|  |  |
| --- | --- |
| **SPRING CORE - CONCEPTS – THEORY - PPT** | |
|  |  |
| * Spring boot uses spring behind the scenes, has embedded server – tomcat. * Can be run standalone using – jar file * Deployed using war file – bunch of jar files | Inversion of control –   * Giving the control of creating and managing object. * injecting object dependencies to spring. |
| Maven –   * Based on dependencies – maven will download the jar files. (spring, hibernate, apache, JSON). * Build management and dependencies * Jar files - Local repo, remote repo. * POM.xml – * Config file of pjt   + GAV- Project name, version etc   + Dependencies - List of projects depending on – spring, hibernate…etc – GAV of other projects.   + Custom tasks to run - Junit reports. | Dependency Injection –   * Constructor – required dependencies - @Autowired on constructor * Setter – optional dependencies - @Autowired on setter * Field – not recommended – makes code harder for unit testing – why ?,how?.   AutoWiring –   * Spring will look for a class that matches by type – class or interface and it is injected automatically. |
| File structure –   * Src/main/java – java source code * Src/main/resources – properties / config files * Src/main/webapp – JSP files, web.xml, xml files, web config files * Src/test – unit testing code * Target – compiled code - .class files * Src/main/resources/static – html , css , js , images ,……etc * Src/main/resources/static/templates – * Management.endpoints.web.exposure.include = health , info | @Qualifier –   * In controller class, in constructor argument - @Qualifier(“name of the class , bean which needs to be picked up”). * Mark it as @Auto wired * Call it in one of the controller method * Higher priority , more specific than @primary * Along with dependency injection – class name – first character lowercase   @Primary –   * In the @component class * Just call the method in the controller class * The primary class object is called for it’s behaviour |
| Application Properties –   * Variable values   Dependencies   * Spring-boot-starter-web * A collection of maven dependencies grouped together * Dev tools – auto re-start the application * Actuator – application health, metrics – by exposing end points - /health, /info ,/beams , /mappings , ….etc in app.properties file. * Starter-security – app.properties – spring.security.user.name=scott * Spring.security.user.password=tiger * To secure endpoints * Exclude end points | Lazy Initialization – on the class @Lazy   * By default when the application starts all beans are initialized – all objects are created of @Component class * Here a bean will be initialized only when required for dependency injection, explicitly requested. * Is added on a class * Injecting means –the object is injected to controller and then the objects are called for their state / behaviour. * In app.properties – spring.main.lazy-initialization=true – lazy initialize all beans. * When end point is accessed – spring will determine the dependencies – initialize the dependencies and then creates instance of rest controller and in jects the bean. |
| Annotations -  @Value (${coach.name})  @Component – marks the class as spring bean (java object managed by spring)   * Makes bean available for dependency injection * Scan java classes, automatically register the beans in spring container.   @Autowired – in a controller   * On a constructor * Give the interface / class reference as arg * Assign it to a local variable * Now since interface is implemented by a class marked as @Component – available for dependency injection – it is automatically injected by the local variable   @RestController –   * Using local variable call the method – here automatically the object picked up is only one – marked as @Component.   @SpringBootApplication –   * Auto configuration, component scanning * @EnableAutoConfiguration – auto – configuration support * @ComponentScan – component scanning of current, sub packages, * @Configuration – register extra beans with @Bean * Creates application context, registers all beans, starts tomcat. | Bean scopes –   * Singleton – only one instance / object created. * This one bean will be injected everywhere – all dependencies will access this bean. * In the @component class we can mention @Scope(ConfigurableBeanFactory.SCOPE\_SINGLETON) * PROTOTYPE – new bean for each container request * Request – new for new http request * Session – new for new session * Application – Web app servlet context   Bean LifeCycle –   * Custom code during bean initialization * Custom code during bean destruction * Inside @Component class * Annotate for a method - @PostConstruct * This executes before a bean is initialized * @PreDestroy – executed before object is de-referenced   Configuration –   * @Configuration – on a class * @Bean on a method – returns the object of the class * Method name = bean ID * @Qualifier – give the method name, do not give the @configuration class name * To make third party class available for spring framework * Use third party class as bean * Inject this bean into controller and use the behaviours of the bean. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**HIBERNATE - JPA**

|  |  |
| --- | --- |
| Hibernate -   * Framework to persist / save java objects into database * ORM - Mapping between class and table * To minimize jdbc code * Write methods – instead of queries * Implements JPA | JPA – Jakarta persistence API   * API for ORM * Defines interfaces – * You need to give implementation * entityManager |
| * Uses jdbc for all database communications * @GeneratedValue(strategy=GenerationType.IDENTITY) | Entity Manager methods – to create queries   * Persist(object) – to save * Find (entity class, id(pk)) – to find * Merge(object) * createQuery(“query”,student.class) * TypedQuery |
| Entity class –   * Java class @Entity which is mapped to a database table – fields to columns * A public no-arg constr is a must | ID Generation Strategies   * AUTO * IDENTITY * SEQUENCE * TABLE * UUID |
| DAO –   * Needs JPA entity manager for saving / retrieving entities * Jpa entity manager needs a data source * Spring creates – Jpa entity manager, data source using app.properties * Inject EM into DAO. | * Entity manager object is created * On that call persist(object) * In save impl – paste the above line. * DAO-save-EM-persist * DAO is calling EM object * @Transactional |
|  | |
| STEPS –   * Create entity * Define DAO interface * Define DAO implementation – inject EM * Update main app | * @Repository – for registering DAO implementations, |
| JPQL –   * Uses entity name, entity fields |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **CODING TASKS** | |
| Actuator | management.endpoints.web.exposure.include=health,info  management.info.env.enabled=true |
| Spring security | spring.security.user.name=test  spring.security.user.password=bro |
|  |  |
|  |  |
|  |  |