Advance - Java.

Advance Java:-

* What is inversion of control?
  * giving our flow control to framework to maintain the flow is called inversion of control

* Application context interface represent the IOC container
* Web Container: it will create all beans at initialisation and are kept ready for use.
* Springboot features
  1. Starter (spring-boot-starter-JPA/test/maydevtools.
  2. autoConfiguration → injecting required beans as per starter metioned
  3. Actuator → provide facility to ~~beb debug~~ monitor the app
  4. Security
  5. logging.

* Spring Vs Springboot:-

Spring	Spring boot
1. Spring framework widely used for Java EE framework, for building applicatn	2. widely used for developing Rest APIs
2. primary feature is Dependancy Injection	3. primary feature is auto configuration
3. Developer manually define dependencies in pom.xml	3. here we add starter in pom.xml and all related dependancies are fetched automatically
4. huge boiler code is involved	4. Boiler plate code is reduced
5. require explicit setup for testing	5. has inbuild support such as tomcat or Jetty.

Spring boot using maven

16

We inherit

```
<parent>
    <groupId> org.springframework.boot
    <artifactId> spring-boot-starter-parent </>
    <Version> 2.4.0 RELEASE </>
</parent>
```

and declare starter dependancies.

Spring Initialiser

Website to create springboot project by adding simple starters -

→ this creates skeleton for the project

→ Just code and run by extracting In IDE

Spring-boot-starters Available

org.Springframework.boot

→ Spring-boot-starter → core starter (auto Configuration support, logging, Yaml)

→ spring-boot-starter-data-jpa → to use spring data JPA with hibernate

→ spring-boot-starter-web → for using web application using spring MVC, RESTFul.

→ spring-boot-starter-security → spring security.

→ spring-boot-starter-test → Junit

Q How to disable specific auto configuration?

```
@EnableAutoConfiguration(exclude = SpringSecurity.class)
@SpringBootApplication(exclude = SpringSecurity.class)
```

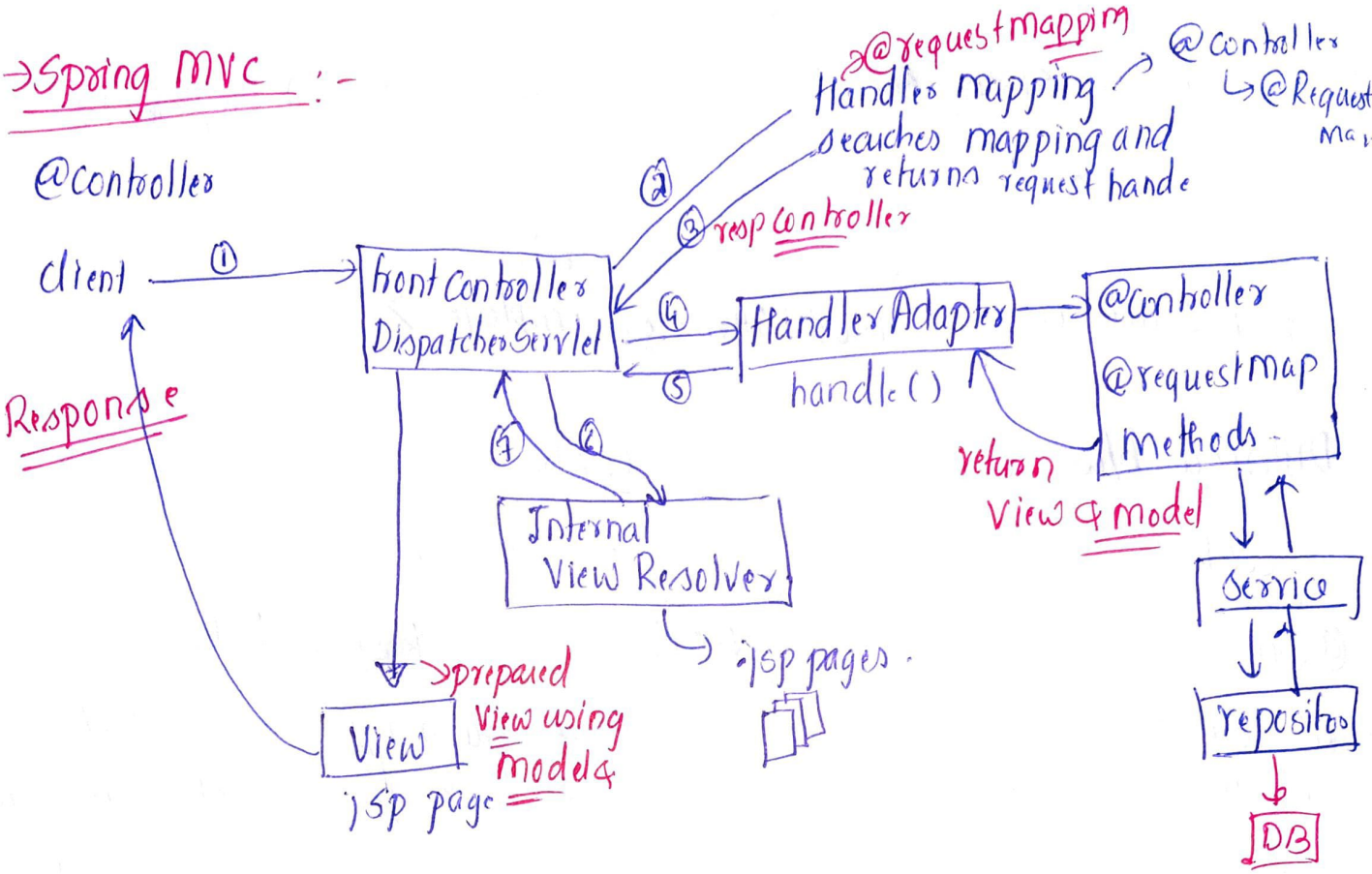
How To create Jar, War in spring boot applicatn (How to add maven)

```
<plugins>
    <groupid>
    <artifactId> spring-boot-maven-plugin </>
</plugins>
<packaging> Jar </p>
```

@EnableAutoConfiguration → looks for auto configuration beans on classpath and applies them.

@SpringBootApplication → @Configuration, @EnableAutoConfiguratn @ComponentScan.

→ Spring MVC :-



→ looks for @Controller

handler Adapter → ⊗ goes hand in hand of Handlers Mapping (@requestmappin("/home"

⊗ DispatcherServlet uses handler adapter to invoke handlers Mapping

⊗ is used to execute methods annoted with @RequestMapping

⊗ ① client sends request to DispatcherServlet ② dispatcher servlet forward to handlers Mapping to search the respective controller

→ By default it uses BeanNameURL HandlersMapping & DefaultAnnotatn HandlersMapping → which is driven by @requestmapping

③

20

① What is role of @autowired annotatn

⊛ used to inject the beans (objects), it is ~~class~~ field level or method level

Model Attribute :-

```
@RequestMapping (value = "/addEmployee", method = RequestMethod.POST)
    public String submit (@ModelAttribute("employee") Employee ){

    }
```

form < action = " ", modelAttribute = "employee" >

Difference b/w Controller & RestController

Controller	RestController
@Controller	@Controller + @ResponseBody
→ It will make class act as Controller and can return View or JSON obj	→ by default return type is serialised and sent into Response body (JSON or xml) → not require to write @Responsebody every method.

@PathVariable :-

```
@RequestMapping("/user/{id}")
    public void handlerequest (@pathVariable("id") int id) { }
```

Validation

@max, @min, @validate, BindingResult → contains validatn error

@ResponseBody @RequestBody

→ data serialised to JSON / xml into response body

→ data deserialised from JSON → Java object

⊛ Binding Result

```
@postMapping("/user")
public String submitForm(@Valid NewUserForm userdata,
                         BindingResult result, Model mode)
{
    if(result.hasErrors())
    {
        return "userhome" ⇒ // converted to userhome.jsp
        model.addAttribute("message", "Valid form");
    }
```

Front-Controller pattern

① model

② doPost, doGet → servlet ⇒ extends HttpServlet.

③ <servlet-name>
<class>

④ here forward → context.getRequestDispatch (target)

<url-mapping> / < >

@Controller @service @Repository

@Componel

→ @Controller, @Service, @Repository

class serves as controller detects @RequestMapping annotn in class

class cointains Methods to call Repository

Dispatcher Servlet Context Loder Listener

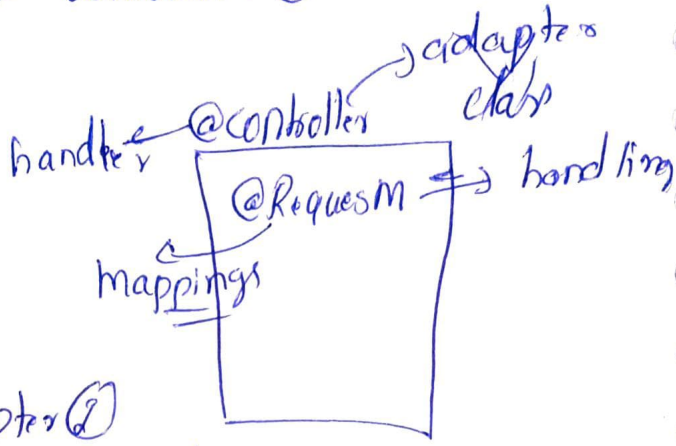
dispatches incomming HTTP Request to right handlers

starts and shutdown "WebApplicatnContext"

Handler Adapter (I)

Simple Controller Handler Adapter
@Controller

Controller (I)



(*) Dispatcher Servlet :-

① finds all implementation of HandlerAdapter (I)
getHandler(), getHandlerMapping() → getHandler() →

② handle() → handles the request
↳ handles object

① getHandler(HttpRequest) → getHandler() → handle(Request, Object handle).
↳ handleExecutionChain →
↳ contain imp of handler → mappings
Returns View

Spring Security

(*) JWT → Json web token → stateless authentication.
↳ Why → so that after spring security every time we dont require to pass username & pass
↳ three part header, playload, verify signature

Spring framework (Aop)

prototype scope → single bean multiple object in ioc container.

Injecting null and empty string

<bean id="employeeBean">
<value=""> → empty string

<null/> → to inject null

@autowired (61)ppt

- can be used at setter, constructor, field level
- it provides more fine grain control over how and where autowiring to accomplish
- can be used with methods with multiple argument

Context → application context represent Spring ioc container responsible for configuring, assembling, beans

Cross cutting concerns :- common services to web layer, Business layer Data layer, e.g security, logging, validation, caching

Aop → is used to implement this Concerns, all cross cutting concerns are moved in aspects.

Jms → Java message service

prototype scoped bean → ioc container is forced to create new bean every time.

Singleton :- single bean returned every time, 62

Singleton → single object per ioc container default

prototype → any number of object as allowed,

request → bean definition to HTTP request.

session → bean definition to httpsession

global-session → scopes bean definition to global http session

Stateless → beans are singleton and are initialised only once

statefull bean → bean with instance variable prototype

BeanPostProcessor → are interfaces used initialise after creation of bean

spring beans are created using class Constructor

Bean attributes →
name/id, class, scope, constr-arg, properties, autowiring, init-method destroy-method,
through setter Bean attribute

factory-method → used to invoke/instantiate bean → thrown bean creation exception

scope → is used to set scope of bean

singleton or prototype decided by (@singleton "true" or false → Prototype

Interface to initiate Beans

InitialisingBean / DisposalBean, interface
↓ Initialise → To destroy beans
init-method, destroy-method

annotatn for initialisatn method
@ postConstruct.
To destroy @preDestroy

To close all beans gracefully?
shutdown hook is used

Message source is used to resolve text messages

Interface - To listen to events
Applicatn listener

Spring security

Servlet filters are used for securing web urls

Context loader listener → creates a root-web-application-context for web application and puts in servlet Context

Dispatcher Servelet ← creates it own web applicationContext and handles /controller/view-resolve
⊗ it dispatches request to web-apps.

Authentication services → authenticatn provider
authentication manager,

user details Servic
User-Service ↳
→ userdetails
↳ store user information

~~JDBC~~ Servlet

⊗ http session is used to perform following tasks
① bind the required object to user
② view and manipulate info about a session, such as session identifier, creation time, & last accessed time.

Sending email through servelt
Java Mail api is used, javax.mail
javax.mail.internet

How to read binary data such as image etc from request object
ServletInputStream, → to read binary data
ServletOutputStream → to write binary data to request object

forward & sendRedirect

① client pull :- url generated by clicking the button or clicking url <a href="" />.

② client pull 2 :- Redirect Scenario
client browser automatically generate new url httpServlet Response →
↳ sendRedirect("");

Server pull
taking client to next page with same request

also know as Resource chaining
↳ request ⟶ Servelt ⟶ JSP
or request Dispatching technique.

① Create RequestDispatcher object to wrap url (only last page can generate)
rd.forward(Req, Res) Respons

Include scenario

* the request creator can add content to the Response but cannot modify, response code or set headers will be ignored

Standalone Servlet Engine
Server that include built in support for Servlet

Httpservlet class
* this is abstract class, we could make servlet able to handle http req; made by user
GenericServld.
↳ Httpservlet

Session Tracking
session :- Http protocol & web servers are stateless → means every request is at new request web server,
requirement → to identify the client
* to remember the client
* default timeout → 30min

Session tracking
Cookie
small amount of text data sent in header

① create cookie
javax.servlet.http.Cookie( )
② add cookie
get cookie → [ ]
getName, getValue( ), setMaxAge( )
def age = -1 → stored in cach. in Secu
0 → delete cookie
>0 → persistent cookie

Session based tracking
only HttpSession id is sent to the client, in the form of cookie
↳ manage by WC

HttpSession ⟶ Interface
① add attribute (Set) ② getAttribute ③ getId
④ invalidate( ) ⑤
attribute scope → page, request session application scope

GenericServlet → can handle any sort of request protocol independent
idempotent :- do multiple request or single request they are sam
1+0=1 ✓
only Post → non-idempotent (because they are used to make new resource at server side)
Get, Head, option, Trace → don't create any resource on server

ServletContext → Defines set of method that servlet uses to communicate with its servlet container
e.g → MIME Type, dispatch request
write log to file
one Context per Web app

ServletConfig	ServletContext
✱ created at initialization time	created at init time
Provide some initial config to servlet	both are used to
✱ it is for specific servlet	✱ it is for servlet Context
✱ passed to servlet as config by Contain.	it is obj created by Servlet Container.
<init-param><value></>	<webapp><context-param><value></>
→ getServletConfig()	getServletContext()

Servlet life cycle

Servlet class → object → config-object → init → service → distroy → method

JDBC

Data types in JDBC

Real → float → support 7 digits of mantissa
Double → double

Driver-manager

✱ is class in java it manages set of java database connectivity (drivers) that are available to use

✱ methods → to get connectn using url in driver manager class

✱ Connection getConnection(String url)

① Register Driver
↳ registerDriver( )

② get Connection by url, password, username

③ create statement

④ execute → resultset

✱ What is Concur_ready-only resultset in JDBC?
If parameter of create statement then result set obtained by such → is only read type not updatable.

✱ What is Concur_~~read-only~~ UPDATABLE?
it is constant of ResultSet class representing concurrency mode for which result set may be updatable.

Rowset : GOG

✱ it is interface in java
✱ javax.sql

Result set in → java.sql
✱ It store data in tabular form
✱ makes easier than resultset
✱ Rowset updable, Resultset is not
✱ it is wrapper over Result set

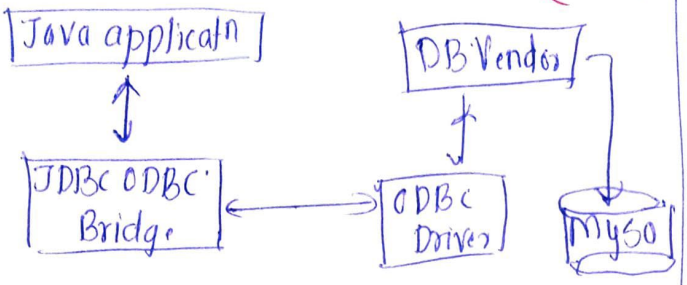
JDBC Driver

it is implementation of interface defined in JDBC API (java.sql)
✱ java.sql contains classes and interfaces, having Impl in

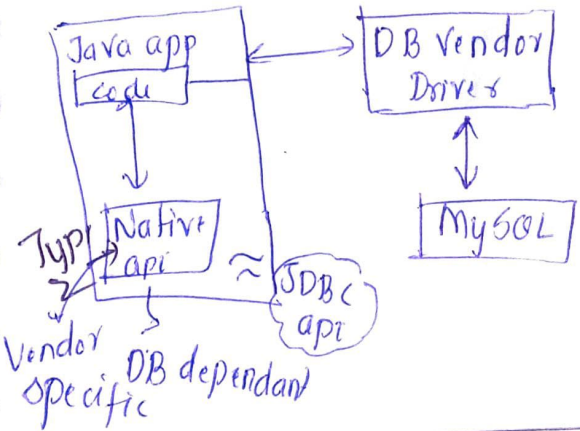
third party Vendor implement java.sql.Driver interface in their driver

↳ Types of driver

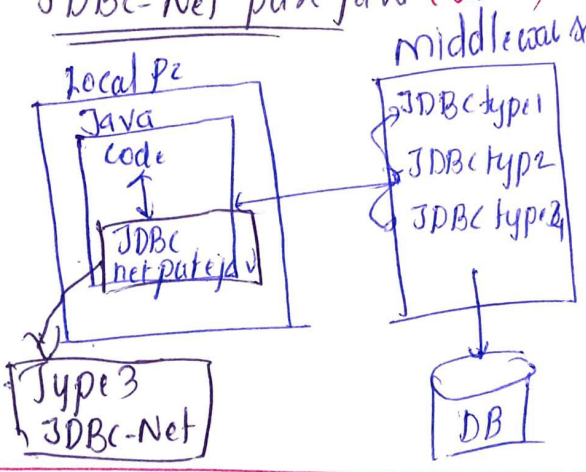
① JDBC-ODBC driver (Bridge) (non)



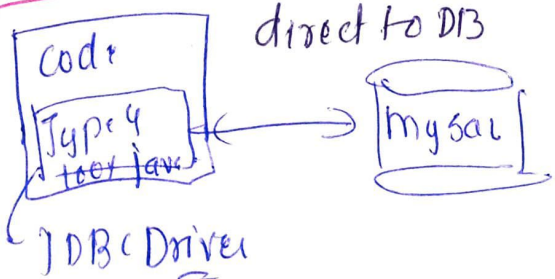
JDBC-Native-api (partial)



JDBC-Net pure java (Java)



Type4 (Java) thin driver or direct to DB



Resultset → can move forward only and it is updatable

Statement Interface

↳ it has methods to execute Queries.

① Create ② prepared ③ callable

① Create statement (iz) ⇒ to obtain object we use connection. CreateStatement

↳ execute(SQL) Boolean.

↳ executeUpdate(SQL) int

↳ executeQuery(SQL) ResultSet.

② prepared Statement : contain compiled Query. '?' used in place of param

? → start with '1'

? → 2 ....

→ ↳ execute()

↳ executeQuery()

↳ executeUpdate()

③ Callable →

↳ execute()

JDBC driver manager class

↳ backbone of JDBC

JDBC

Statement (parameterless query)
↳ executeQuery → Select
↳ executeUpdate → insert/update/delete
↳ execute → stored p

executeQuery return resultset ①
↳ autoclosable

Columns number start from 1
1 id 2 name 3 add

rst.getInt(1).

Prepared statement (2n parameterised)

? → placeholder (L → R)
pst.setInt((1, id)) → start from 1

"select * from emp where id = ?"

↳ pst.executeQuery()
pst. update
executeUpdate();

↳ callable statement → prepared statement
In → set
Out param → register

2xx, 3xx, 4xx, 5xx
2xx → ok
3xx → Redirection
4xx → client side error (failure)
5xx → server side error

Servlet

javax.servlet →
Generic servlet
↳ HttpServlet

Servlet (I)
↓
Generic Http

@WebServlet("/test")

hierarchy:
① Servlet
↳ GenericServ
↳ HttpServlet
(init, service, destroy)
↓
doGet doPost

Deployment Discriptor: - instructn fn
web.xml to create successfull deployment
↳ WEB-INF folder

WEB-INF → hide private (only WC)
↳ web.xml
↳ .class
↳ .jar / library

WC → Service → Service → d
Gener protected

print writer → to send server → client

@WebServlet(value = "/validate")

key	
/valid	FQ name of servlet class

HashMap

Request life cycle

① loading of servlet class → object creation
↳ Init → service → protected service
(Generic) (HttpServlet)
Respon

Xml → Configuratn of servlet class

servlet
name
class
mapping
url pattern

* default loading policy is → lazy (wc) (waits for first request)
* load on start up = 1 (①), 2 (②), 3 (③) ..

---

on start up welcome page
src → main → webapp → index.html
start
web.xml
<welcome> </welcome>

Response.sendRedirect(); → default Get
* flush → not allowed before redirect

HttpServlet, GenericServlet → abstract class
100% concrete

Why abstract :- (To prevent programming errors)
* service method impl is left to programm. compulsory
* no object creatn for these class.

Cookie → class
small text data →
* generated by server send in header
AGE → -1 → store in Browser client Cache
0 → delet
+ve → store on hadis
addCookie, → response meth
getCookie → request
[ ] cookie.

HttpSession (I) Interface
* Serverside management
* request.getSession() → HttpSessi..
session.isNew()
session.getId()
* setAttribute("emp", emp)
ns.
hs.getAttribute("emp")
* hs.removeAttribute("emp")
* session.invalidate();

Send Redirect("logout")

Session Map

outer map <String, HttpSession>
(sinionzd)
Inner map <String, Object>
attribut.

Servlet life cycl

① map of url mapping created
② loads servlet class & creates object
③ Servlet configic object i~~nject~~ creat
④ passed in init
⑤ servic
⑥ destroy

→ config object created used
web.xml

Request Dispatcher - Interface

```
rd = request.getRequestD("logout")
rd.forward(req, resD)
```
(logout ↑ url pattern)

⊗ goes to next page in same request.

After rd.forward
→ Current page suspend
→ pw buffer cleared
→ go to do post method of "/logout"
(not get)

Scopes

private → Default → protected → Public

page → request → session → App.

---

getParameter Vs getAttribute
limited to string ↘ object

Include scenario → pw is not cleared

JSP :-

seperatn of html/css and business logic.

⊗ server page are auto translated to servlet

⊗ servlet
↳ Jsp page
↳ jsp Init()
↳ jspServi
↳ destroy()

Overriding not required wc does automatically

Jsp life cycle

test.jsp (web-app folder) → request

↓ translation phase (JSP → Servlet) once

↓ ~~jsp~~ test.java

↓ Compilatn test.class

↓ load in method area

↓ public void init()

↓ Service

↓ destroyed

Comment → <%-- ~~~ --%>

Implicit object

(abs class) JspWriter, Req, Res, session, Config, out.
page, pageContext, exception (abstract class)
↓ Implicit session tracking

→ pageContext (page environment)

12 JSP → 12 page Context

⊗ pageContext.setAttribute("emp", emp)

Scriplets → To add Java code.
<% java code %>

<%= expression to eval %>
↳ sent to client
<%= new Date() %>.

directly session also.
session.setAttribut.)

EL syntax

Implicit objects accessible to scriplet & expression.

${ }

Param map (parameter map

maps for EL

1. pageScope
2. requestScope
3. sessionScop.
4. applicationScope
5. pageContext.

---

pageContex.setAttribute → getNM.
pageScope.nm1

request.setAttribute("nm2")
→ ${requestScope.nm2}

${pageContext.session.id}

Implicit params are not allowed directly

JSP Expression <%= session.getAtt %>.

EL ${ → sessionScope.nm

Encode sessionid in url

encodeRedirectURL(String url)
respons.

Page directiv.

<%@ %> to include other JSP pages.

<%@ include file="" %>
test.jsp

Jsp Actions

1. useBean 2. setProperty 3. getProperty 4. forward 5. include 6. param 7. plugin.

<jsp:actionName attributlist>
</>

<jsp:forward page="two.jsp" />.
↖ to forward.

<jsp:useBean id="ub"
set property="email" param="email"
Userbean ub = new UBC()
ub.setemail(reques.getparam("email")

<jsp:getProperty name:"user"
property = "validateUser"

Hibernate readme_hibernate.

Why?
JPA → (Java persistent api)

it is open source & light weight

(2) fast performance → due to caching
two cache L1 & L2
↓
default enabled

third level → Query level (not enabled implicitly)

(3) DB independant Query (HQL) JPQL

(4) Auto table creation.

(5) it translate sql exception checked to unchecked

JPA
↳ Hibernate

100% DB independant

Session :- (scope + session)
↳ persistant manager
↳ it is hibernate engine
↳ provides methods to curd operation

Connectn → Session

↳ associated with L1 cache
↳ save()
↳ get(), load()
↳ update(), delete()

Session factory (Interface)

S1 S2 S3 S4

(1) singleton
(2) openSession()
(3) getCurrentSession()

Configuratn object
↳ used to create session factory object
↳ after calling this object hibernate.cfg.xml placed in classpath.
↳

Entity life cycle

(1) Transient State
↳ when object not associated with Session.
↳ no matching record in database.

e.g Account acc = new Account();
acc.setAccno("101");
acc.setName("Amol");

(2) persistent State

session.save(account);
when object is associate with session (L1 cache)

any changes in persisted state will be reflected in DB after commit)

Auto dirty checking

Detached

present in DB but not linked with any session
→ no reflection in DB

session.clear(), .evict(Object)
session.close(),

Update → detach → persist
delete → persist → transient

clear() → all object are dissociate but connectn not closed

Void close → ↓ + connection close

Ⓐ Open session / get Current Session.
↳.

Annotatn

@Table @Column @Transient (skipped from saving)

*@Temporal → java.util.Date, Calendar

LocaleDate(), LocalTime() → no temporal

@Lob

steps ① get session from SF
② Begin transaction
③ try ( )
{
commit
}
catch { roll back
}

---

hibernate.cfg.xml → configuratn to create session factory : ie boot strapping hibernate framework

hibernate.hbm2ddl.auto = update
if table dont exist it create new table

Annotatn	mandatory	optional
@Entity	@Id	@Table(name = "")
class	field	@GeneratedValue

Custom Query

List<Object[]> list
= session.createQuery(jpql, object[].class).setParameter("start", startdate).setParameter("end", enddate).setParameter("rl", userRole).getResultList();

jpql = "select u.name, u.regAmount from user u where u.regDate between :start and :end and userRole = :rl"

Pojos.User(name, regAmount)

Ⓐ executeUpdate();

Maven :-

Build automation Tool for overall project management
it helps
↳ checking Build status
↳ generate reports
↳ auto buil & monitor

pom (project object model)
↳ core of maven
↳ cointains
↳ contents
↳ Dependencies
↳ plugins
↳ Versions
↳ developers involved
↳ Build profile

build life cycle
① validate ② compile ③ test
① clean ②
④ package → verify → install → deploy

JPQL

List<user> users = session.createQuery(JPQL, user.class) .setParameter("date", d) .getResultList();

Advance Hiberate

course 1 ——→ * student

course — student / stu. / stu

parent side → @oneToMany : course
child side (@manyToOne) : student

Owning side :- column name present in table

non-owning → opposite (inverse)

※ Course : one, parent, inverse
Student : many, child, owning side (fk)

---

* course 1 ——→ * student

@oneToMany → inverse side
(mappedBy = "selectedCourse", cascade = All, orphanRemoval = true) → name on owning side
private List<Student> students = new ArrayList<>() **

@manyToOne
joinColum (name = "course_id",)
private Course courseChosen

Spring

why spring?
① loose coupling b/w dependant & dependancy
② ready made impl for (mvc, singlton, orm)

③ Spring Container :- manages life cycle of spring beans

Dependency Injection :- (wiring = collaboration b/w dependant & dependency)
→ IOC instead dependant managing dependance third party (spring / sc) managing dependencies @ runtime is call IOC

Spring SPEL

#{ }

Springbean :- java object whose life cycle completely managed by SC
e.g restController, controller, service, dao

D.2 → JSP → Javabean → DAO(utils) → pojo
Dependant object → Java beans, Hibernate based DAO, JDBC based dao
dependencies → DAO, sessionfactory, DB connectn

※ here dependant objects are managing their own dependancies

User Controller class

@Autowire
private sessionfactory

wiring

Explicit wiring
↳ must supply setter, constructor, factory Method + xml configuration
↳ ① Setter based
↳ provide setter in dependant
↳ <property name = "" />
in xml config

Constructor Based DI.
↳ provide parameterized constructor
↳ <constructor-arg name/type/> tag in xml

factory method Based

Implicit wiring
supply setter/constructor, N/O conf in xml

autoWire ByName
① provide setter in dependant bean
② must match setter name = "BeanId"

Autowire By Type
① provide setter in dependant
② SC match data type of property to type of the dependancy bean.
③ ambiguity should not present

Autowire Constructor
① provide parameterized const
② SC match datatype of const arg to type of dependency bean.
④ throw except if unique not found

Spring-bean-life-cycle-png

① spring container start by classpath xmlApplicationContext or automatic in web app by Dispatcher servlet
② SC → xml configuration / Anno
③ checks the scope
④ if (Prototype)
↳ Complete life cycle for every call ctx.getBean
⑤ if (singleton)
⑤ checks lazy init
true / false
~~false~~ True
same invoked after first call
⑥ init)
locate
load
instantiate
⑦ I/c by default const or parameterized const
Const DI
⑧ setter Based DI
invokes matching setter
↓
init method
↓
invok myInit()
↓
Bean ready

Shutdown.
checks destroy method
↓ marked for gc.

<context:component-scan base-package = "comma sep list" />.

SC searches @Component @Service.
@Repository @Controller @RestController
@ControllerAdvice

@Component
↳ @Controller
↳ @Service
↳ @Repository
↳ @RestController

@Component → <bean id, class />.

e.g. @Component("abc")
public class MyBean{ }

<bean id = ~~MyBean~~ abc class = "beans.MyBean" />

or

<bean id = MyBean ... />

@Scope(value = "singleton/prototype/request/session") → scope
class level

@Lazy(true/false) → lazy-init

@PostConstruct → init-method
method level

@PreDestroy → destroy-method

@Required(T/F) → if bean is mandatory or not

@Autowired → field level
private service
↳ By Type.

@Autowired
@Qualifier("soapTransport")
↳ By Name
↓
<bean id = soapTransport />

success

(X) @Autowired
private PatientService pservice

any PatientService type Bean is injected

MVC

model
View
Controller → Servlet (manages Navigation)

front-Controller
gate keeper to intercept all requests

In spring we have ready made implementation for front controller, handler mapping, view Resolver..

In Webapp → DispatcherServlet starts spring Container

Handler Mapping Bean
(I) Interface.
impl class:- RequestMappingHandlerMapping
↳ Map → populated by SC @ deployment time.

Key	Value
@RequestMapping	FQ ControllerClsName + MethodName

View Resolver Bean

spring provides web.servlet.InternalResourceViewResolver
p: prefix = "/web-inf/view" p: suffix: ".jsp"

View Resolver ①
↳ InternalResourceViewResolver

"/welcome"
→ /Web-Inf/Views/welcome.jsp

@Controller → tells 'SC' that this class contain @RequestMappings

---

DispatcherServlet → starts spring Container in webapp
WebContainer (WC) → will start life cycle of DispatcherServlet

---

Model Attribute :- (key value)
→ used to store result of B.L
→ created by Controller & sent to D.S
↳ After D.S gets View Resolver name from VR DS checks from Model attribute

---

Model attribute are stored in Request scope
↳ {requestScope.attributeName}

how controller sends add model attribut
ModelAndView ⇒ holder for attribut + View name

@requestMap
-public string/ModelView/ui.model addEmp(){

}

---

ui.Model ②/①
map for model attribute.
model.addAttribute.

---

map is supplied by SC
IOC → DI

To tell SC
@Request( )
public string addPatient (Model M)

Request Parameter
@RequestParam ("price") double price
= price

↳ price = Double.parseDouble (request.getParameter ("price"))

How to format date
Default mm/dd/yyyy.
@DateTimeFormat
e.g @RequestParam ("DOB")
@DateTimeFormat (pattern="")

Spring Boot

⊛ "opinioted default Configuratn"

⊛ Advantages
↳ embedded server

Spring boot = Spring + Embedded server + xml Based configurn efforts in identifying pom xml

Opinioted Default :-

↳ can we auto configure
Data sourw, connectn, session factory
Tx manager, If hibernate jar is
on classpath.

---

when spring-mvc-jar added
to class path can we auto configur
Dispatcher servlet, Handler Mapping,
View Resolver

Spring-boot-starter
→ set of convient dependancy to start
the project

@SpringBootApplication
used to bootstrap and launch a
spring application from java main
method
① It will do
① create applicatn context
② manage bean life cycle

① @SpringBootConfiguration
↳ to tell that there are beans
and keep them ready @runtime
② @EnableAutoConfiguration
↳ automatically configure based
on dependencies. in classpath

---

e.g pom.xml ↓Properties
mysql dependcy → ○ → ready to use bean

Spring Web → DS bean
HandlerMapping bean
ViewResolver bean

@ComponentScan
(context:component scan)
to tell contain all annotatn
@Component & sub components)
Default pkg → main class present
package

@ComponentScan(basePackages = "com")

JPA in Springboot

@PersistenceContext / @autowired
EntityManager mgr
mgr.unwrap(Session.class)

To get http session → add to arg
method

@Transactional →

Why Spring Data JPA

Repositories →
① Crud Repository
② Paging & Sorting Repository
③ JPA Repository.

① crud Repo → Create Read Update
delete operatn.
↓
extend
to
↓
paging → findAll + flush

Crud Repository
↓
paging and Sorting Repository
↓
JPA repository

Property traversal

P ⟶ has address ⟶
↳ strin zip Code.

P. address. zipCode

Rest

Representational State Transfer
↓
text JSON XML

@ RestController	@controller
@ Controller + @Response Body	
⊗ @ ResponseBody on Return type is used tell that send data as JSON	

@ResponseBody

↳ written on return type of request handling method

⊗ it tells spring send Object directly

⊗ as Jackon Jar is on class path SC can convert automatically

@Response / Request Body

@RestController

@Controller (at class level)
+ @ ResponseBody added on all request method return type

@ PathVariable(nam="pid") int id)
argument level

@ CrossOrigin
class level.

To allow react client

```
@Cross Origin(origins="http://localhost:3000")
@RestController
public class Controller{ }
```