

    // private CurrentTime time =
    //     new CurrentTime();

    @Autowired
    private CurrentTime time;

    @RequestMapping("/")
    public String hello(){
        return time.getTime();
    }
}
```

@Component

- **Osztály** szintű annotáció
- Az @Component egyéb elnevezései:
 - @Controller
 - @Service
 - @Repository

De használhatjuk az @Component-et mindenhol!

@Bean

- Amennyiben nem férünk hozzá a forrásához egy osztálynak, akkor az @Component nem fog működni.
- **@Bean** segítségével **metódusokat** annotálhatunk, amik visszatérő értéke Spring bean-ként tárolódik!
 - **A bean az egy olyan objektum amit a Spring Framework kezel.**
- ComponentScan-nel nem kompatibilis, manuálisan kell példányosítani.
- Leválasztja a példányosítást az osztály definícióról, emiatt tudunk vele létrehozni külsős osztályokból Spring bean-eket.

@Configuration

- Ezzel megjelölt osztályra a Spring úgy tekint mint **bean forrásra**.
- Az @Bean-nel annotált metódusnak egy olyan példányt kell visszaadnia amit szeretnénk mint komponenst tárolni.

@Configuration

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.Date;

@Configuration
public class BeanContainer {

    @Bean
    // A Bean azonosító a metódus neve, ha nem adjuk meg az
    // annotációban.
    public Date dateBean(){
        return new Date();
    }
}
```

@Configuration

```
@SpringBootApplication
```

```
public class DemoApplication {
```

```
    public static void main(String[] args) {
```

```
        ApplicationContext applicationContext =
```

```
            SpringApplication.run(DemoApplication.class, args);
```

```
        String[] beans = applicationContext.getBeanDefinitionNames();
```

```
        Arrays.sort(beans);
```

```
        for(String name : beans)
```

```
            System.out.println(name);
```

```
        Date date = applicationContext.getBean("dateBean", Date.class);
```

```
        System.out.println(date.getTime());
```

```
    }
```

```
}
```

RESTful

RESTful

Representational State Transfer

- REST architektúra kliensekből és szerverekből áll:
 - Kliensek kérést indítanak a szerver felé,
 - a szerver kéréseket dolgoz fel.
- A kérések és válaszok erőforrás reprezentációk köré épülnek fel.
- Erőforrás bármi lehet ami megnevezhető, átadható válaszként.

RESTful

Representational State Transfer

- Bármely adott pillanatban egy kliens **vagy állapotok közötti átmenetben van, vagy "nyugalmi" állapotban.**
- A nyugalmi állapotban lévő kliens képes interakcióra a felhasználójával, de nem hoz létre terhelést és nem fogyaszt tárolót a szervereken vagy a hálózaton.
- Ha a kliens **készen áll az átmenetre egy új állapotba,** akkor **elkezd küldeni a kéréseit a szerverekhez.**
- Míg legalább egy olyan kérés van, amelyre nem érkezett válasz, a **kliens átmeneti állapotban marad.**

Hozzunk létre egy Product.java osztályt

```
public class Product {  
  
    private String name;  
    private String price;  
  
    public Product(String name, String price) {  
        this.name = name;  
        this.price = price;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public String getPrice() {  
        return price;  
    }  
  
    public void setPrice(String price) {  
        this.price = price;  
    }  
}
```

@RestController

- RESTful web szolgáltatást definiál.

```
@RestController  
public class ProductsController {  
}
```

@RequestMapping

- A REST end-point címét megadhatjuk.
- Az alapértelmezett **request method** a **GET**

```
@RestController
public class ProductsController {

    @RequestMapping("/product")
    public Product getProducts(){
        Product product = new Product("Cola", "10 EUR");
        return product;
    }
}
```

@RequestMapping

- Visszatéréskor megadhatjuk a HTTP Status Code-ot is.

```
@RestController
public class ProductsController {

    @RequestMapping("/product")
    public ResponseEntity<Product> getProducts() {
        Product product = new Product("Cola", "10 EUR");
        return new ResponseEntity<>(product, HttpStatus.OK);
    }
}
```

```
{
    "name": "Cola",
    "price": "10 EUR"
}
```

HTTP Status kódok

Status code	Meaning	When to use
200	OK	Successful operations that return data (such as GETs).
201	Created	Successful operations that create new data (such as POSTs).
204	No Content	Successful operations that do not return any data (such as DELETEs).
400	Bad Request	Bad syntax of the request, unsupported message format, data validation errors, etc.
401	Unauthorized	The user is not authenticated.
403	Forbidden	The user is authenticated, but does not have permission to access the resource or perform the operation.
404	Not Found	The user tried to access a resource or perform an operation that does not exist. This code can also be used instead of 403 if we want to hide the existence of the protected resource.
409	Conflict	Optimistic or pessimistic locking errors.
500	Internal server error	All other errors.

@RequestMapping

- Tömb visszaadása:

```
@RestController
public class ProductsController {

    @RequestMapping( "/product" )
    public ResponseEntity<List<Product>> getProducts(){
        List<Product> products = new ArrayList<Product>();
        products.add( new Product( "Cola", "10 EUR" ) );
        products.add( new Product( "Salad", "4 EUR" ) );
        return new ResponseEntity<>(products, HttpStatus.OK);
    }
}
```

```
[ {"name": "Cola", "price": "10 EUR"}, {"name": "Salad", "price": "4 EUR"} ]
```


@RequestMapping

RequestMethod.*POST*

```
@RestController
public class ProductsController {

    @RequestMapping("/product")
    public ResponseEntity<List<Product>> getProducts(){
        List<Product> products = new ArrayList<Product>();
        products.add( new Product("Cola", "10 EUR") );
        products.add( new Product("Salad", "4 EUR") );
        return new ResponseEntity<>(products, HttpStatus.OK);
    }

    @RequestMapping(value = "/product", method = RequestMethod.POST)
    public ResponseEntity<String> createProduct(@RequestBody Product product) {
        // Store in repository
        return new ResponseEntity<String>("Created", HttpStatus.CREATED);
    }
}
```

Created

PostMan próba

The screenshot displays the Postman application interface. At the top, the URL bar shows 'Keretrendszer anyag / /product POST'. The request method is set to 'POST' and the URL is 'localhost:8080/product'. The 'Body' tab is selected, showing a JSON payload:

```
{
  "name": "Sandwich",
  "price": "15 EUR"
}
```

. The response status is '201 Created' with a response time of '113 ms' and a size of '175 B'. The response body is shown in the 'Pretty' format, displaying the text 'Created'.

HTTP Keretrendszer anyag / /product POST

Save

POST localhost:8080/product Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "name": "Sandwich",
3   "price": "15 EUR"
4 }
```

Body Cookies Headers (5) Test Results

201 Created 113 ms 175 B Save as Example

Pretty Raw Preview Visualize Text

1 Created

@RequestMapping

RequestMethod.*PUT*

```
@RequestMapping(value = "/product/{name}", method = RequestMethod.PUT)
public ResponseEntity<Object> updateProduct(@PathVariable("name") String name,
    @RequestBody Product product)
{
    // Update
    return new ResponseEntity<>("Updated: "+name, HttpStatus.OK);
}
```

HTTP Keretrendszer anyag / **/product Update**

Save



PUT

localhost:8080/product/Cola

Send

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

Beautify

```
1 {  
2   ... "name": "Cola",  
3   ... "price": "20 EUR"  
4 }  
5
```

Body

Cookies

Headers (5)

Test Results



200 OK

45 ms

177 B



Save as Example



Pretty

Raw

Preview

Visualize

Text



1 Updated: Cola

Updated: Cola

@RequestMapping

RequestMethod.*DELETE*

```
@RequestMapping(value = "/product/{name}", method = RequestMethod.DELETE)
public ResponseEntity<Object> deleteProduct(@PathVariable("name") String name,
    @RequestBody Product product)
{
    // Delete
    return new ResponseEntity<>("Deleted: "+name, HttpStatus.OK);
}
```

 Keretrendszer anyag / **/product Delete**

 Save

DELETE ▾

localhost:8080/product/Cola

Send ▾

Params

Authorization

Headers (9)

Body ●

Pre-request Script

Tests

Settings

Cookies

Query Params

	Key	Value	Description	⋮ Bulk Edit
	Key	Value	Description	

Body

Cookies

Headers (5)

Test Results



200 OK

114 ms

177 B



Save as Example



Pretty

Raw

Preview

Visualize

Text ▾



1 Deleted: Cola

Annotáció annotációja

```
@GetMapping(value = "/product")  
@RequestMapping("/product")
```

```
@PostMapping(value = "/product")  
@RequestMapping(value = "/product", method = RequestMethod.POST)
```

```
@PutMapping(value = "/product/{name}")  
@RequestMapping(value = "/product/{name}", method = RequestMethod.PUT)
```

```
@DeleteMapping(value = "/product/{name}")  
@RequestMapping(value = "/product/{name}", method = RequestMethod.DELETE)
```