# Spring Class

(Faculty Omkar)

**Created by AZAM AMIR REZA (**azam20104u@gmail.com**)**

30-Dec-15:

**MVC**

* MVC is an n-layered arch
* It makes an application re-usable and loose coupled
* It is the common arch to develop EE applications

Note:

* MVC can be used to create application with multiple layers
* MVC is used when designing framework or application
* MVC will have minimum three layers

**Controller**

Technology’s we are using

1. Servlet
2. Spring
3. etc.

**View**

Technology’s we are using

1. Html
2. CSS
3. Ajax
4. etc.

**Model**

**DAO**

Jdbc, hibernate

**Service**

Spring, java mail, EJB

DB

**View layers:**

* A view is used to write a presentation logic
* A view can interact with client and controller

**Controller:**

* Controller is used to write a request processing and navigation logic
* Controller can interact with view and Service

**Service:**

* A service is used to Business logic
* And it interact with controller and DAO

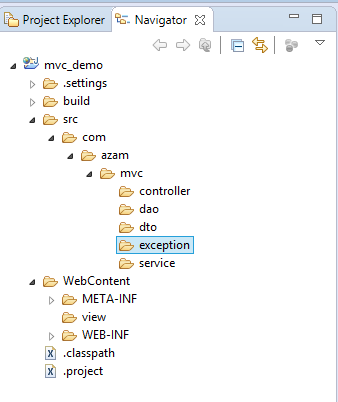
**DAO:**

* Is used to persistence Logic
* And interact with database

**DTO:**

* Is used to transfer the data
* And its interact with all the layers

**Project Structure:**



Example,

**package** com.azam.mvc.dto.login;

**public** **class** **LoginDTO** {

**private** String userid;

**private** String password;

**public** String getUserid() {

**return** userid;

}

**public** **void** setUserid(String userid) {

**this**.userid = userid;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

}

package com.azam.mvc.util;

import com.azam.mvc.controller.login.LoginController;

import com.azam.mvc.dto.login.LoginDTO;

public class **View** {

public static void main(String[] args) {

LoginController controller= new LoginController();

LoginDTO dto=new LoginDTO();

dto.setUserid("userid");

dto.setPassword("password");

controller.checkLogin(dto);

}

}

package com.azam.mvc.controller.login;

import com.azam.mvc.dto.login.LoginDTO;

import com.azam.mvc.service.login.LoginService;

public class **LoginController** {

public String checkLogin(LoginDTO dto){

System.out.println("inside controller...");

//pass data to service

LoginService service=new LoginService();

boolean flage=service.checkLogin(dto);

if (flage) {

System.out.println("valid user");

return "Home";

}else {

System.out.println("invalid user");

return "Failure";

}

}

}

package com.azam.mvc.service.login;

import com.azam.mvc.dao.login.LoginDAO;

import com.azam.mvc.dto.login.LoginDTO;

public class **LoginService** {

public boolean checkLogin(LoginDTO dto){

System.out.println("inside the service");

if (dto!=null && dto.getUserid()!=null&& dto.getPassword()!=null) {

//pass data to dao

System.out.println("to dao");

LoginDAO dao=new LoginDAO();

return dao.check(dto);

}

return false;

}

}

**package** com.azam.mvc.dao.login;

**import** com.azam.mvc.dto.login.LoginDTO;

**public** **class** LoginDAO {

**public** **boolean** check(LoginDTO dto) {

System.***out***.println("inside dao");

//check with data base

**if** (dto.getUserid().equals("userid")&&dto.getPassword().equals("password")) {

System.***out***.println("with DB");

**return** **true**;

}

**return** **false**;

}

}

04-Jan-16:

**Spring**

* Spring is an open source enterprise edition application framework
* Spring is an important integration technology
* Spring is an configuration framework
* Spring implements IOC and DI

Note:

Spring uses java bean specification. Java bean is a class like DTO but can have methods to perform logic

**Steps**

1. Create a java project
2. Create standard package
3. Create lib folder and add spring jars into class path
4. Copy spring configuration file inside source folder
5. Jar files we need
6. Commons-logging
7. Spring-beans-4.2.2
8. Context
9. Core
10. Expression
11. Create a Bean inside bean folder
12. In the xml file for object creation

<bean id=”casio” class=”fully qualified name of Bean class”></bean>

1. In the application class

//container way of creating objec

ApplicationContext container=**new** ClassPathXmlApplicationContext("spring-config.xml");

CalculatorBean fromContainer=container.getBean(CalculatorBean.**class**)

fromContainer.add(4, 6);

05-Jan-16:

IOC (Inversion of Control):

* It is a process of inverting a control to xml entity
* IOC can be implemented as container
* IOC is a container which creates and manages life cycle of a bean object
* In applications we invert to control for two things

1. Creation of object
2. DI (Dependency Injection)

**Application Context:**

* Application context is a container
* Application context creates and manages life cycle of a spring bean
* Application context is a Bean factory

Bean factory

Application Context

Web application context

Note:

Application context is suitable for creating web application as it supports AOP, IN8N, themes etc.

**Spring configuration:**

Ex,

**package** com.azam.spring.bean;

**public** **class** RestaurantBean {

**public** **void** serveFood(String foodItem){

System.***out***.println("Food Item srvred\t"+foodItem);

}

}

ApplicationContext container=**new** ClassPathXmlApplicationContext("Spring-config.xml");

RestaurantBean fromRestaurant=container.getBean(RestaurantBean.**class**);

fromRestaurant.serveFood("Dhosa");

**Spring configuration can be done in two ways:**

1. Xml
2. Java Configuration that is @annotation

**Spring Xml file:**

* Since it is root element in xml file
* We stand is the container for declaring being element
* Bean tag is used to declare a java class for which container has to create and object
* Bean has two attributes this id and class.
* Id attribute is a reference of the bean object created by container. Id attribute must have unique value
* Class attribute is used to declare the class for which object has to be created. Value for class should be fully qualified name of the java class

For initialize the property with the xml file:

First you need to create set method in the Restaurant class

**public** **class** RestaurantBean {

**private** String name;

**public** **void** setName(String name){

**this**.name=name;

}

Then you need to configure in xml

<property name=*"name"* value=*"New Shanti Sagar"*></property>

**Property Tag:**

* This tag is used to set value for a property to set() method
* Property tag has two attributes 1) name and 2) value

1. Name: Value for name attribute must be property name from java class
2. Value: value attribute is used to pass value for the property. Value attribute should be used the data type of the property is a string primitive or wrapper class

09-Jan-16:

**Constructor arg:**

**public** **class** TeaShop {

**private** String name;

**public** TeaShop(String name) {

**this**.name =name;

}

**public** **void** makeTea(){

System.***out***.println("name\t"+name);

}

**package** com.azam.spring.bean;

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

TeaShop ts=**new** TeaShop("azam");

ts.makeTea();

}

}

In XML to set the values through parameterized constructor we use property tag

<constructor-arg value=*"azam"*/>

* This tag is used to pass the argument for the constructor
* Constructor arg pass single argument for the bean
* Constructor arg can be used with <property/> tag

Note:

To initialize property of Bean class we can use <constructor-arg> or <property > tag, depending upon the logic used in the class setter or constructor for the initialization.

**Scope:**

Scope is the boundary which can be used to carry the data

**package** com.azam.spring.bean;

**public** **class** RobotBean {

**private** String model;

**private** **boolean** working;

**public** RobotBean(String model){

**this**.model=model;

}

**public** **void** setWorking(**boolean** working) {

**this**.working = working;

}

}

<bean id=*"tea"* class=*"com.azam.spring.bean.TeaShop"*>

<constructor-arg value=*"azam"*/>

<property name=*"price"* value=*"10"*/>

</bean>

**Spring Bean can five different scopes:**

1. Singleton
2. Prototype
3. Request
4. Session
5. Global

**Note:**

By default spring creates only one object for a bean tag

To change the scope of bean we need to use scope attribute in the bean tag.

Ex, <bean id=*"tea"* class=*"com.azam.spring.bean.TeaShop" scope=”prototype”*>

**Prototype:** It will create a new instance of a Bean whenever referred in an application.

**Request & Session:** Can be used when working with servlet container.

**Global:** It can be used when working with portal application.

**Note:**

Spring avoids developer from writing singleton or factory design patterns

**Dependency Injection:**

* Dependency Injection is a design pattern
* It makes an application light weight
* It is a process of injecting the dependent objects through references by an external entity
* Dependency Injection can be done in two ways

1. Constructor Injection
2. Setter/ Property Injection

Note:

When an object is having reference to another object for functionality, it is referred as dependent object.

**Dependent on Functionality:**

**public** **class** TransportBean {

**public** **void** toDestination(String dest){

System.***out***.println("Reached \t"+dest);

}

}

**public** **class** CommutorBean {

TransportBean tb;

**public** **void** travel(String toTravel){

System.***out***.println("Travel to \t"+toTravel);

tb.toDestination(toTravel);

}

**public** **void** setTb(TransportBean tb) {

**this**.tb = tb;

}

}

**public** **class** Test {

**public** **static** **void** main(String[] args) {

CommutorBean cub=**new** CommutorBean();

TransportBean trb=**new** TransportBean();

cub.setTb(trb);

cub.travel("Hydrabad");

}

}

o/p:

Travel to Hydrabad

Reached Hydrabad

* If the dependent object is referred through a set() method is called setter injection
* If the dependent object is referred through a constructor is called as constructor injection

Ex,

**public** **class** FuelBean {

**public** **void** burn(){

System.***out***.println("burning....");

}

}

**public** **class** VehicleBean {

**private** FuelBean fb;

**public** **void** move(){

System.***out***.println("moving...");

fb.burn();

}

**public** **void** setFb(FuelBean fb) {

**this**.fb = fb;

}

}

**public** **class** VF {

**public** **static** **void** main(String[] args) {

ApplicationContext container=**new** ClassPathXmlApplicationContext("Spring-config.xml");

VehicleBean vb=container.getBean(VehicleBean.**class**);

vb.move();

}

}

In Sprin-config.xml:

<bean id=*"petrol"* class=*"com.azam.dep.FuelBean"*>

</bean>

<bean id=*"bus"* class=*"com.azam.dep.VehicleBean"*>

<property name=*"fb"* ref=*"petrol"*/>

</bean>

**Auto Wire:**

* It is also called as dependency lookup
* It is a process where we ask the container to inject dependency object
* For auto wiring we need to use “autowire” attribute of bean tag.

Ex, <bean id=*"petrol"* class=*"com.azam.dep.FuelBean"/*>

<bean id=*"bus"* class=*"com.azam.dep.VehicleBean" autowire=”byType”*>

</bean>

11-01-16:

* Auto wire can be done for both constructor and setter injection
* Auto wire constructor works only for constructor injection
* Default value for auto wire is no.

**Three type of Auto wire:**

1. By Type
2. By Name
3. Constructor

**By type:**

* Auto wire byte type will match dependent type with other object type
* Auto wire byte type must have unique object for dependency injection

**By Name:**

* ByName is injected by matching property reference name with bean id
* ByType and byname works only for property injection

Note:

Auto wire can’t be done for properties of type String primitive or wrapper class

**Spring java configuration:**

* Configure @annotation we need to use a tag spring xml

Syntax,

<context: component-scan base-packag=”fully qualified of package”/>

**@Component:**

* It is used to create a bean object from spring container
* @component should be placed about the flash definition.

**@Value:**

* This is used to pass value for property with data type spring primitive or wrapper class

**Auto wired:**

* Is used to inject dependent object by container and auto wired by default is byType
* @Qualifier is used to inject dependent object byName;
* @autowire works property injection and constructor injection

13-Jan-16:

Life cycle of spring web application:

@Component

@RequestMapping(value="/")

**public** **class** ReachargeController {

@RequestMapping(value="/rc")

**public** String recharge(@RequestParam String email, @RequestParam String msg){

/\*String email=request.getParameter("email");

String msg=request.getParameter("msg");\*/

System.***out***.println("message send to \t"+email);

**return** "/success.jsp";

}

}

14-Jan-2016:

**Architecture of MVC Spring:**

**2**

Handler Mapping

View Reserver

DC

AC

Client

**1** 3.Controller

C1 

req

resp

C1 

C1 

**Dispatcher Controller:**

* Dispatcher servlet: it is implementation of front controller design pattern
* Dispatcher servlet is the entry and exit for the application
* At the initialization of dispatcher servlet its load spring configuration file and instantiate application context.
* Dispatcher servlet is responsible for sending request to the controller
* Dispatcher servlet its sends response to the client

**Note:** dispatcher servlet is a servlet

**Handler Mapping:**

* Handle mapping is the helper component to find controller or dispatcher servlet.
* Handle mapping takes URL and returns the name to dispatcher servlet

**Controller:**

* Controller is helper component of dispatcher servlet which process request for dispatcher servlet.
* Controller takes request and can return Model, view name to dispatcher servlet.

**View Resolver:**

* It is a helper component for Dispatcher
* View resolver takes view name and returns to dispatcher servlet

@annotation:

1. @Component
2. @RequestMapping
3. @Value
4. @Autowired
5. @RequstParam

**@Model Attribute:**

* It is used to convert request parameters to a DTO object.

@RequestMapping(value="/sendWithModel")

**public** String sendMsgWithModel(@ModelAttribute MsgDTO dto){

System.***out***.println("exam of model attribute");

System.***out***.println("email\t"+dto.getEmail());

**return** "/success.jsp";

}

**Model and View:**

* Model and view is used to send data and view name to dispatcher servlet
* Data in model and view object will be in request scope
* To view data we should use express language

${senderEmail}

Task

Register page

@com

@req

@model attri

@Model and view use this thing

23-Jan-16:

SPRING ARCHITECTURE:

M

V

C

J

E

E

ORM

A

O

P

JDBC

I O C

NOTE:

* Spring is an important integration technology
* In spring there are 6 modules
* Spring IOC is the core module which is required for a spring application others modules or optional and can be used to integrate other technologies like Hibernate, Struts etc.

Note: Spring is not an ORM tools

You need more 4 jar files apart from 18

1. spring-orm-4.2.2.RELEASE
2. commons-dbcp-1.4
3. spring-dao-2.0.3
4. jta-1.1.jar