Spring Boot

Date: / /
Page No.

* Spring Container

The Spring container is core of Spring framework. The container use for creating the object and configuring them.

Spring container is also called as IOC (Inversion of Control)

* Types of Spring container

1) Bean Factory
2) Application context

1) Application Context

Application context is an interface which is implemented by these 3 class

i) ClasspathXML Application Context → ~~used when xml file~~
ii) FileSystemXML ApplicationContext
iii) WebXML ApplicationContext.

* Spring Bean

Object created by spring container is called spring beans

Ways to write spring beans :- i) using xml file
ii) annotations
iii) ~~Java configuration~~

4 jars needed to run simpal spring application

1) Spring bean 2) Spring core 3) spring context
4) Spring expression

Date: / /
Page No.

MCQ Whenever we create Source folder files in that path will automatically added in classpath

<web-app> <web-app> is root element of web.xml

<persistance> is root tag of persistant.xml

* Creation of bean using XML

→ We will add .xml file in resource folder ∴ it will be added in classpath and that's why we can use ClasspathXMLApplicationContext

if .xml file is somewhere else in another directory then use FileSystemXMLApplicationContext

Root Tag of XML file is <beans> </beans>

```
<bean id="employee" class="com.met.model.Employee">
</bean>
```

* getBean()

```
Employee emp = context.getBean("employee1", employee.class);
```

To get data from spring bean.

* Scope of Spring bean

1) Singleton

per bean Id there will be 1 object only

Interview Question

Q. How we can write emp and emp2 but

Ans. It is spring bean level Singelton

bydefault Scope is Singelton

eg. SessionFactory

Framework is used so that developer can only focus on business logic.

Date: / /
Page No.

When this object gets created during initialization and only one time

2) Prototype

	Singelton	Prototype
When?	during initialization	Whenever we require (When we call getbeans()
how many times?	one	~~how~~ no. of times you specify
eg.	SessionFactory	Session (hybernate)

* lazy-init = true

If we want Singleton object but it should be created at runtime not on initialization then use "lazy-init = true" property

To have all Spring beans as lazy initialization use default-lazy-init = "true"

* ~~Factory~~ Builder Pattern

We use private constructor is ~~factory~~ Builder pattern. But spring can call private constructor using reflection internally.

But we don't want that to happen so whenever we implement ~~Factory~~ builder pattern we use static ~~class~~ method so mention that method while creating bean

<bean id = "stage1" class = "com.met.model.Stage"
factory-method = "getStage"> </bean>

always use Interfaces instead of class so that code will be more maintainable & expandable

Date / /
Page No.

It we want only one value object everytime we call it

whenever we create get or set using getters & setters default constructor is compulsory else it might give null pointer exception

```
Private static Stage stage = null

public static stage getstage () {

if (stage == null) {
        stage = new stage();}
        return stage;
}
```

* Dependancy Injection

Dependancy Injection is a process where in spring beans defines dependancy with other bean via constructor arguments, property or via factory method.

At run time Spring container will create beans and inject dependant beans.
Since Spring container is taking control of it We call it as <u>inversion of Control (IOC)</u>

Dependancy injection is achieved by IOC
In IOC SC (spring container) will create bean and inject in dependant bean at runtime

DI provides loose coupling
So that we don't need to change source code only need to configure xml files.

constructor-arg>

To use DI
i) configure the referred class in xml
ii) use ref = "dependant bean" to add referred class in xml file.

* Wiring
To identify dependant bean.

Manual wiring
using ref = "dependant bean"

Auto wiring
~~Create connection~~ " auto-wire = byType" use this
To enable autowire

How Spring autowires
1) First go in the class "METSol"

2) How many properties / data member is there in class

3) Interface or class
1) If interface
a) Find class implementing interface (JavaTrainer)
b) Do we have any Spring bean for that class

* injection of depedant bean via property is called property injection autowire = "byType"

* injection of depeandant bean via ~~property~~ constructor, called constructor injection
autowire : "~~default~~ constructor"

Date: / /
Page No.

1) Find class (Training workshop)

2) Since autowire = constructor it will find how many constructor are in that class

think* 3) Then start with max no. of argument constructor

before new 4) it will find Type of argument.

keyword 5) Find class implementing that interface (if it is interface)

in spring bean

6) check if bean is there if it is there it will inject.

When we use default constructor bean will be created but injection doesn't happen

When we use autowiring using constructor it should have constructor with argument whoes spring bean exist otherwise it will throw exception.

If we only have default constructor it will throw null pointer exception dependency injection will not happen

So never use autowiring by constructor use autowiring by property instead.

* Autowire By Name
- works on name of property
- Should be same as name of Spring id.

* Primary = True
If object can only accepts single bean but if we have two bean and you want

to give priority to one bean mark it as primary = true"

otherwise create ~~array~~ list of object and then bean will be able to accept that list of object.

If the name of bean matches with name of object then also ambiguity will be resolved

* Creating Spring bean with annotations

@component = creates bean

@autowired = inject bean automatically
you can write it on constructor, type and also put on Setters

To make annotations ~~work~~ work use
context: component-scan base-package --> It will scan all classes mark with @component annotation from specified package.

* @Autowired (required = false)

Inject bean if it exist if it doesn't exist please don't throw an exception.

but still we will have to keep configuration in XML file even if we are using annotations.

* Creating spring bean with java configurations

- Remove configurations from XML file

Java configuration

1) @Configuration ⟶ Beans
2) @Bean ⟶ bean
3) @ component Scan ⟶ <context : componet >

Date: / /
Page No.

to give priority to one bean mark it as "primary = true"

If the name of bean matches with name of object then also ambiguity will be resolved

otherwise create ~~array~~ list of object and then bean will be able to accept that list of object.

* Creating Spring bean with annotations

@component = creates bean

@autowired = inject bean automatically
you can write it on constructor, type and also put on Setters

To make annotations ~~work~~ work use
context: component-scan base-package --> It will scan all classes mark with @component annotation from specified package.

* @Autowired (required = false)

Inject bean if it exist if it doesn't exist please don't throw an exception.

but still we will have to keep configuration in XML file even if we are using annotations.

* Creating spring bean with java configurations

- Remove configurations from XML file

Java Configuration

1) @Configuration ⟶ Beans
2) @Bean ⟶ bean
3) @ component scan ⟶ <context : componet >

Spring container will call methods under @Bean.

Date: / /
Page No.

@Bean :- bean can be written only if it has @configuration

Annotation
@ Value :- is used to get value

Properties
We use properties file because we dont want Values to be hardcoded.
Properties file will have
Jt1.name ="ajay"
Jt1. Subject = {"Generics", "OOPs", "Reflection"}

@configuration → Source of Beans
@Bean → treated as spring bean
@value → expression language

Expression language
@value(${jt1.name})
private String trainer1name

@Bean
@value
@componenScan } Java configuration

@propertySource → to read property file

@primary → "primary = true"
@ lazy → " lazy-init = "true"
@scope → " prototype / singleton " } bean attribute

Date / /
Page No.

@Bean can be written only if class has @Configuration annotation.

ApplicationContext ac = new AnnotationConfigurationApplicationContext("TrainerConfiguration.~~properties~~");

To go and search Configuration of annotation type (instead of XML) in TrainingConfiguration.java

we are maintaing ~~Separating~~ separate properties file ∴ we use

@propertySource("app.properties"),

@value("#{${jt1.subjects}}")

↓

To treat it as single value instead of treating whole thing as a string cuz jt1.subject is array.

Date: / /
Page No.

* ASPECT ORIENTED PROGRAMMING

* Boiler Plate code

1) code gets duplicated across multiple file (Employee DAO, Account DAO, Student DAO)

2) No contribution to business logic / cuts developer from business logic

* Aspect

add boiler plate code in single file is called aspect.

* @ aspect

Mark aspect class with @ aspect annotations. So spring can recognize that this class is going to modularize ~~across~~ cross cutting concerns.

* JoinPoint

All methods executing in a program.

* weaving

@ EnableAutoProxy

merge both classes i.e business logic & aspect code.

* Pointcut

Method which has ~~boiler plate~~ business logic code

* Target object

object whoes proxy you have created.

(participants) is proxy

* Target method
which has business logic

Date: / /
Page No

If trainer is implementing interface then proxy will be created using JVM Proxy
Spring container autogenerate this code

```
class Proxy implements Trainer {
        Trainer trainer;
    public void train() {
            @ Before training      // Before Boilerplate
            @ trainer.train()
            @ After training       // after Boilerplate
```

training has boilerplate code

If class doesn't implement an interface then proxy is created using CGLIB Proxy

```
class CGLibProxy extends JavaDotnetTrainer {
    train() {
        try {
            Before B.L
            => B.L Super.train();
            After B.L
        }
        catch() {
        }
        finally {
        }
    }
```

If class is declared as final then proxy will not be created.

1) Spring supports only method execution in AOP
2) Spring supports only 15% of AOP which is 90% of the time used.

4) If you want whole Aspect oriented Programming use
Aspect J.

@pointcut (* * . train(. .));
1 2 3 4

*1) → can have any return type
* 2) → can be present in any class
train 3) → Business logic
.. 4) → can have any no. of argument.

If we want all methods from class to be ~~business~~ treated as business logic use
~~@pointcut (* * . train~~
@pointcut (* * . * (. .)));

~~@Before~~
@pointcut(" execution (* * :train (. .))")
public void training () { } //Boilerplate code

@Before ("training()" boilerPlate()),
// logic to be executed before Business logic
}

@After ("training()" boilerPlate()); {
logic to be execute After Business logic
}

@AfterThrowing ("training()" boilerplate()), {
// logic to be executed if ~~boiler plat~~ Busnelogi throwes exception
}

Date: / /
Page No.

@ AfterReturning ("training()") {
// executed even if Business logic throws exception exactly like Finally
}

@around ("training()")
public void aroundTraining (ProceedingJoinPoint pjp){
try {

pjp.proceed();

}

catch (Throwable e) {

}

Finally {

}

proceedjoinPoint
it lets you run your joint Point (business logic methods)

Date: / /
Page No.

MVC

* org.springframework.web.servlet.DispatcherServlet

Spring provides dispatcherServlet and this is fully qualified name

Server side Rendering

* FrontController Design Pattern

Server side redirection

Each and every request must be intercepted from this controller and this ~~Dispatchers~~ DispatcherServlet implements FrontController.

By default it dispatcher servlet used forward dispatcher it doesn't use sendRedirect

dispatcher Servlet has to be mapped using <servlet> in web.xml

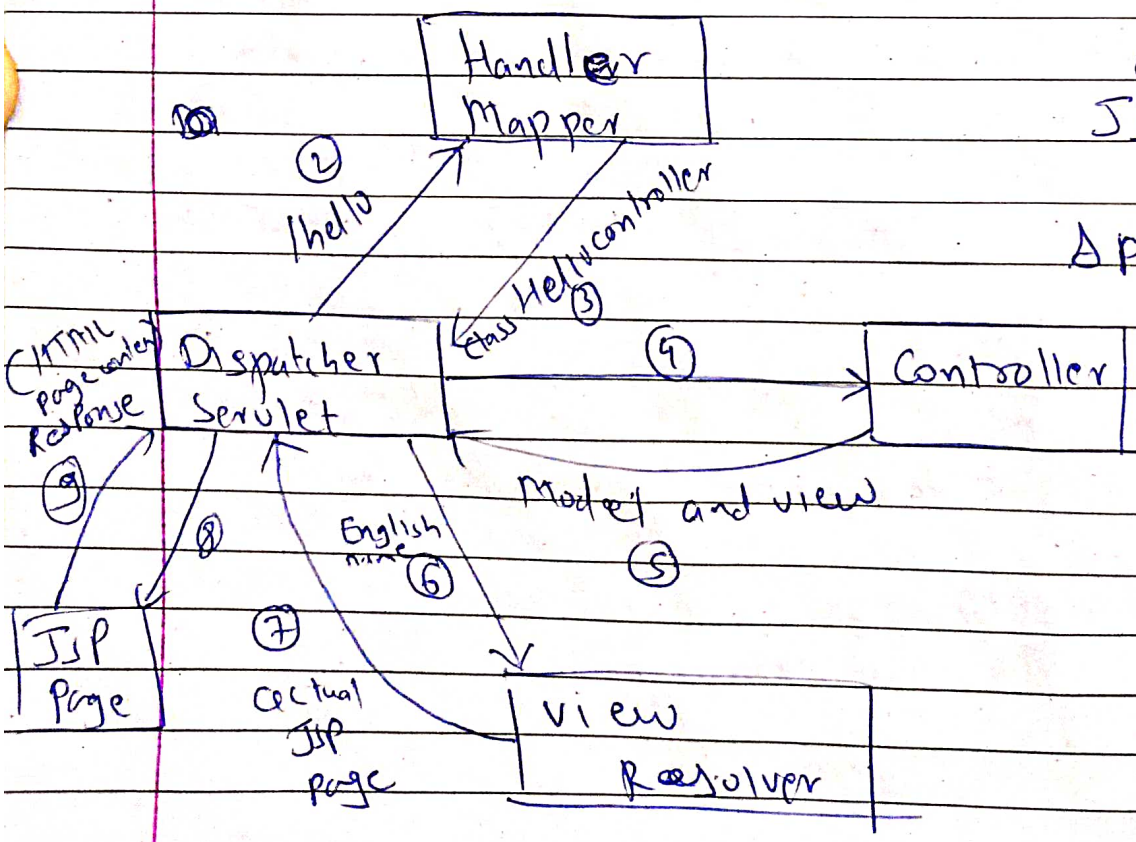
Steps :-

1) add jar

2) create new project as dynamic web project

3) Request comes by URL as http://localhost:8085/SpringMVCTest/mvc/

① http://localhost:8085/springMVCTest/mvc/hello



Other than JSP page everything is a spring bean.

Date: / /
Page No.

* English name refers to URL given in web-mvc-servlet.xml

```
<bean class="org. ....
    <property name="prefix">
        <Value> /WEB-INF/Pages/ </value>
    </property>
    <property name="suffix">
        <value>.jsp</value>
    </property>
</bean>
```

Dispatcher servlet will read contextRootname-servlet-servlet.xml file" and create calls controller

@RequestMapping

It is used to map web requests onto specific handler classes and/or handler methods

~~@Get~~ @GetMapping

it acts as shortcut with @RequestMapping(method = RequestMethod.GET)

it maps HTTPGET requests onto specific handler methods.

@RequestParam

to extract Query parameters, form parameters and even files from request.

we use @GetMapping("queryParam") here

When need to return single object use path variable

@PathVariable

@PathVariable is a spring annotation which indicates that a method parameter should be bound to a URI template variable

So if give URL as "path/jack/22" then and annotation is given as

here context root name is web-mvc thats why we have web-mvc-servlet.xml in project.

Date: / /
Page No.

@GetMapping("pathVar/{name}/{age}")
then that name & age will be taken as
@PathVariable string name, @PathVariable int age

mv.setViewName("view")
It sets the name of view to be resolved by configured ViewResolver ("English name") from web-mvc servlet.xml

client side Redirection	Server side Rendering
directly JSP page is given to browser and browser will execute JSP	~~Redirect~~ only HTML page will given to browser

get method	post method
get method have limit to pass parameter cuz it is written in header of HTTP	It doesn't have limit cuz parameter are written in body of HTTP

<load on startup> 1 </load on startup>
Whenever tomcat is initiallize Dispatcher servlet will initialize init() of dispatcher servlet so that loading time will be reduce.
This is written in web.xml.

<mvc:annotation driven>
This is written in webmvc-servlet.xml
It is used for enabling the Spring mvc components with its default configuration. If you don't add this tag you will have to manually add beans for component like HandlerAdapter, HandlerMapping etc.
It scans @GetMapping

Date: / /
Page No.

Handshake

If we are taking values from HTML form and we need to create object from it then we don't want to write ~~requ.~~ request.getParameter() many times we want that JSP page should populate the object ~~and pass~~ It is done by "handshake"

Handshake between JSP and controller

controller will give default object to JSP and JSP will populate that object.

then this populated object you can either print or again give to the JSP which will be another handshake

@Controller tells that particular class serves role of controller

@GetMapping :- get method
@PostMapping :- post method

ModelAndView.addObject("objname", object)
to populate object

@ModelAttribute
to get populated object from JSP

Date: / /
Page No.

Controller layer	Service layer	DAO layer
1) user input can be taken in this layer	other service such as validation, authorization can be done	Saves data into database
2) @Controller	@Service	@repository
3) It is dependant on ~~Conts~~ service layer	It is dependant on DAO layer	

* DataSource

It is an alternative to DriverManager Facility a Datasource object is preferred means of getting connection

* Template

If you want to use your own SQL queries use JDBC Template
1) need to configure it in webmvc-servlet.xml

* RowMapper

to map each row with Bean of employee class (instance of model) (Converts rows into objects)

* @Qulifier

In this code if want to use two databases then use @Qulifier over one of them to resolve ambiguity
This annotation defines a preference when multiple beans of same type are present.

Date / /
Page No

@autowired
is applied because of the dependancy between Controller, Service & DAO layer

* Transactions

Programmatic transaction management	Declarative transaction management
1) allows you to manage the transaction with help of programming in your source code	allows you to manage transact" with the help of configuration instead of hardcoding in source code
2) difficult to maintain	2) easier to maintain

@Transactional
In declarative approach we annotate the methods with @transactional annotation.

It lets you make use of attribute such as rollback, norollback, propagation. So that you can configure your transactions.

@Transactional has to be present in service layer.

@Transactional (rollback for = Exception.class)
should rollback for all type of exception.

@Exception Handler
@ExceptionHandler is an annotation used to handle the specific exceptions and sending the custom responses to the client.

Saving data (object) (Populated Object) taken from JSP into the database

1) using JDBC
2) using DataSource
3) using Jdbc Template
4) using hybernate

Hybernate
To use hybernate configure it under webmvc-servlet.xml
then create session using sessionFactory open Session and begin Transaction and save employee.

Difference between getCurrentSession() and openSession()
open session → always creates a new Session
getcurrentSession → check whether an existing Session is present or not. If not present it throw error.

@Transactional
@Transactional internally calls @around and generally in @around we write boilerplate code ~~opensession~~
∴ SessionFactory and opensession is boilerplate code ∴ it is written in @around.

∴ In @Transactional you always calls getcurrentSession() because if @around is called internally the session will be created and so you don't

Date / /
Page No.

have to open new session.

If we write openSession() in @Transactional then it will give error

To use transaction write configure <tx:annotation- Transaction Manager in webmvc-servlet.xml

<tx : annotation-driven transaction-manger = hibernat... />
If we write this we are saying transactions are annotation driven

@Transactional (rollbackFor = Exception.class)
i.e rollback for every kind of exception

—||— (....., noRollbackFor)
If this particular exception comes don't rollback (eg. if nullpointer exception comes)

—||— (readonly)
If you want to fire select query.

—||— (timeout = 5)
For a 5 ~~second~~ record insertion, ~~minutime~~ minimum time to be alloted for single record.

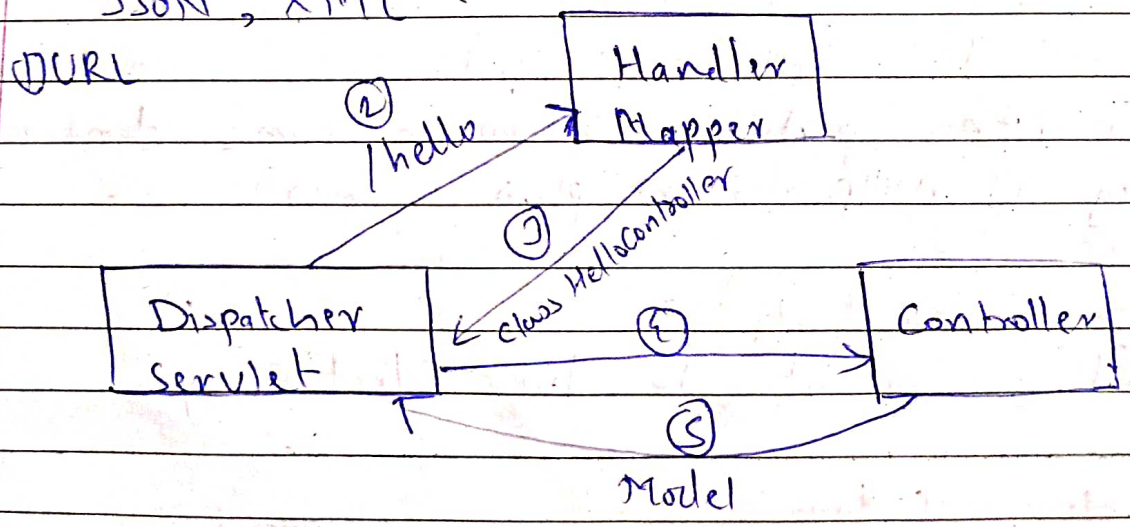
* Can also make a whole class @Transactional but think twice before you make whole class Transactional cuz it may add overhead.
* In MVC sessionId is created when we use get or post

* REST Service

MVC is called as view Technology cuz without view it will not ~~create~~ MVC will not work

Transfering state of an object to an representational form

This representational form can be JSON, XML.



Remove ~~to~~ view Resolver and ~~view~~ JSP from MVC then it will be REST Service

@ResponseBody

To convert view technology (MVC) To rest ~~services~~ use this annotation.

Date / /
Page No.

@RestController

To ~~Sta~~ indicate that this controller is going to return rest Service

@ Controller + @responseBody = @RESTController

3 ways to write REST
1) @controller + @responseBody
2) @RESTController
3) ResponseEntity

all shows that this method is going to written REST Service.

PUT and POST

PUT is idempotent and POST is not.
idempotent means ~~S~~ IF you place order using POST and refresh page it should not place another order.

Get and PUT are idempotent.

JAXSON API

Convert Java obj to json string (Serialization) and json string to java object (Deserialization)

* Produces ~~Cattri~~ It is attribute
Send JSON String

produces can have both MediaType ~~JSON~~ Application JSONvalue and MediaType ~~JSON~~ Application XML-VALUE

@RequestBody
@PostMapping (
Produces = MediaType.Application JSON-VALUE
Consumes = MediaType.Application JSON.VALUE.

can have produces and consumes on the same method

@CrossOrigin
will allow AJAX call from different different port numbers. for eg Angular can use your REST web service running on different servers/port-no

* RootWebApplicationContext
Will have common services which can be used by different apps.
(intialize common Spring beans)

ApplicationContext.Xml is file which will be read by ~~Spring container~~ Servlet Context Listener.

RootWebApplicationContex

rest-s web-mvc

common been scope so both container will share it.

* Way to write <ref= " " > in configuration
1) ~~OSSO~~ use callback
2) @autowired
3)

Date / /
Page No.

* Spring Boot

It is Bootstrap your Spring framework application
bootstrap means to quickly start application

1) Go to Spring initializer
2) download zip file
3) add zip file ~~to project~~ to existing maven project

It will give you all ~~required~~ dependant jars that are required to startup your application
add this in projnamelpom.xml file
<dependancy>
<groupId> org.springframework.boot </groupId>
<artifactId> spring-boot-starter-web </artifactId>

When we create project
1) application.properties file will get created and
2) springbootwebapplication.~~jaa~~java file will be automatically created it will have main method.

1) uses apache Tomcat as default embedded container
2) spring boot will do the configuration of dispatcher Servlet.
means creation of Dispatcher Servlet, xml file for bean and setting up ~~tope~~ tomcat container (will be handlled by spring boot developer don't need to worry about it

Spring says we follow Convention over configuration

Date: / /
Page No.

for eg we are creating JDBC or hibernate application then JDBC requires datasource, JDBC Template etc. and hibernate requires SessionFactory so these are convention so Spring provides this all so developer doesn't need to do configuration

@SpringBootApplication => @Configuration (creating beans) + @ComponentScan + @EnableAutoConfiguration

here @component Scan will only work If Controller is in same package as @SpringBootApplication is written

∴ Controller classes should be in package which has class which contain @SpringBootApplication

@EnableAutoConfiguration

It scans classpath components and registering the beans These beans will be added (Configured) like dispatcher servlet, tomcat, jdbc, you can see all this configured things into Maven dependancies → spring-boot-autoconfigured-2.6.4.jar

Creates FatJar

Fat jar is jar insider jar

Remote Repository

It is require if company has bought license product eg oracle the if ~~you~~ company wants to share it will put all jars in remote repository

Date: / /
Page No.

If you don't have remote repository

~~add~~ download that jar file &
~~and~~ add ~~pom~~ dependancies in springbootweb/pom.xml file

· 3 vendors or implementors of dataSource
1) Hikari → Springbot version 2.x
2) Tomcat → Springboot version 1.x
3) DBCP

Springboot is opinionated framework
It take opinion from developer and uses implementor according to opinion eg in 1.x it was using Tomcat but not its using hikari

JdbcTemplate, SessionFactory, dataSource are beans will be created by Spring and have @autowired on it.

details about database like password, username is given in application.properties file

SpringBootApplication(exclude = "dataSource.class")

here you can have different classes whichever you want to exclude

To exclude ~~classes~~ from auto configuration means don't create spring bean for that class

Instead of that excluded class you can give your own bean

Date: / /
Page No.

Actuator

For monitoring of application we use actuator
it checks 1) health of application :- whether you are optimizing resources or not there should not be load on application
2) matrix :- at what time applicat^n started, Session created, GC Started, session ended

Add actuator <dependancy> ... </dependancy>
in springboot2web/pom.xml
and specify port no. to run actuator in application.properties

Features of Springboot

1) Stand alone → it doesnot require external tomcat configuration for running app ie it has embedded tomcat

2) production grade - actuator, metrics, health checks

3) just run → just run app using java, jar, jarname
no need of classpath

4) opiniated + follows convention over configuration
(Autoconfiguration.

lecture03

To Implement JPA

1) Take hibernate from Spring initializer
2) Import it on IDE

3) In ~~proper~~ application.properties File
1) Change port no

Date / /
Page No.

2) context path of application
3) Configuration for connecting to dB

If tommorow we remove hybernate vendor and use some another vendor and if we have used session Factory then app might ~~to~~ crashed ~~then~~ that's why use Standard JPA
EntityManger and EntityManager Factory

@ PersistanteContext
Spring will automatically create bean of entityManger

@ Entity
@ Table (name = "Employee").
add it in PoJo class

Spring.Jpa.hibernate.naming.physical-strategy = org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
Whatever name we have given in PoJo use same name for select Query don't convert name into select Query

* Starter Dependancy
Whenever you work with different vendors like spring boot and hibernate in same project then migh be version conflict use starter dependancy to remove that

@Repository
Extent one interface from JPARepository Interface
then you don't have to write Entity Manager

Date: / /
Page No.

save(emp), get(emp) again ~~&~~ again and use it as @Autowired

Repository (I)
↓
CRUDRepository (I)
↓
PagingandSorting Repository (I)
↓
JPARepository → provides ORM (I)