Spring Framework

→ It is extension of Servlets Technology.

Example -

Preparing Tea - Approch 1 - At Home

needs - (milk, sugar, Tea powder) → Technology is Combo of 50% Inbuilt library + 50% Programmers

Approch 2 - Tea Bags

Framework Combo of 80% Inbuilt library + 20% programmers.

Note - An application can be developed by using technologies & frameworks, but using frameworks, we can develop in less Time. Hence Frameworks are preferred.

Drawbacks of Servlets (OR) Need to Learn Spring framework.

1) Model - 1 Architecture.

Example - Adding Two numbers → one HTML file → one servlet class.

Subtracting Two numbers → another HTML file → Another servlet class.

Conclusion - One HTML file request is given to One servlet class. this is referred as Model-1 Architecture.

If we have 10-HTML file, then 10-

Example - Registration Process → Insert query
Login Process → Select query
Forgot Password Process → Update query

[According model-1 Architecture, We require 3. Html files and 3. Servlet classes]

Drawback - Model-1 Architecture (Servlets) consume more memory as Number of file increases. Hence, Performance of an Application decreases.

2) Model-2 Architecture

Conclusion - multiple HTML files is given to one Spring class. this is referred as Model-2 Architecture.

Example -

Two Html files → 1 - Springclass.

In add.html <form action = "m1">

In sub.html <form action = "m2">

i.e. By using method we can connect springclass with html file.

```
class Springclass
{
    + void m1()
    {
        // addition
    }
```

Example - Registration Process → Insert Query
Login Process → Select Query
Forgot Password Process → Update Query

[According to Model-2 Architecture, 3- HTML file & 1- Spring class with 3- methods.]

Advantage - In Model-2 Architecture (Springs), memory occupied would be less. Hence, performance would be high.

Note -
→ In model-2 Architecture, As a <form action = "">, we represent Servlet class Name.
→ In model-2 Architecture, As a <form action = "">, we represent One of the method name present in spring class.

Reason-1 - Springs framework provides model-2 Architecture for developing web-pages

Reason-2 - Springs Framework avoids Boilerplate Coding.

Boiler Plate Coding - Repeatation of code to perform task is referred as Boilerplate coding.

Example - JDBC Concepts

→ Insert Data
URL
Insert query
Connection
platform (stmp, pstmp, cstmp)
executeUpdate
close.

→ Modify Data
URL
Update query
Connection
platform
executeUpdate
close

→ Remove Data
URL
delete query
Connection
platform
executeUpdate
close

→ Reterive Data
URL
select query
Connection
platform
executeQuery
Resultset
Close.

→ URL, connection, close repeated in all the task ⟶ It is referred as BoilerPlateCoding.

Note - Platform can't repeated bcoz its depends on that particular programmer who perform the task.

How Springs framework avoid BPC?

→ Springs follow "write Code Once And Use It For Multiple Times".

→ Programmers will make use of XML files or Annotations in order to follow springs principle.

Drawback & Advantage of object Creation -

mainly there are 3 ways of access Nsm by using obj creation -

1st - new HelloWorld().msg();
new HelloWorld().great();

Drawback - 1) We can call only one method at a time by creating one obj.
2) Acquires more memory for every new object.
3) BoilerPlate Coding.

2nd - HelloWorld h = new HelloWorld();
h.msg();
h.great();

Drawback - If we want to access that methods in another class then again we have to create the object of class in same project then it leads to BoilerPlate Coding.

3rd - (Spring Approach) - Keep Boilerplate Coding in xml file or use annotations.

XML files - (1st Approch in Spring's Framework)

→ In XML files, Group of objects is referred as Beans.
One object is referred as Bean.

→ <bean> </bean> → takes argument
1) id - reference variable.

2) class - FQCN (Fully Qualified Class Name)

to get this ↑ right click on class Name & select Fully qualified class Name option.

Spring Containers -

→ The main Purpose is "Carry objects / Beans or XML files to Java class".

XML file → Spring Containers → class.

→ Objects are present in XML files but usage of them is in java class.

Note - All objects / Beans will be Upcasted to object type by spring Containers. Programmers have to downcast it to their respective types.

→ Spring Containers is of 2 Types:

1. Application Context interface.
2. Beanfactory interface.

Syntax for Application Context interface.

1) ApplicationContext context = new ClassPathXmlApplicationContext(String xmlFileName);

2) getBean("identifier") is used to fetch Bean object from Application Context Spring Container.

Steps to write Spring Program.

1. Create an Project (java Project).
2. Copy & paste spring core 2.0 jars as well as perform java build path.
3. Copy & paste xml file into src floder.
4. Write code in main() method along with Spring Containers.

Note - In xml file, we can create multiple objects But we can't duplicate "id" in xml file.

Scopes of Beans in XML file -

1) <bean> tag allows five important attributes.
   (a) "id" attribute takes reference variable name.
   (b) "class" attribute takes FQCN of object which has to be created.
   (c) "scope" attribute defines behaviour of object creation.
   It takes 2-important values. -
   I) Singleton - we have only one object created. (check by using obj addr)
   II) Prototype - we have multiple instance of object created, for every call of getBean().

Note - "scope" attribute is optional. Hence by default, scope is taken as "singleton".

(d) "init-method" attribute takes method

name which has to be executed after object creation

(f) "destroy-method" attribute takes method name which has to be executed immediately after object destruction

Note - "id", "class" attributes are mandatory for <bean> tag whereas "scope", "init-method", "destroy-method" are optional for <bean> tag.

Dependency Injection -

1) Process of Initializing variables of an class through xml file or annotations is called as Dependency Injection.

2) We have two types of Dependency Injection
   (a) Dependency Injection using Constructor.
   (b) Dependency Injection using getters & setters

a) Dependency Injection using Constructor.

→ To perform this process, we make use of <constructor-arg> tag

→ The Number of <constructor-arg> tag in xml file decides the number of arguments in constructor to get executed.

Example - ~~If~~ <constructor-arg> tag ~~is~~ In case if we use 4 <constructor-arg> tag in xml file then 4-arguments constructor is executed.

Note - If <constructor-arg> tag is not used in XML file, then default constructor is executed.

→ The attributes allowed for <constructor-arg> tag is -

① "value" attribute represents actual content to be intialized for primitive type of Data.

Note - In XML file, It is mandatory to maintain the order of arguments given in constructor which is declared in java class..

② "index" attribute is used to represent index of argument in constructor. It is used only if order of argument is not maintained in XML-file.

③ "ref" attribute represents reference variable name to be given for Non-primitive data type.

Note - 1) To list type in XML file, we make use of <list> tag & for set type in XML file, we make use of <set> tag.

2) within <list> & <set> tag, <value> tag is used to add multiple values.

b) Dependency Injection using getters and setters method -

→ Tag used to perform dependency injection using getters and setters methods is <property name = " "/>

→ The important attributes used with <property> tag is:

① "name" attributes represents Variable name to which values has to be passed

② "value" attribute represents actual content to be initialized for primitive type of data.

Note - Rest of the attribute is same as constructor's dependency injection.

Special Case -

What happens if we perform dependency injection of constructor and setters method for bean?

→

Generates exception if No default Constructor

Default Constructor mandatory for getters & setters.

Conclusion

Programmers prefer setters dependency injection over constructor dependency injection.

Special Case - Working with map interface

Note - map interface works based on key-value pair concept where key is unique & value may be duplicated.

→ In XML file, map interface is represented as <map> tag.

→ Into <map> tag, we add key-value pair by using <entry> tag, which has "key" & "value" has attributes.

→ <map> tag can be used for dependency injection of constructors & setters.

syntax -

```
<map>
    <entry key="" value=""></entry>
</map>
```

Annotations - (2nd Approach)

→ To create object using Annotations, we have to make use of @component above the class.

→ @component ("st"), this indicates st has rv name. → For annotations copy SpringCore-latestVersion

Note - If programmer doesn't provide rv, then Spring Framework follows default bean Id's, which is same as respective class name.

Spring framework follows de

Example - If classname is "Student", then
default name is "student".

Note: <context : component-scan base-package = "current-package-name"> in xml file, which indicates where exactly @Component annotation is present. Also eliminate the need for declaring all the beans in xml

→ In annotations approach, we can pass values objects in three ways;

1. By using variables.
2. By using constructors.
3. By using getters & setters method.

→ @value annotation is used to pass values for primitive type of variable. It is used above the variables.

→ @Autowired annotation is used to pass objects For non-primitive type of variables. It is used above the variables, constructors & setters method.

Note - 1) If @Autowired annotation is used above constructors, then it is referred as "Constructor Dependency Injection" of Annotation Approach.

2) If @Autowired annotation is used above setters methods, then it is referred as "setters Dependency Injection" of Annotation Approach.

→ @Scope annotation represents behaviour of object creation. It is given by "ConfigurableBeanfactory" interface, which consists of two constants.

1. SCOPE_SINGLETON
2. SCOPE_PROTOTYPE.

Note: In java constanst i.e. final variables represents in UPPERCASE.

Syntax -

@Scope (value = ConfigurableBeanfactory.SCOPE_SINGLETON)

@Scope (value = ConfigurableBeanfactory.SCOPE_PROTOTYPE).

→ @Qualifier annotation represents the exact bean which must be given for @Autowired annotation. This is used to avoid confusion only if their are multiple object of same type. (Example - Generalize methods in core java)

Note -
Like @Qualifier annotation @Primary annotation also work, we just need to declare above the class who creates confusion / whoes object we need. It is declared as substitution of @Qualifier.

Note -

→ xml file can be completely replaced by "App class, which performs same functionality an xml file.

→ AppConfig class is represented by @Configura annotation, which is declared above the cla

→ @Bean annotation is used ~~for~~ Spring Conte ~~+s~~ "~~Annotation Config Applic~~ to represent cre of Object, which must be declared above th method along with reference variable nam

→ Implementation class used for Spring Conte is "AnnotationConfigApplicationContext".

→ @PostConstruct annotation represents an method which has to be executed immediately after object creation.

→ @PreDestroy annotation represents a method which has to be executed immediately before object destruction.

Note -

→ To work with Collections using Annotati we have to make use of appConfig file

→ Any Collection Type in java is represe as "Object." Hence we create Collectio objects in the form of methods.

Conclusion -

→ Inversion of Control
(Creating the objects by xml file & annotation)
→ Dependency Injection
→ Spring Containers

Inversion of Control + Dependency Injection +
Spring Containers = Spring Core

* Spring AOP (Aspect Oriented Programming)

Example - PhonePe App.

→ Unlock the App → (New feature)
→ Payment to merchant → (old)
→ enter UPI PIN → (old)
→ transaction is successfull → (old)

→ The purpose of Spring AOP is "to add new feature without affecting old features", which results in Loose coupling.

→ few Terms of Spring AOP is:

① Concern: feature of an Application.
Two types of concerns.
(a) Core concern: Existing feature/old feature.
(b) Cross-cutting concern: New feature.

② JoinPoints: multiple location in program where cross-cutting concern has to be executed.

③ PointCut: Exact location where cross-cutting concern must be implemented

④ Advice: Actual implementation of cross-cutting concern.

⑤ Aspect: Combination Pointcut & Advice.

Example -

Person class	CrossCutting Concern Cla
① eat()	① gym()
② sleep()	② walk()

→ Five Types of Advices are available.

① Before Advice: It is represented as @Before Annotation where CrossCutting Concern is executed first followed by Core Concern.

② After Advice: It is represented as @After Annotation where Core Concern is executed first followed by Cross-cutting concern.

③ After Throwing Advice: It is represented by @AfterThrowing annotation. It is executed only if their is an unsuccessfull execution of core concern.

④ After Returning Advice: It is represented by @AfterReturning annotation. It is executed only if their is an successfull execution of core concern.

⑤ Around Advice - @Around represent that Adv which ensures that an advice can run before & after the method execution.

→ @Aspect annotation represents a class as an cross-cutting concern class.

→ @EnableAspectJAutoProxy represents a link bet^n core concern & cross-cutting concern.

Note: <context: component-scan> tag of xml file can be replaced with scan(), refresh(). scan() considers package names.

Example: money Transfer.

→ due to poor internet connection, Transaction failed.

@ AfterThrowing - Unsuccessfull message.

@ AfterReturning - successfull message.

Note - 1) To store the object of class in ~~AOP~~ we use annotationConfigApplicationContext bcoz it represent xml file.

2) For Spring AOP copy Spring AOP folder of jars.

Syntax:-

```
AnnotationConfigApplicationContext context =
    new AnnotationConfigApplicationContext(classname.class);
```

* Spring - MVC (Model View Controller)

Section-1: View

Section-2: Controller

Section-3: Spring mvc Block (DispatcherServlet, HandlerMapping, ViewResolver)

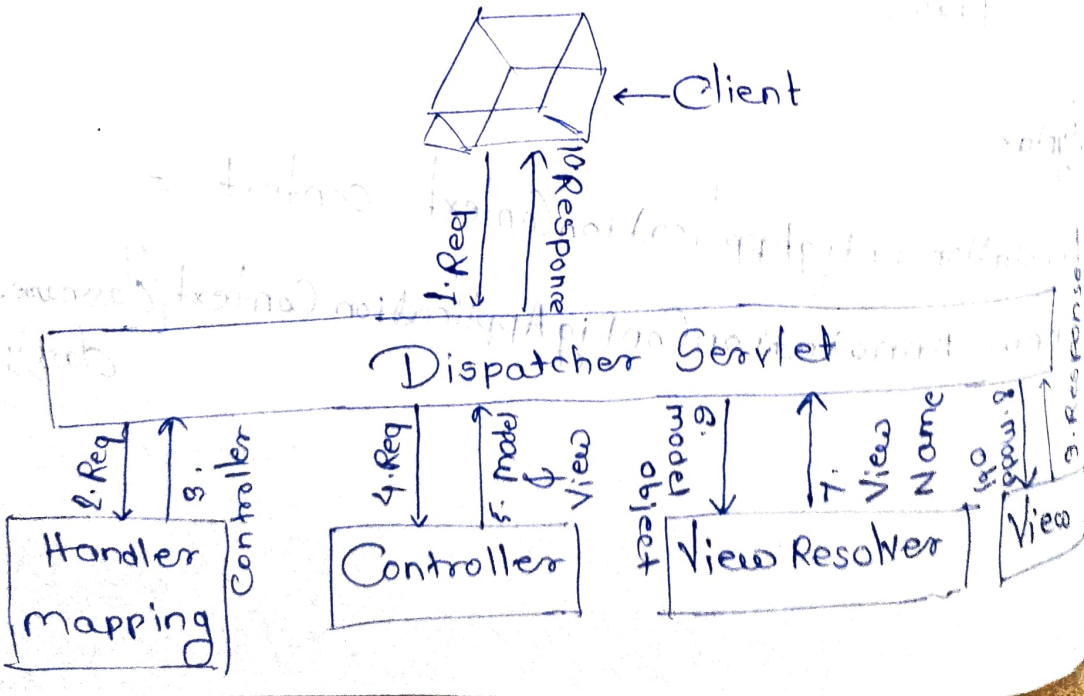
① DispatcherServlet → represented in the form of "web.xml" file

② HandlerMapping → represented in the form of @RequestMapping annotation.

③ ViewResolver → represented in the form of second-xml file

Note: To Generate web.xml file, Right Click on the Project & select "Java EE Tools" & Click on "Generate Deployment Descriptor stub".

Note: for one Dynamic web Project, we can have one web.xml file.



Code for DispatcherServlet on web.xml file is:

```
<servlet-mapping>
    <servlet-name> identifier </servlet-name>
    <url-pattern> /FormActionName </url-pattern>
</servlet-mapping>

<servlet>
    <servlet-name> identifier </servlet-name>
    <servlet-class> FQCN of DispatcherServlet
    <servlet-class> (type in class then copy it)
</servlet>
```

→ <url-pattern> is used for receiving requests from HTML file into Dispatcher servlet & from HandlerMapping

→ <servlet-name> is an "identifier" used to create "second xml file".

Example-1 - If <servlet-name> given as "sample", then second xml file name would be "sample-servlet.xml".

Example-2 - If <servelet-name> given as "dummy", then second xml file name would be "dummy-servlet.xml".

→ Second xml file will be created under WEB-INF folder.

→ WEB-INF folder contains two xml files.

① web.xml

② Second xml file

Note - Every method in spring class must return ("ModelAndView object".

"Model&View object" represents two informations.

① "model" represents content which has to be displayed as o/p.

② "View" represents fileName which is used to display that o/p.

→ "modelAndView object" represents "Incomplete Response".

→ ViewResolver is an interface & its implementation class is "InternalResourceViewResolver" (same like dispatcher copy)

It consists of two important properties.

① "prefix" represents path of file (View)

② "suffix" represents extension of file

Note - Incomplete Responce (modelAndView) is completed by ViewResolver, after adding "prefix" & "suffix".

→ ViewResolver is in second xml file.

Example -

- Task-1 : Addition of two numbers.
- Task-2 : Subtraction of two numbers

Conclusion -

1) Request from HTML file goes to DispatcherServlet (web.xml) with the help of <url-pattern> tag

2) DispatcherServlet communicate with HandlerMapping about particular Spring cls method for execution.

3) Every SpringClass method must return ModelAndView object.

4) ModelAndView object is Passed onto ViewResolver to add prefix & suffix.

5) complete response will be given to DispatcherServlet.

6) Then finally response will be on browser.

Note - 1) DispatcherServlet is referred as "Front-End controller", since he manages request & response.

2) for SpringMVC copy SpringMVCJars folder jars files. & for second xml file use SpringMVCAnnotations_DTD.txt file code in xml file.

Note - 1) <mvc:annotation-driven> is used for enabling the Spring MVC components with its default configurations. 2) would registers the HandlerMapping & HandlerAdapter required to dispatch requests to your @Controllers.

html file → web.xml (Dispatcher Servlet) → Spring class (Controller) (& Handler mapping) → model & View obj → Second xml file (View Resolver) → prefix & suffix to obj → web.xml (Dispatcher servlet) → o/p → Display browser.

→ ① Controller - create a controller class that can handle one or multiple requests.

② Request mapping - used for specifying URL mapping. That maps the URL-patter / login to be handled by the annotated method or class.

Note - ① When this annotation is used at the class level, the class becomes single-action controller. - Write annotation above class.

② When this annotation is used at method level, you can have a multiple action controller.

③ This annotation can be also used for specifying multiple URL patterns to be handled by a single method.

④ Request param - Retrive the value html form.

Note - Spring allows us to return either string or a modelAndView obj from the handler method.

ModelAndView object constructor have three parameters → ① file name — View
② r.v which has to mentioned on second jsp.
③ Content which has to be retur... model

<servlet-mapping> - indicates what URLs will be handled by which Dispatcher servlet.

${message} is the attribute which we have set up inside the controller. You can have multiple attributes to be displayed inside your view.

* Spring MVC with Sessions.

→ @ModelAttribute annotation performs the following functionalities.

① Fetch UserInfo from frontend file.

② Create the respective object of DTO (Data Transfer object).

③ Store/Bind the UserInfo values into DTO object based on names given.

Note - It is used as method arguments

@ SessionAttributes & @ SessionAttribute -

Example -

JSP-file → Name, Age fields

SpringClass 1 → store the values into session

DTO class → user

Spring Class 2 → Retrive the values from session.

→ @ SessionAttributes is used for storing the values/object into sessions. It must be declared above class.

→ @ SessionAttribute is used for retrieving the values/object from session it must be declared as an argument in the method.

N

* Properties file -

→ As a programmer, we use properties file to store the values which may vary.

→ In Properties file, contents are represented in the form of key-value pair.

→ Using key, we fetch the value from Properties file.

Case 1 : Properties file in XML approach.

⟶ <context : property-placeholder> is used to specify the location of properties file by using "location" attribute.

```
<context : property-placeholder location =
    "properties file"/>
```

⟶ From Properties file, to fetch values we make use of ${key} where key is given in properties file.

Case 2 : Properties file using Annotations.

⟶ @PropertySource annotation is used to specify location of properties file.

⟶ ${key} is used to fetch values from properties file.

Spring init method & destroy me

→ In spring, we can use init method & de
method as attribute in bean configuration fi
for bean to perform certain actions upon
initilization & destruction.

→ Alternative to initializing Bean & Disposable Be
interface.

Employee — class
public void init() ⟶ public void cleanup
application.xml
<bean id = "employee" class = "com.ram.model
init-method = "init" destroy-method = "cleanup
<property name = "id" value = "100" /></prop
<property name = "name" value = "peter" /></proper
</bean>

→ import org.springframework.context.Configurab
ApplicationContext;
public class App
{
psvm (s [] args)
{
ConfigurableApplicationContext context =
new ClassPathXmlApplicationContext(
"applicationContext.xml");
Employee employee = context.getBean
"employee", Employee.class);
Sysout (employee);
context.close();
}
}

→ The ConfigurableApplicationContext.close()
will close the application context, releasing
all resources & destroying all cached singleton
beans.

→ The init() is called, after the id & name proper
is set.
The cleanup() method is called after the context
close();

→ Better not to use initilizing Bean & Disposable
Bean interface, because it will tight caupled your
code to Spring.

→ A better approach should be specifying the init-
method & destroy-method attributes in your
bean configuration file.

```
Resource resource = new ClassPathResource
                    ("applicationContext.xml");
BeanFactory factory = new XMLBeanFactory
                      (resource);
factory.getBean("StudentBean");
```