**Spring Framework 😊**

**SPRING CORE**

Setting Spring Environment

Spring main part is IOC (INVERSION OF CONTROL) and Dependency Injection.

Spring IOC container takes the POJOS and the configuration metadata to make the application ready to use.

And this configuration meta data we can give to container by three ways as:

* XML file
* Annotation Based
* Java Source code

**Beans**

Scopes of the beans:

* Singelton – Every time a get beans request hits ioc gives the same bean. Single bean in IOC,

Scope=”singelton”

* Prototype – Every time a get beans request hits ioc gives newly created bean.

Scope=”prototype”

Beans Callbacks:

After instantiation a bean needs to be some initialization and before destroy there need some cleanup.

Initialization callback – void afterPropertyset() method defined in IntializingBean interface. Init-method=”init”

Destroy callback – void destroy() method defined in DisposableBean. Destroy-method=”destroy”

We don’t need to implement these callback interfaces we can make method by any name but having the same signature as in the interface:

Signature is:

Void methodName(){} //with no argument

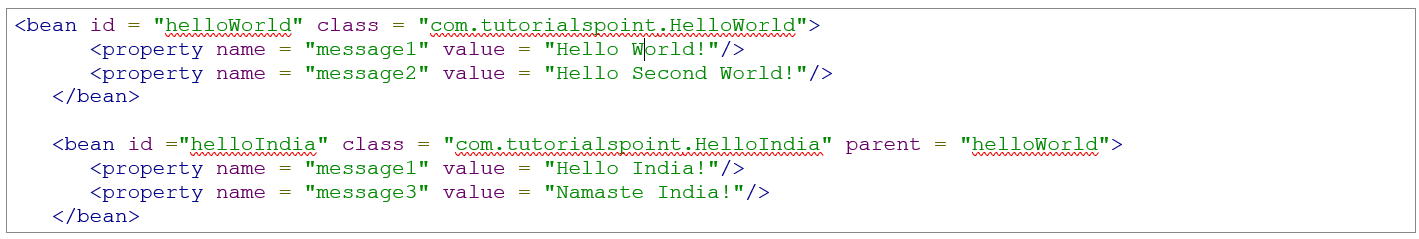
If we are having lots of beans having the same name callback methods the we can define default call back methods in xml.

default-init-method=”myInit”

default-destroy-method=”myDestroy”

**Inheritance of Bean Definition**

Suppose a bean is defined in xml configuration meta data and as we know the bean definition contains lot of values as constructor argument, property values,intial method etc. One a bean definition is defined this can be inherited by the child bean by the parent attribute



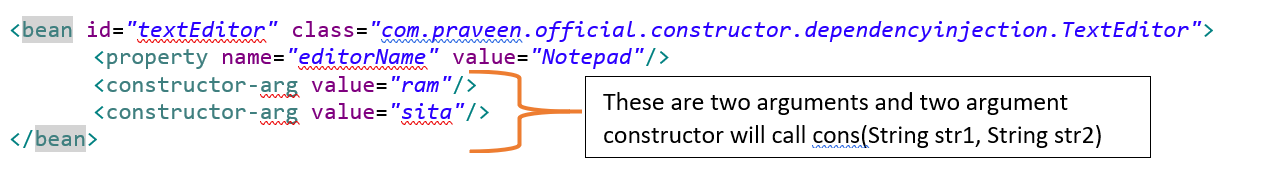
Inheritance of properties values:

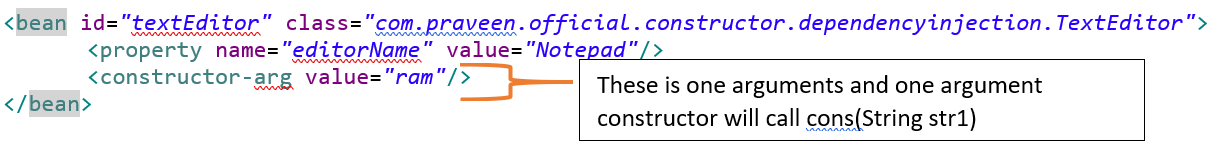
If a bean is have a parent and acting like a child then the child bean should have all the properties of parent bean with their setter method otherwise PropertyNotWritable Exception will come.

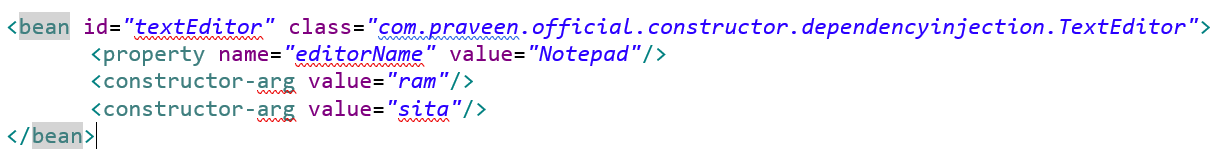
We can define a parent bean and attach it to a child bean but also can make a abstract bean and can attach to child bean. By attribute:

Abstract=”true”

**Constructor Overloading through XML**

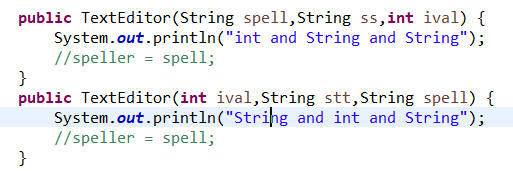






**🡪Constructor Argument Resolution**

* Order of writing constructor args line in XML does’t metter as written in constructor args in bean java.



But the order is different and also different in java code

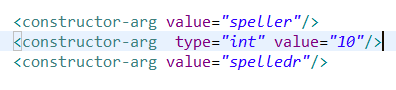
String type = 2

Int type = 1

String type = 2

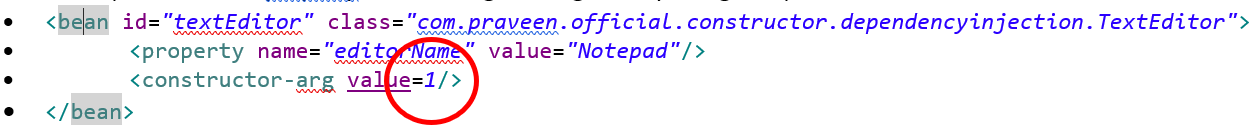
Int type = 1

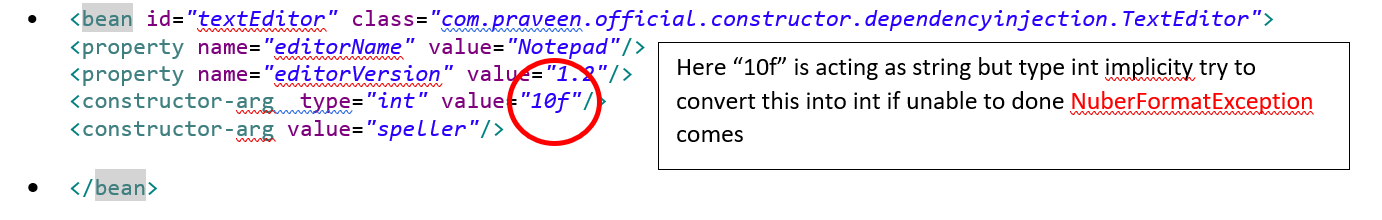
Int this situation the spring will run the method which will come first in order of writing. i.e., the above one.



This order does not matter

* In the parameters of property we can not give integers only string accepted





* **Type attribute**

If we did’t mention any thing and just write value=”takeOnlyInQuotes” , first of all it takes only value in quotes.

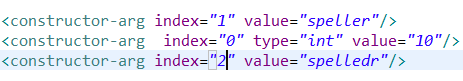
If in java code there is int parameter and we passed anything in dual quotes then it will try to cast in int if happed then ok otherwise Number format exception will come.

We can explicitly define a value that of what type it is of.

Example: 

* **Index attribute**

We can define index attribute to order constructor arguments.

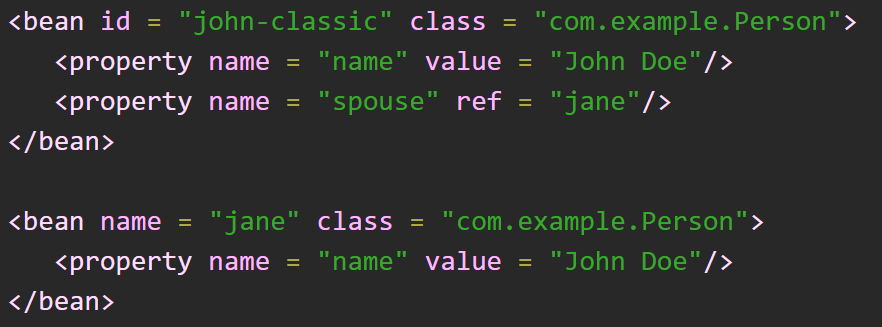


**A final note, in case you are passing a reference to an object, you need to use ref attribute of <constructor-arg> tag and if you are passing a value directly then you should use value attribute as shown above.**

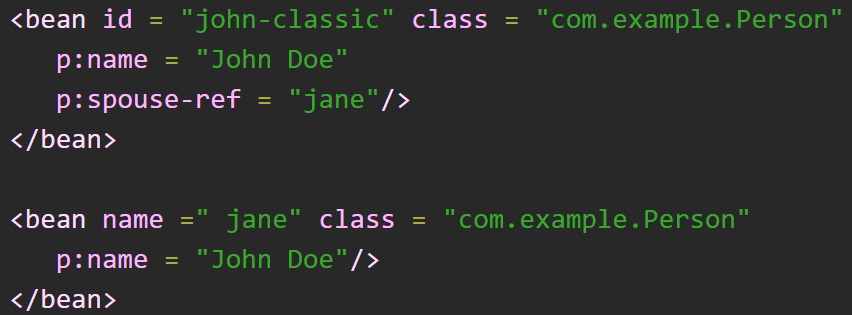
**XML Configuration using p-nameSpace**

P name space:

xmlns:p = "http://www.springframework.org/schema/p"

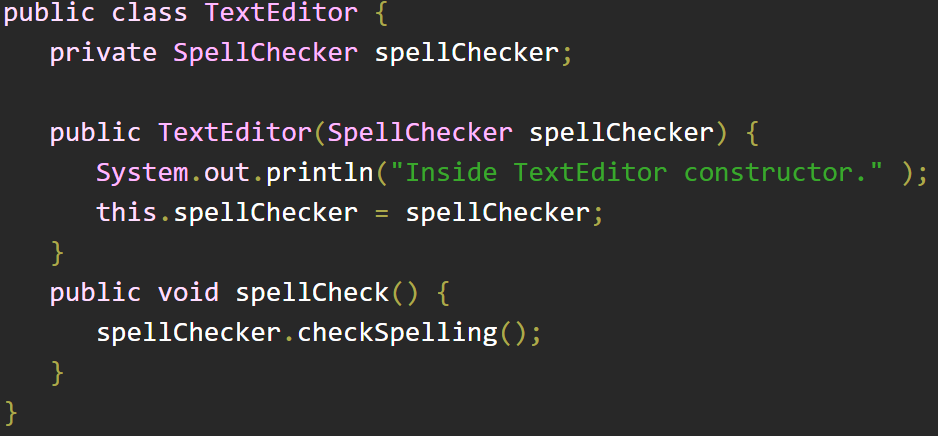


p:objectRef-ref=”classId”

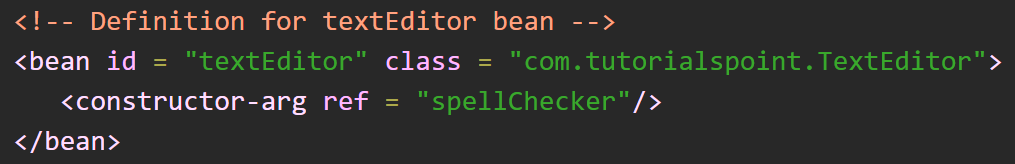


**Dependency Injection**

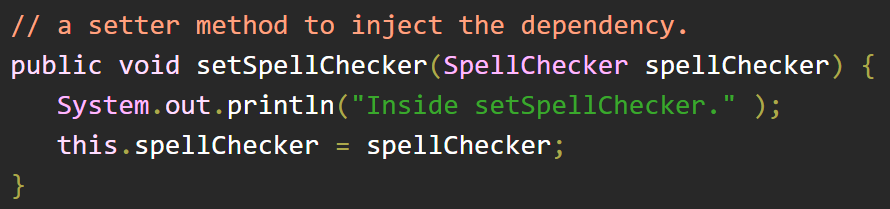
**By constructor method:**



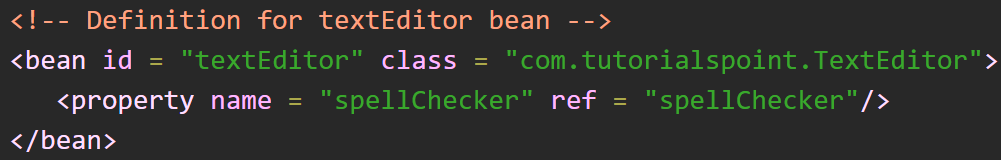
**Bean definition for constructor injection**



**By setter method:**

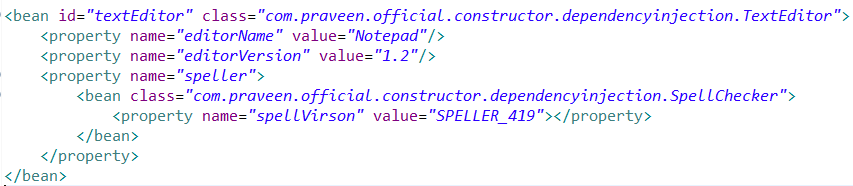


**Bean definition for setter injection**



**INNER BEAN**

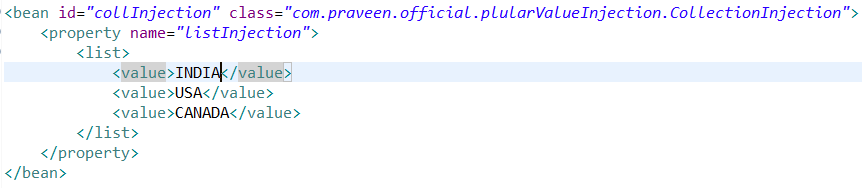
**As we have java inner class declaration similarly, we have inner bean definition in spring.**

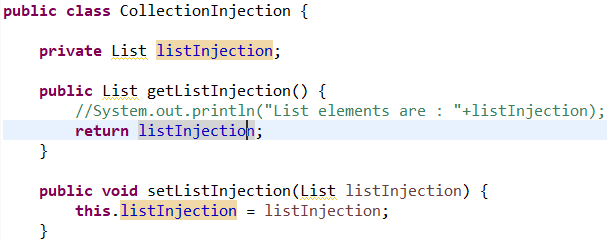


**We can also give reference in constructor by inner bean.**

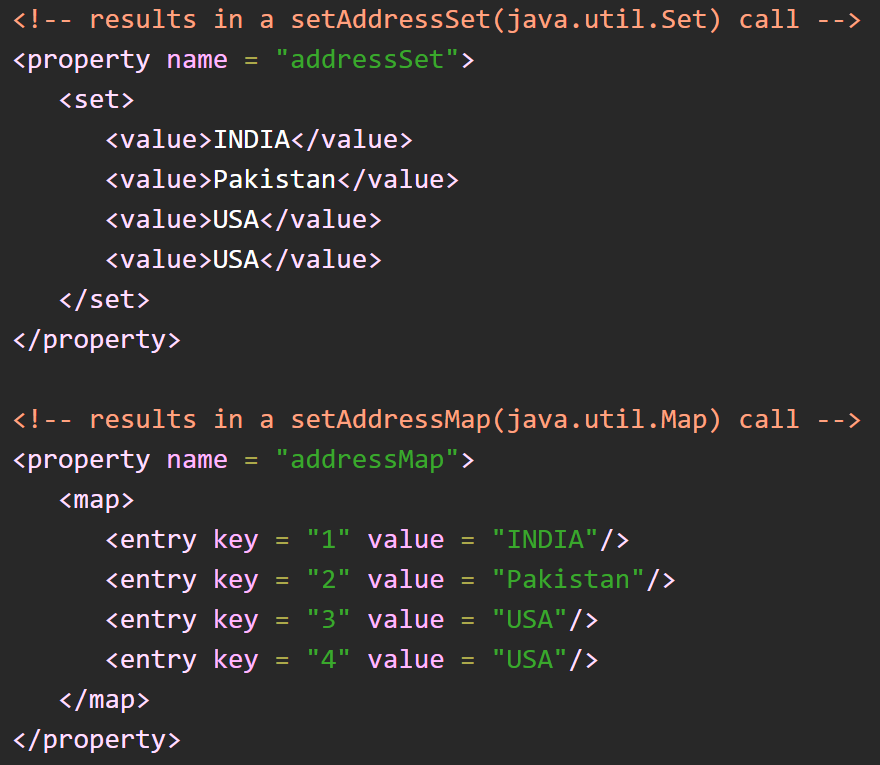
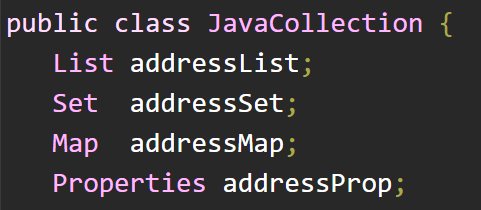
**Plural Value Injection (Collection Injection)**

**As we put values in primitive data type with single values similarly, we can also put collection values.**





**Similarly we can put values of list map set and properties data type**



**Autowiring**

**In spring autowiring the spring ioc container automatically wires the dependency a bean have.**

**It can be done in 4 ways**

* **byName**
* **byType**
* **constructor**
* **autodetect**

**ByName**

In by name the spring container searches for the setter methods of the bean property.

By name works upon setter method.

**Autodetect**

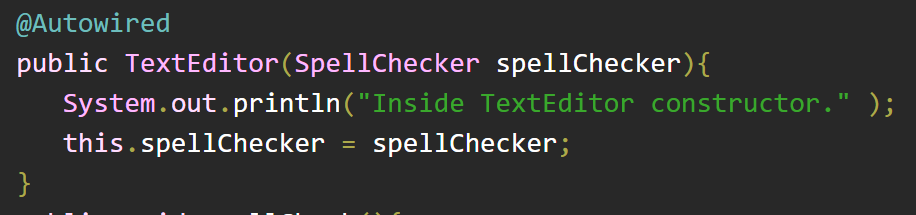
Spring first tries to wire using autowire by *constructor*, if it does not work, Spring tries to autowire by *byType*.

**Annotations**

**@*Required***

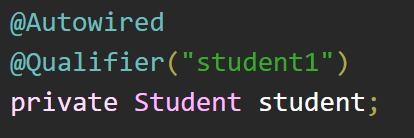
* Put on setter method
* We cannot put this annotation on properties when need to put it on setter methods only.

**@*Autowired***

* Put on setter method
* When container finds it on property it do byType dependency injection
* We can also put this annotation on property instead of setter methods.
* We can also put this annotation on constructor for bean dependency injection
* 
* By default autowired is set to true.
* And if we did’t create the bean which we suppose to autowire in main class then an execpttion will pop up that no bean found which this blab la name
* We can also set autowired to required false – in this case if bean will available then container will inject this otherwise it will not generate this.

**@*Qualifier***

* Suppose we are autowiring a bean and we know that autowire works upon byType injection if with the same type if ioc container will have two beans there will be an ambiguity that which bean that need to inject as dependency. To overcome this we can specify the bean name in the qualifier annotation



***JSR-250 Annotation***

* Spring also have jsr 250 annotations as

**@PostConstruct**

**It is similar to init method()**

**@PreDestroy**

**It is similar to destroy method()**

## @Resource Annotation

You can use **@Resource** annotation on fields or setter methods and it works the same as in Java EE 5. The @Resource annotation takes a 'name' attribute which will be interpreted as the bean name to be injected. You can say, it follows **by-name** autowiring semantics as demonstrated in the following example −

package com.tutorialspoint;

import javax.annotation.Resource;

public class TextEditor {

private SpellChecker spellChecker;

@Resource(name = "spellChecker")

public void setSpellChecker( SpellChecker spellChecker ){

this.spellChecker = spellChecker;

}

public SpellChecker getSpellChecker(){

return spellChecker;

}

public void spellCheck(){

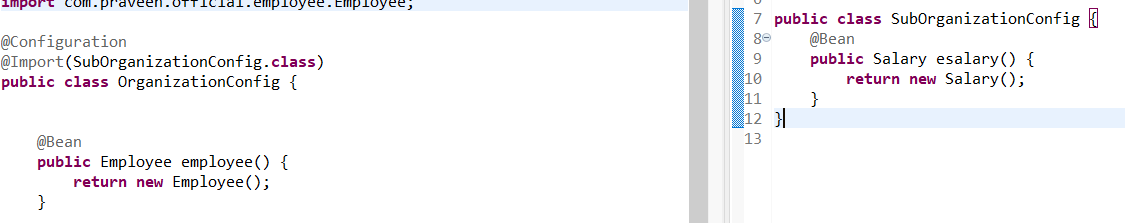
spellChecker.checkSpelling();

}

}

* We cannot put this annotation on properties when need to put it on setter methods only.
* **@*Import(OtherConfig.class)***

We can use import config for importing bean definition by another configuration class.



# **Event Handling in Spring**

**Spring CORE Points…**

1. if bean is not created and we called “context.getBean()” a exception comes
   1. Exception in thread "main" org.springframework.beans.factory.NoSuchBeanDefinitionException: No bean named 'heloWorld' is defined
2. if we have a property in a bean without having of its setter method and we setted bean property in xml then NOTWRITABLEPROPERTYEXCEPTION will come. And if we don’t assign any value in xml file then no exception will come and the null values will be assign to all properties.
   1. Error setting property values; nested exception is org.springframework.beans.NotWritablePropertyException
3. Setting empty string values as parameter
   1. <property name=”email” value=””></property>
4. Setting null values as parameter
   1. <property name=”email”></null></property>
5. If we have any parameterized constructor and do not defined default constructor the exception will come : no default constructor found; nested exception is java.lang.NoSuchMethodException and if we try to creating a bean then nested exception is org.springframework.beans.BeanInstantiationException will come.
6. In context.getBean(“beanName”); the bean Name is case sensitive.

**Need to Understand More in CORE**

* registerShutdownHook()
* Injecting Bean References, setter method in such a way that it can take both references as well as values.
* You can load various configuration classes as follows −

public static void main(String[] args) {

AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext();

ctx.register(AppConfig.class, OtherConfig.class);

ctx.register(AdditionalConfig.class);

ctx.refresh();

MyService myService = ctx.getBean(MyService.class);

myService.doStuff();

}

* Spring core event handling

**CORE REFERANCES**

**Steps to Start with Spring Project:**

What we need to run and test only spring core components

* A java project.
* Spring all jars to the class path of the project, add by go to project build path and add external jars.
* Commons logging libraries to show core output on console window otherwise we need to add a tomcat server to run on.

MVC

**SPRING MVC**

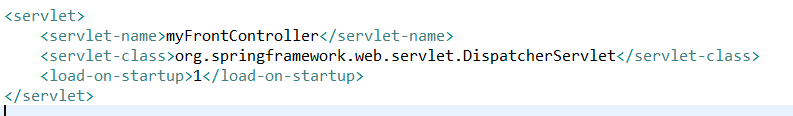
Here MVC stands for Model View Controller.

There are three main thing to start with spring mvc project.

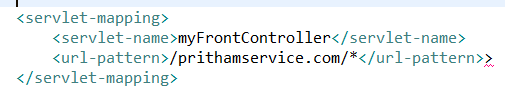
1. Spring jars in web-inf/lib folder with 2 commons.logging jars
2. A web application server to run our web application. i.e., tomcat
3. A dynamic web project.

**In spring mvc, suppose a client puts the request to the server then this request will be handled by the front controller we redirects the client request to the controllers and gets the job done.**

**We don’t need to create front controller because the spring gives it to us with the name “Dispatcher Servlet”.**



**And we can map this servlet to the request urls like**

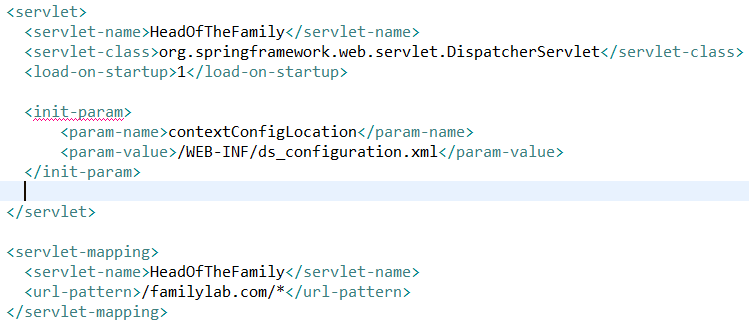


**When we will start this project, spring will load the web.xml file where our servlet is defined with “*load on startup tag”*. when this dispatcher servlet will get initialized before this it will find the dispatcherservletname-servlet.xml file in web-inf/ folder.**

**If spring will find it then our servlet will be initialized successfully.**

**The dispatcher servlet need a xml file to initialized itself so it finds the file with name “anyservletname-servlet.xml” in web-inf folder. If it found the file then good.**

**If you want to change the default file name of DS then we can do and link this name to DS in web.xml as below.**



**LIFE CYCLE OF SPRING MVC PROJECT**

**Whenever our tomcat web server starts it looks for the web.xml file in web-inf folder.**

**And in web.xml we can defined our dispatcher servlet while initializing DS the server will see the DS-servlet.xml file in web-inf folder. This is actually spring configuration file.**

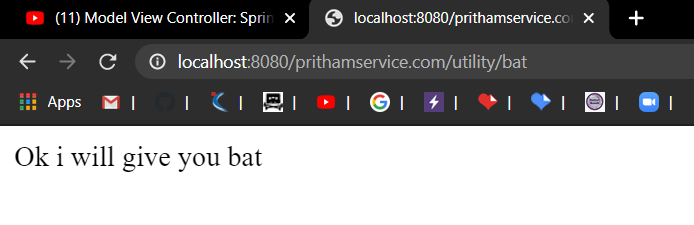
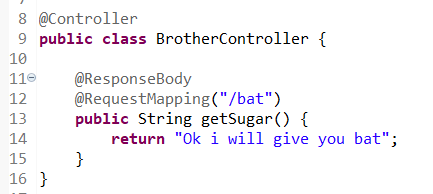
**When server will find DS-servlet.xml file then server will make a container named as WebApplicationContext by the file definition.**

**All the url mapping servlet will hold and on the basis of coming request it will decide which method to call.**

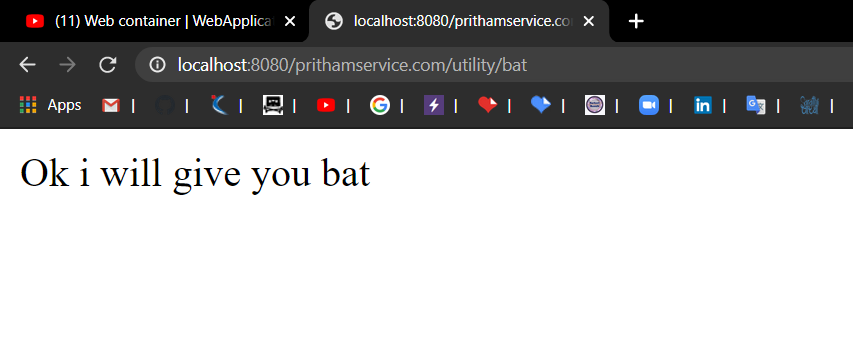
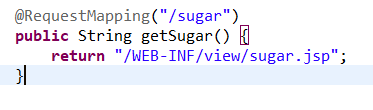
**These mapping are case sensitive…….small and capital matters**

**Views about Views**

**The controller methods always returns view name to present to client browser but if we put @Responsebody annotation over method body then this converts view name to simple string and returns it to clients browser.**



**But if we did’t put @Responsebody annotation then this controller will give view name as default.**



Here we are giving fully qualified name in our code means if in future there is change in directory structure or a technology change then we need change in every controller wherever we use this fully qualified path.

To encounter this situation, spring give us “view resolver” class.

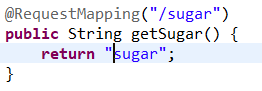
Class name in -> InternalResourceViewResolver extends ViewResolver

**View Resolver**

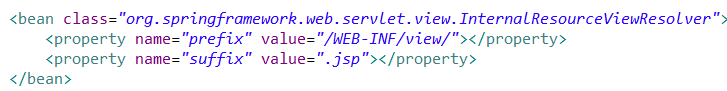
As we saw the controller gives fully qualified view location to the dispatcher servlet and then ds gives that view to the client browser.But there is ambiguity as we saw above.

To resolve we will use the view resolver.

Now the controller will only give the view name without extension and directory path.

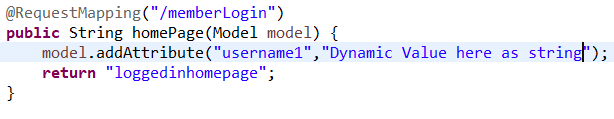


Now dispatcher servlet will give this name to view resolver and the VR will find this view and will give fully qualified view path to DS then DS gives it to client’s browser.

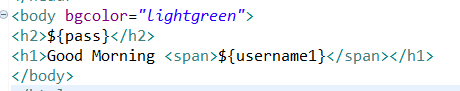


**Model**

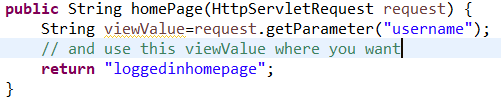
**As we are returning view here, and with the view We can also sends data with the views on the browsers.**



**And this model value we can capture on the jsp page like this…**



**This is above we showed that we can transfer data from controller to view and now we will show that we can also send the data from view (taken by user) to the controller.**



**And here respective jsp**



**Spring MVC java base configuration**

**Spring MVC Points…**

1. Difference between @component and @controller

**Need to Understand More MVC**

* registerShutdownHook()
* Injecting Bean References, setter method in such a way that it can take both references as well as values.
* You can load various configuration classes as follows −

public static void main(String[] args) {

AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext();

ctx.register(AppConfig.class, OtherConfig.class);

ctx.register(AdditionalConfig.class);

ctx.refresh();

MyService myService = ctx.getBean(MyService.class);

myService.doStuff();

}

* Spring core event handling

**REFERANCES MVC**

**Steps to Start with Spring Project:**

What we need to run and test only spring core components

* A java project.
* Spring all jars to the class path of the project, add by go to project build path and add external jars.
* Commons logging libraries to show core output on console window otherwise we need to add a tomcat server to run on.

**MVC PROJECT SETUP**

**First goto eclipse or sts then right click in project creation section and choose maven project**

**Then next window will be project archetype then choose “*maven-archetype-we*” as shown below**

