**Spring Beanek**

**(Spring alkalmazás építőköve; fejlesztő hozza létre de a Spring vezérli; függőségek kapcsolatok beállítása konfigurációval; POJO= egyszerű, specifikációmentes, interfacementes, elnevezési konvenció mentes; AC-n belül, AC életcikkus vezérlés; rétegekben vannak; singleton tervezési minta; ne legyenek állapot tároló atrik; main metódus; project template start.spring.io-val -> XXXApplication, @SpingBootApplication(@EnableAutoCinfiguration, SpringBootConfiguration, ComponentScan, -> Controller, @Service, @Repository)**

A **Spring alkalmazás alapvető építőkövei**; egyszerű java osztályok, amiket a **fejlesztő hoz létre**; azért spring beanek, mert a **Spring tud róla**; a **Springes keretrendszer példányosítja** őket a fejlesztő helyett

a beanek között lehetnek **függőségek**; ezek **beállítása, összekapcsolása** **a keretrendszer feladata**; hogy hogyan tegye ezt, azt valamiféle konfigurációval lehet megadni

a Spring beanek **POJOK** = semmiféle elnevezési konvenciónak nem kell megfeleljenek és nem kell interfacet implementálniuk

Az **Application Contexten belül helyezkednek el**, a konténer **vezérli az életciklusukat** (példányosítás és referencia elengedés hogy garbage collector be tudja gyűjteni, ha már nincs rá szükség)a beanek **nem ömlesztve, hanem tipikusan rétegekben vannak elhelyezve;** repositoryban a **repository beans, service beans**, **controller beabs**

alapesetben **minden bean csak egyszer jön létre** (**singleton tervezési minta**) -> egy példány jön csak létre; **ha bejön egy webes alkalmazás esetén egy http kérés, akkor ugyanez a példány kerül megszólításra;** ha **tárol valami állapotot, párhuzamossági problémák** keletkezhetnek, ezért vigyáznunk kell erre, kivédhető azzal, ha **nem definiálunk benne olyan attribútumot, ami állapotot tárol**

az **alkalmazás belépési pontja a main metódus** (start.spring.io-is legenerálás esetén létrejön egy XXXApllicaiton osztály , rajta van a @SpringBootApplication annotáció és van egy main metódusa; ezt kell elindítani

**@SpringBootApplication** egy metaannotáció

->rajta is több annotáció van

egyik az **@EnableAutoConfiguraiton,** ami alapján ha a **classpathon talál libraryt,** amihez van konfigurációja, akkor azt **automatikusan fel fogja konfigurálni** és elérhetővé teszi

egy másik a **@SpringBootConfiguration(ezen pedig a @Configuration**) -> magában ezzel az osztályban is tudunk a java kódban saját konfigurációt megadni, ha nem jók az intelligens alapértékek

**@ComponentScan** – automatikusan **felolvassa** azokat az osztályokat, **amelyeken rajta vannak bizonyos annotációk**; az egyik ezek közül**: @Component**(általános objektumok), **@Repository, @Service, @Controller**

**Controller komponensek**

**(Spring MVC komponens,; bejövő http request -> metódus hívás -> visszatérési érték kiírása http repsonsba; egyéb annotációk a Controllereken; kerülni a servlet API függőséget; kétféle Contorller használat -> html tartalom server oldali előállítása és RESTFUL WEB)**

A Spring MVC alapvető komponensei

**a felhasználótól a bejövő kéréseket értelmezzék, és választ adjanak neki vissza; bejön egy http request, a Spring ezt átalakítja metódus hívássá és a metódus visszatérési értékét kiírja a http responseba**

POJO, csak rajta van a @Controller annotáció

**sok egyéb annotációt raknak a controllerekre**; **milyen** **urlen** lehessen őket meghívni, **milyen** **kéréseket** vár, **milyen válaszokat** ad vissza kell-e ezeket **validálni**

**nem feltétlen van Servlet API** függősége bár eredendően a Spring MVC erősen támaszkodik rá; a **legjobb gyakorlat, ha kerüljük ennek a használatát és kizárólag Java SE-s osztályokat használunk Springes annotációkkal**

a **metódusok**, amelyek **megfeleltetődnek a http kéréseknek,** azok **flexibilisek**, mi magunk adhatunk nekik nevet (nincs konvenció), paraméterezése is változó lehet

**kétféleképpen használunk kontrollert**; 1**. html tartalmat az alkalmazás server oldalon állítja elő -**> a metódusok template logikai nevet adnak vissza és ez később le lesz renderelve server oldalon majd visszaadva http responseban, 2.**RESTFUL webszolgáltatás; amit visszaadok objektumot azt egy az egybe megpróbálja valahogy leképezni a http responseban,** **tipikusan valamiféle json vagy xml formátumban = valamiféle mapping eszköz használatával**

**Controller annotációk**

**(ComponentScan, @RequestMapping -> milyen urlen figyeljen; @ResponseBody -> visszatérési érték egy az egybe mappelődjön le)**

**@Controller**: hogy a **component scan** megtalálja (A Spring MVC tudjon róla, ismerje fel)

**@RequestMapping:** megadhatjuk, hogy **milyen urlen figyeljen az adott metódus**; meg lehet adni **osztály szinten** is, ilyenkor **az összes metódusra** vonatkozik; ha osztály- és metódus szinten is szerepel, akkor konkatenálódik, összeadódik**; megadható wildcard karakterekkel** (=Ant-szerű megadási mód) pl. /admin/\*.html; a http metódus megadható a method paraméterrel

**@ResponseBody** – a metódus visszatérési értékét egy template értékként, egy logikai view értékként próbálja a Spring értelmezni**, ha azt akarjuk, hogy a Spring a visszatérési értéket egy az egyben mappelje le valamilyen formátumba, alapesetben jsonbe, akkor kell ez az annotáció**

**Controller**

egy egyszerű kontroller osztály így néz ki:

@RequestMapping - / urlen figyel = ez lesz a főoldal

@ResponseBody - a metódus által visszaadott Stringet egy az egyben ki kell írni a http response-ba



**Service**



**Kapcsolatok**

**(magasabb szintű lát rá az alacsonyabbra; függőség konstrukoros Dependency Injenctionnel; egy konstruktor esetén @Autowired)**

Hogyan lehet egyik komponensből hivatkozni egy másikra?

**tipikusan a magasabb szintű lát rá az alacsonyabb szintűre** (ez egy kötelezőség, bár a Spring nem számoltat ezzel el)

**Controller csak a Servicet látja, a Service csak a Repositorykat látja**; ill. a maga szintjén lévőket is (egy service egy másik sercitet lát) **csak fölfelé ne legyen függőség**

**a függőségekhez Dependency Injectionnel lehet hozzáférni;** **3 fajta** van A Sringben a beaneknél:

**attribútum, konstruktor és metódus(setter)** alapján = deklarálunk egy attribútumot, ami értéket közvetlen ezeken keresztül adunk

a **legjobb gyakorlat**, hogy a beanek működéséhez szükséges kötelező függőségeket **konstruktor injectionben,** ami pedig opcionális (anélkül is tud működni, pl. null is lehet)

**Ha csak egy konstruktor van, a Spring automatikusan megpróbálja a dependency injectiont,** ha **több** konstruktor van, **melyik konstruktor kerüljön meghívásra**? ilyenkor kell rátenni az **@Autowired** annotációt

**Függőség a Controllerben**

= controller hogyan függ egy service-en?

**a kontrollerünkben a servicet egy atriként definiáljuk**

majd kell egy konstruktor, ami paraméterül várja a servicet

**a Spring tudni fogja, hogy a Controller lepéldányosításához szükség van a service lepéldányosítására is és a sorrendet is tudni fogja**



**Spring Beanek – gyakorlat**

**@RestController annotációval** a Spring **automatikusan felderíti** az osztályt**(spring beanként fogja kezelni** = ő **példányosítja** és **felel a kapcsolataiért**, **vezérli az életciklusukat**) mert az EmployeesApllicaiton osztályon pedig rajta van a **@SpringBootApplication annotáció** -> erre CTRL-el ráklikk -> megjelenik a forráskódja -> download sources link -> látszik, hogy ez egy Metaannotáció; van benne pl. @SpringBootConfiguration,@Enableautonfiguration stb.

1.HelloService osztály létrehozása -> ő fogja előállítani az üdvözlő üzit

@Service annotáció rája

bele egy String sayHello() metódus, „…(service)..”.now()-al

2.Controllerből a **hívást delegáljuk a service felé**

Controllerbe HelloService ati

sayhello() a controllerbe is; delegálás servicenek; metódusra @GetMapping(”/”) annotáció

a **service atri értéke** hogy inicializálódik? -> **konstruktor injection**, public HelloController, már ide az **@Autowired sem kell, mert csak egy konstruktor van**

**3.** futtatás, böngészőbe localhost:8080, frissítés

**Kérdések**

*Mik azok a Spring beanek? Ki implementálja és ki vezérli?*

*Mit jelenet az, hogy a spring beanek POJOk?*

*Miért felel az AC?*

*Mi az a singleton tervezési minta a beanekre vonatkoztatva?*

*Mikor keletkezhetnek párhuzamossági problémák?*

*Hol és mi lesz a springbotos alkalmazás bleépési pontja?*

*Milyen annotációkat tartalmaz a @SpringBootApplication metaannotáció és miért felelősek?*

*Milyen osztályokat olvas fle automaitkusan a @ComponentScan?*

*Miért felelnek a Controller komponensek?*

*Miért felel a @RequestEMaping annotáció és hol lehet megadni?*

*Miért felel a @ResponseBody annotáció? Hogy érem el egy annotációval kiváltsam a @Controller és a @ResponseBody annotációkat?*

*A beanek közötti kapcsolatokat hogyan definiáljuk? Milyen fajtái vannak?*

*A komponensek tipikusan(jó gyakorlat szeirnt) melyik másik komponenesekre látnak rá?*

*Több konstruktor esetén mivle jelöljük, hogy melyiket használja a Spring a DI-re?*

*A Cntroller hogyan függ egy Service-en?*