SPRING

DATE
PAGE

Introduced in 2003

Spring :- Demanding, robust, stable. JAVA framework.

⇒ It is open-source java-framework which is used to develop Enterprise Application, Stand-alone Application

⇒ Rord-Johson (developer of Spring).

⇒ Initial-release in 2003., Production-release 2004

It provides security, modular approach, it makes applications light-weight, provides features like AOP, IOC, DI, transaction management

Advantages :-

① DI (Dependency Injection) :-

DI is a design pattern, DI is technique where an object receives its dependencies from an external source (Spring Container) rather than creating them its own.

Beans = Component

② Tight-Coupling :- one component totally depends on another

Coupling.

Example :-

```
class Car{
Engine engine;

public Car(){
engine = new Engine();
}
}

Car c = new Car();
```

```
class Engine{

}
```

new car()

engine

Engine is totally depends on car

Application ke Common problems ko resolve karne ke liye ⇒ design pattern banaye gaye hai

DATE
PAGE

TightCoupling :- Agar ak component me changes karte hai, To dusre me bhi changes karna hota hai, werna code fat jata.

(B) Loosly - Coupling :-

```
Class Car {
Engine engine;

Public Car (Engine engine) {
  this.engine = engine;
}
}
```

```
Engine
class Car {
public Engine() {
}
public Engine(String FuelType) {
}
```

(1) Engine eng1 = new Engine();
Car Car1 = new Car(eng1);

(2) Engine eng2 = new Engine("Diesel");
Car Car2 = new Car(eng2);

Dependency Injection make your Application loosely Coupled.

(C) Supports multiple Configuration. :-

Spring IOC (Inversion of Control) :- It creates the objects automatically. HUMF Bas IOC ko batana Hota hai kisko

DI <- Constructor injection
<- Setter (getter)
<- ~~autowiring~~ Auto-wiring

DATE
PAGE

object banana (spring, bean, pojo) bas Hume beans, or pojo ka configuration karna Hota Hai

(3) Support multiple-Configuration

① XML-Based Configuration (outdated)
✓ upto spring 5 x spring 6 x.

② Java Based Configuration

③ Annotation Based Configuration.

④ Spring-Security :-
(like authentication / authorization)

⑤ AOP :- Aop pradigm
(Aspect oriented Programming)
we can implement Cross-cutting features
#CROSS-cutting :- like 3bar login HI-HO, use baad account block/ HO jaye

⑥ Make database Access easy
we can also use this with ORM tool, JDBC, etc.

⑦ Light weight and modular :- It makes your Application light weight & modular.

Spring Container is like JVM for spring Application.

DATE
PAGE

Spring Core Component:

Heart of Spring / Spring ko control de diya.

① Spring Container (IOC (Inversion of Control))
main Component of spring.

⇒ It is responsible to create bean, manage lifecycle of SpringBean, and provides Spring features (AOP, etc)

- Types of Spring Container :-

Ⓐ BeanFactory	Ⓑ Application-Context
Both are the Interface	⇒ It is child of BeanFactory
① if Application is small scale, go for Bean Factory	①. enterprise-level-Application, use Application-Context
② Provides Basic-level DI	② provides advanced-level-DI
③. It doesn't support AOP features	③. Supports AOP features, Auto wiring features

How to define a Spring Bean ?

① using @Bean,

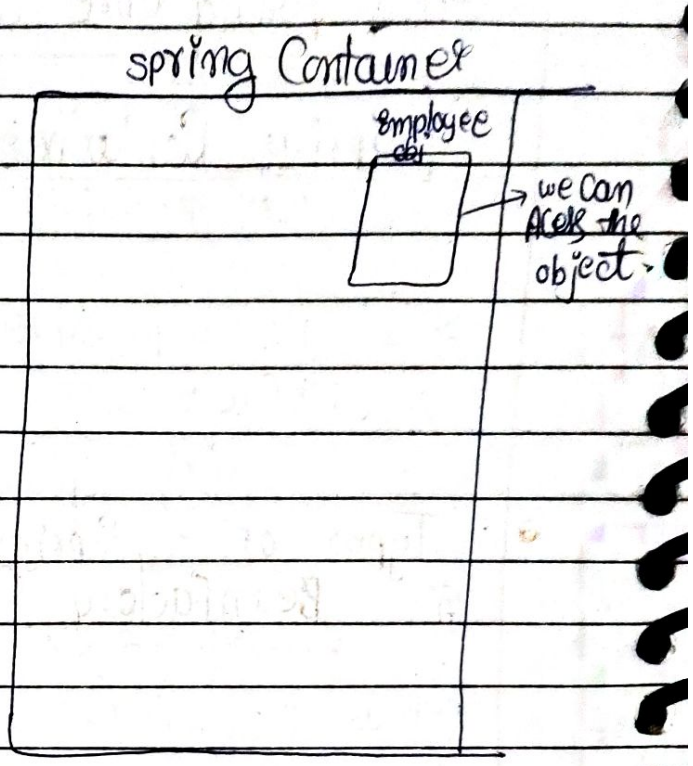
② Using @Bean in @Configuration class

③ using XML <bean> tag

Example :-

It is pojo/⇒ Bean class.

```
Class Employee{
  private int id;
  private String name;
  // G & S
  // Constructors
}
```



~~JAVA Class~~

xml file → xml-Based con. ← outdated

→ Java-Based con. ✓

→ Annotation-Based con ✓ } Java class.

Configuration file ↑

spring Container reads the configuration-file. At the Start-up of Application

Interview question :- which design pattern is used by spring?

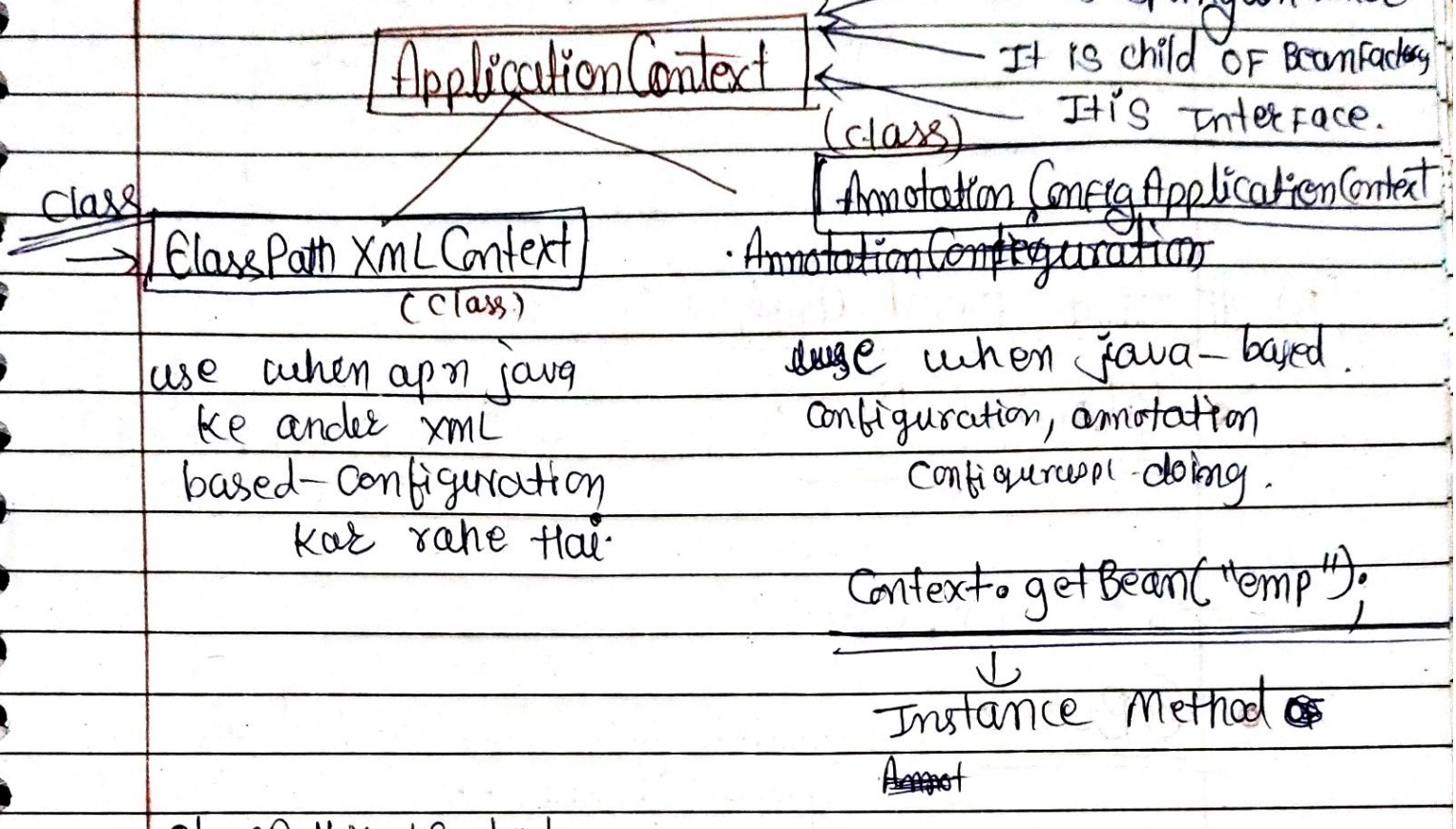
→ ① DI (Dependency Injection) pattern

→ ② Factory Design pattern :-
It creates the obj, but vo apne like matlab nhi karta ke kis ka tarah se Banaya hai

③ Singleton Design pattern :-
Ak hi object ko wapis use me le leta hai, or call karne par usi obj ko return kar deta hai

Ex :- SessionFactory
EntityManagerFactory

④ MVC Design Pattern



Data setting tag ⇒

< property name="id" value="100" />
< property name="name" value="Atul" />

(A) XML-Based-Configuration :- In this Configuration, we create a configuration file in resource folder (like resources / applicationContext.xml) :-

Now In this Configuration-file :-

⇒ Configure, the bean class using Bean tag.

< bean > < / bean >

< bean class="com.info.welcomespringapp.entity.Employee" id="emp" >.

This tag is used to set the data in variables

```
<property name="id" value="100" />
<property name="name" value="Atul" />
</bean>
```

(B) # Java-Based-Configuration. :—

- @Configuration :- ⇒ use to get know container that this class is Configuration file.

⇒ used upon any class

- @Bean :- Indicates that this method is Bean Object.

ApplicationConfigContextApplication is used in java-Based-Configuration.

for example. :-

```
@Configuration.
public class JavaConfig {

@Bean
public Employee. getEmployee(){.
  Employee e = new Employee();
    e.setId(100);
    e.setName("Atul");
    e.setDepartment("Java team");
  return e;
}.
```

In case me. getEmployee() hi (id) bean jata hai method → Id in JAVA-Based Configuration.

(C)

Annotation-based-Configuration :-

- @ComponentScan(basePackages = {"com.info.annotationConfig.entity"});

⇒ used to create the objects of all classes in entity Packages

It Scan the entity packge & create the objects.

- @value :- variables ka data set karne ke liye.
- @Component :- Entity package ki classes ke upper lagenge.

XML Configuration :-

APP.Java class

```
Class APP {
PSVM(){
ApplicationContext context = new ClassPathXmlApplicationContext
                              ("applicationContext.xml");
Employee e = (Employee) context.getBean("emp");
S.o.p(e.getId()+" "+e.getName()+" "e.getDepartment());
}
}.
```

// why we downCaste. :-

getBean() method ka return type Object Hai

⇒ object obj = context.getBean("emp");

Spring ko nhi pata ke apka bean Employee type ka hai ya koi aur isliye default return type Object hota hai.

object se Employee banane ke liye DownCasting zaroori hai

Apka bean actually Employee type ka Hai.
Par Object type me Employee ke methods nahi mille, jaise

- getId()
- getName()
- getDepartment()

{ object me sirf basic methods Hai jaise (toString, equals, hashCode)

Isliye hum downcast karte hain.

```
Employee e = (Employee) context.getBean("emp");
```

Taaki aap Employee ke methods ko access kar sako.

Short reason :- getBean() returns Object → Employee methods chahiye → DownCasting required.

⇒ JAVA 5

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

⇒ Type-Safe

Spring is a framework of framework. initial release 2003
open-source DI framework Production release 2004
Based.
Spring XML Schema

IMP. questions interview

Difference between Bean & Object ?

Object	Bean
1) Instance of class created by using new keyword	2) object that is created, managed & controlled by Spring Container.
2) manually created by developer	2) Created automatically by Spring.
3) keyword used new className()	3) @Component, @Service, @Respository, @Bean etc.
4) Developers manages lifecycle	4) Spring manages lifecycle.
5) Depecdency Injection not Supported	5) Dependency injection is supported

All beans are Objects, but not all objects are Beans.

Dependency Injection

① Setter Injection
② Constructor Injection
③ field Injection (Auto-wiring)

Setter Injection :-

<propety name="id" value="100" />
Object ki property me data set karneke liye, setter method ka use hota hai internally

Constructor Injection :-

<constructor-arg value="200,"/>
<constructor-arg value="Peeku"/>
Name nhi likha kyuki, jis order me parameter hai vese, hi data set hoyega.

```
public Customer (int id, String name){
    this.id = id;
    this.name = name;
}
```

⇒ index, name bhi use kar sakte hai (optional)

<Constructor-arg value="2000" index="0" type="int" />
<constructor-arg value="Aftal" index="1" type="java.lang.String" />

when Dependency is of Reference type use ref

<bean class="com.impo.testdb.entity.NomineeDetails" id="nominee">
<property name="name" value="chinky" />
<property name="contact" value="" />
</bean>

<bean class="..... .Customer" id="customer">
—
—
<property name="nomineeDetail" ref="nominee" />
</bean>

Bean. ⇒

Class Employee {
private int id;
private String name;
private List<String> addressList;
private Set<String> contactList;
private Map<String, String> projectlist;

Configuration ⇒

List :-
<property name="addresslist">
<list>
<value>800, sche71</value>
<value>700, Rchoma</value>
</list>
</property>

Set :-
same as it as List
only <list> (tag) replace to <set> (tag)

Map

```
<property name="projectlist">
  <map>
  <entry key="John" value="Thoren..."/>
  <entry key="Hari" value="---"/>
  </map>
</property>
```

autowire (4 Modes) (2 objects ki aapas me wiring karna)

① By Default (Default)

② "byName" → Name se dekhta hai but Bean ki property ka Name or Id ka name. Same Hona chahiye.

③ "byType" → Id se koi farak nhi pdta, but when we create 2 Id of same Type than it creates Exception.

Solution :-

```
<bean autowire-candidate="false">
```

ByDefault mode { it does participate in autowiring }

① is Default

② false.

IN JAVA Based Configuration if we can do both.

(1) @Bean(name="") // ya to bean ka name rakh do. (like name="Neha")

(2) public Student getStudent(){ // name hi assign karenge to, jo method ka name rai, wahi bean ka name hai ⇒ for unique identification

}

// getStudent()

Spring JDBC Module :-

use to connect JAVA application to Database.

Spring JdBC module :-

- DriverManagerDataSource
- JdbcTemplate

} classes available in Spring Jdbc module

① DriverManagerDataSource :-

establish.

~~Connection Establish~~ responsible to ~~set~~ the connection with our database.

~~set~~DriverName, setURL, setUsername, setPassword

Spring - JdBC 6.2.11

(2) JDBC Template :-

when you want to perform. database operation, we use JdBC Template (like session):

DriverManagerDataSource responsible. to provide. JDBC Template object.

In Spring :-

Modele → Student

DAO → StudentDAO
// Database operation.

SERVICE → Service
StudentService
// for Businesslogic
// in service layer, we call the dao methods, and we handle the data. Transaction in Service layer.

Sterotype Annotation :-

@Repository :- marks a class as DAO / data access Layer

@Service :- marks a class as Service Layer (for Business logic)

@ Controller :-

@ Component :-

Use :-

Sterotype Annotation :-

which can be detected by @ ComponentScan.

Base package ko scan karega, or jin classes ke upper @ Service, @Component, @Controller annotation hai, To ye khud object bana dega.

DATE
PAGE

✗ @Autowired
-JdbcTemplate template ; // field Injection

(1) But it is not immutable
(2) NullPointerException (in case of unit-Testing)

unhi

~~Uhi class or~~ Beans. ke bich autowiring kar sakte hai, jeanka object banane ki controll / responsibity IOC ke pass HO.

Un class, Bean ka. ki entry IOC me nhi hogi, jnka data hum runtime par set karenge like Model class

kar sakte hai Entry, per chuki yeh singleton design pattern use karta, hai hai bar wahi same data dega, To data inconsistency ki dikkat ho sakti,

Solution is @Prototype.

DI = we inject one object into another object.

JAVACONFIG class

```
@Bean
public DriverManagerDataSource getDataSource(){
DriverManagerDataSource dataSource = new
                                DriverManagerDataSource();
dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
dataSource.setUrl("jdbc:mysql://localhost:3306/jtep-
                                Springjdbc");
dataSource.setUsername("root");
dataSource.setPassword("...");
  return dataSource;
}

@Bean
public JdbcTemplate getTemplate(){
  JdbcTemplate template = new JdbcTemplate();
  template.setDataSource(getDataSource()); //manual DI
    return template;
}
```

DML ke liye update()

Update()

template.update(SQL, object... args);

- This update method is present in JdbcTemplate class.
- the return Type of this method is int;
- Used for DML operations

Generic

Interface

- **RowMapper :-** ← Behind, Table Model se data fetch karna hai, from JdbcTemplate than we need to create separate row mapper.

(1) Query () // for whole data
(2) queryForObject // for any particular data.

ResultSet ke har row ko, RowMapper iterate karega.

for Select query :-

```
public List<Student> getList(){
List<Student> list = template.query("select * from student", new StudentRowMapper());
return list;
}
```

RowMapper (interface)
↓
overridden method
⇒ mapRow( );

for example :-

id	Name	course
1	Neha	JAVA

↓ converts into java object

```
Student s = new Student();
s.setId(1);
s.setName("Neha"); /s.setCourse("JAVA");
```

RowMapper working :-

RowMapper converts each row of the ResultSet into a java object. JdbcTemplate automatically calls this RowMapper for every row returned by query.

final = Autowired
static = X

- @EnableTransactionManagement :- Handle the transaction, it ~~active AOP~~ Transaction automatically
- @Transactional :- where to handle the transaction.

kis method ke ander transaction Handling chahiye, uske upper yah annotation lagega.

DataSource Interface

DriverManagerDataSource class

Need to Create DataSourceTransaction ⇒ when we need transaction in our application

```
@Bean
public PlatformTransactionManager transactionManager(
    DriverManagerDataSource dataSource){
    return new DataSourceTransactionManager(dataSaurce);
}.
```

we should use :-

DataSource instead of DriverManagerDataSource

Because in production, it supports and recomanded for "Connection pooling"

~~Jab hum~~ ye ak solution, hai jab ~~hum~~ ak hi connection object ko pool me store kar leta hai, or use karte hai, bar bar..... more faster and frequent

@Autowired :-

It tells Spring to automatically inject the required dependency.

Spring-Data-JPA

- It use internally JPA
- It provides us some interfaces :-
  CrudRepository, PagingAndSortingRepository, JPARepository

Bean object ←
- LocalContainerEntityManagerFactoryBean
- JPATransactionManager.

@Configuration, @EnableJPARepository, @EnableTransactionManagement
Class JavaConfig {.

DriverManagerDataSource.

LocalContainerEntityManagerFactoryBean → Bean name must be EntityManagerFactory

JPATransactionManager.

}

Spring-Data-JPA

it is a specification, it provides us a specification (like Hibernate)

Dependency set up for Spring-Data-JPA

version 6.1.11 (spring-context)

version 6.1.11 (spring-ORM)
version 3.2.3 (spring-data-jpa)
version 6.4.4 (hibernate-core).
version 8.2.0. (mySQL).
version. 3.1.0 (Jakarta)

LocalContainer --- ko batanapadega ke kis package me entity hai, sath hi ORM vendor name. bhi batana padega.

```
{ ⇒ factory.setPackagesToScan("com.info.springJPA.entity");

HibernateJpaVendorAdapter adapter = new
adapter.setShowSql(true);
factory.setP
Properties prop = new Properties();
prop.put("hibernate.hbm2ddl.auto", "update");
prop.put("hibernate.dialect", "org.hibernate.dialect.
factory.setDataSource(dataSource);          MySQL8Dialect");
factory.setJPAVendorAdapter(adapter);
factory.setJPAProperties(prop);
return factory;
}
```

Collection ki class ka ven

Directly using Spring-Data-JPA in SPring

```
@Bean( name = "entityManggerFactory")
public LocalContainerEntityManagerFactoryBean getFactory
    ( Dr------ DataSource){

LocalContainerEntityManggerFactoryBean Factory =
 new LocalContainerEntityManagerFactory();

}
```

Spring khud hi Repository Interface ki child classes banadeta hai, or. fir proxy object return karta.

bss @Autowired Annotation laga denge.

What is AOP ?

AOP (Aspect-oriented Programming) is used to separate cross-cutting concerns like :-

- Logging
- Security
- Transactions
- Performance monitoring

What is Aspect :-

Aspect :-
A module that contains cross-cutting logic.

IOC Container responsibilities :-

The container responsible for :-

- Creating beans
- Injecting dependencies
- Managing lifeCycle

Example :-
- ClassPathXmlApplicationContext
- AnnotationConfigApplicationContext

Spring web MVC (module)

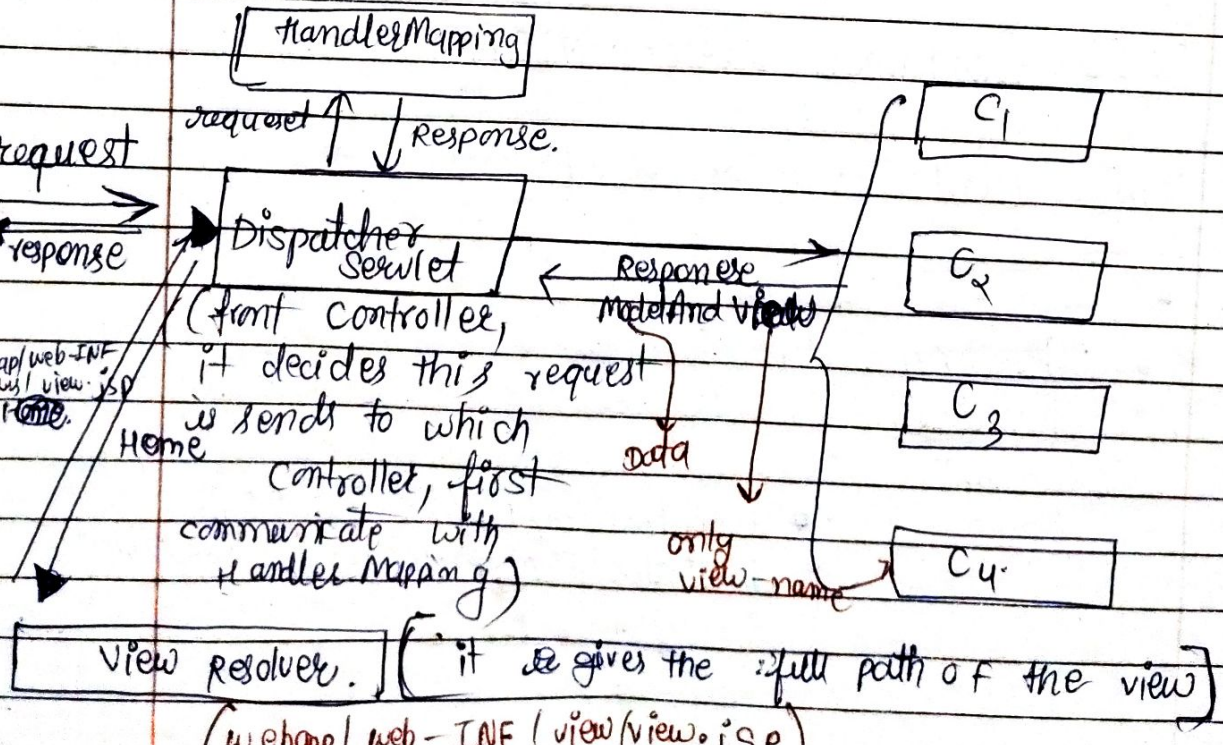
Spring web Module:-
it internally implements Spring web MVC

Normal MVC (Architecture)
M → Model → Represent data in application / Data Layer

V → View → UI layer / UI part / representation Layer.

C → Controller → / Backend.
↓
it handles requests and generat response.

Spring MVC Module :-



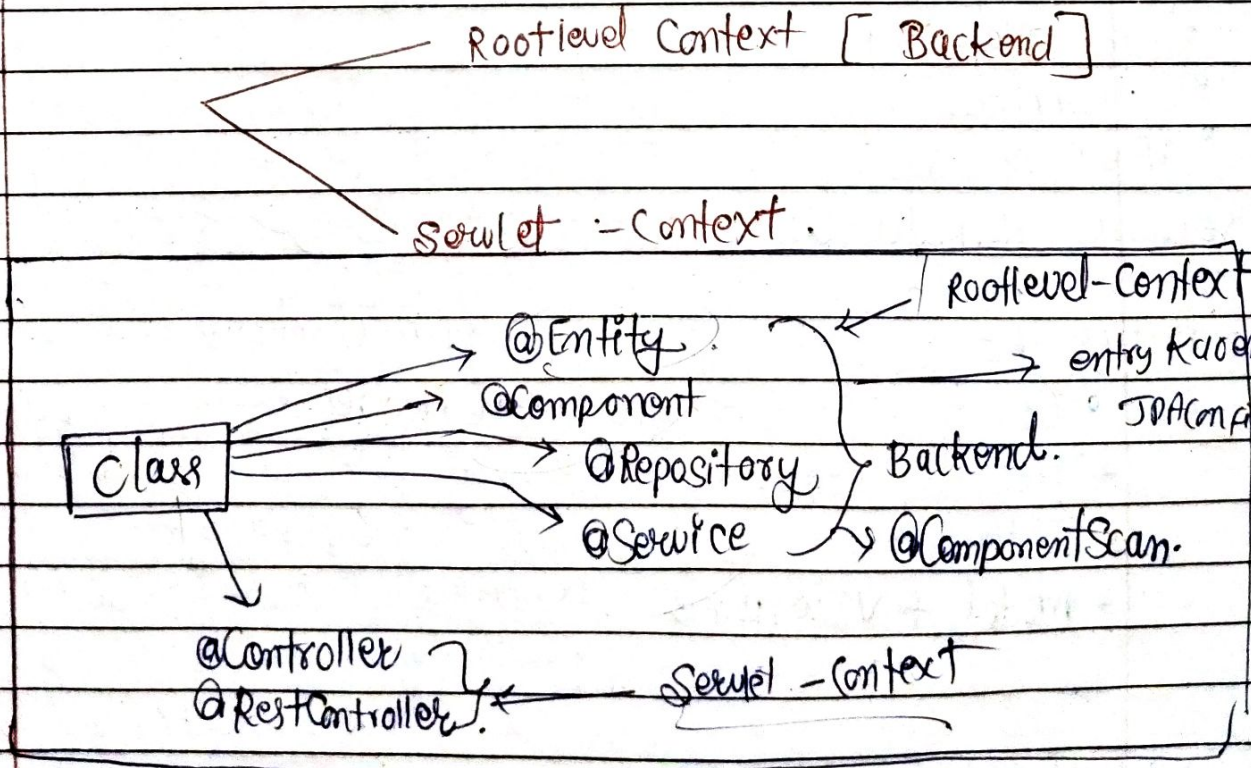
In Spring, there is a class called Model.

AbstractAnnotationConfigDispatcherServletInitializer class.
It replace complet web.xml | this class consists some methods, one handle ~~root-level-context~~, and one Servlet-context

To use Spring web MVC, we create. here.
[WebMVCConfigurer] ← entry of view resolver is here
If activates the web-MVC.

Spring Data, transcational, Backend related configuration is here.
[JPAConfig]

Spring web MVC manages two Context :-



⑤ View Rendering

- View receives model data, • View generates HTML PAGE
- Respect sent back to browser.

Spring Web MVC (Internal working - flow)

① DispatcherServlet (front Controller) :-

what it does :- DispatcherServlet is the central controller of spring webMVC. All HTTP requests first come to DispatcherServlet.

- Responsibilities :- • Receives the request, • finds the correct controller, • Calls controller method, • choose the view, • Return respone to client.

DispatcherServlet follows front Controller Design Pattern.

② HandlerMapping :-

what it does :- HandlerMapping tells DispatcherServlet which Controller method should handle the incoming request.

How it works :- • DispatcherServlet asks HandlerMapping "which controller can handle this URL?"

- HandlerMapping checks mappings like :-
  - @RequestMapping
  - @GetMapping
  - @PostMapping.
- Returns Handler (Controller + Method)

③ Controller :-

Role :-
- Contains businesses logic
- Processes request
- Returns :-
- **Model + View name**

④ ViewResolver :-

what it does :- ViewResolver converts logical view name into actual view page.

example :- return "Home";

ViewResolver maps it to :-

/WEB-INF/views/Home.jsp

Common ViewResolvers :- • InternalResourceViewResolver (JSP), • ~~Thy~~ ThymeleafViewResolver, • FreeMarkerViewResolver.

getObject ← If returns entityManagerFactory

Tomcat Apache create dispatcher servlet

for view Resolver (in webMVC Config).

@

InternalResourceViewResolver getResolver(){

}

@EnableWebMvc

public class webMVCConfig implements WebMvcConfigurer{

@Bean

Public InternalResourceViewResolver getResolver(){

InternalResourceViewResolver resolver = new InternalResourceViewResolver();

resolver.setPrefix("/WEB-INF/view/");

resolver.setSuffix(".jsp");

return resolver;

}.

In Controller → we define action method, which returns 'root'

@Controller

public class HomeController{

// https://localhost:8080/HelloMVC/ ——> GET/POST/PUT/DELETE

@GetMapping("")

public String getHomePage(){

return "Home";

}

}.

https://localhost:8081/testmvc/.create-new-account

Base URL — DATE/PAGE/Route

<%@ page IS IsELIgnored = false %>

"${pageContext.request.contextPath}" ⇒ contextPath

Base URL in jsp (it returns the base URL of your Application)
(expression in jsp)

it works only when

Indirectly routing :— "redirect:/home"
it creates new request every time

direct :— "home"
it Dispatch the view directly.

Magic Method.

Rule :— (By anna chakro).

Optional<Admin> findbyEmailAndPassword(String email, string password);

it returns the optional Admin.

It is class, used to handle the null.

Question :— Spring Data JPA me Optional kyu use hota hai?

Optional null values ko safely handle karta hai aur NullPointerException se bachata hai, especially findById() jaise methods me

- **Complete Flow (step-by-step)**

Client Request

↓

DispatcherServlet

↓

HandlerMapping

↓

Controller Method.

↓

Model + View Name.

↓

ViewResolver.

↓

Actual view (JSP/HTML)

↓

Response to client