INDEX

INTRODUCTION TO STRING

Q. What is Spring?

Spring is framework of java which is used for develop any kind of application.
Means using spring framework we can develop the console base application, database application, web application, enterprise level application etc
Means we can say spring name suggest us you can develop any kind of application by using spring framework.

Q. Why use spring framework or what are the benefits of spring framework?

1. It helps us to manage the dependency injection and inversion of control
2. Able to develop any kind of application
3. Huge library or able to integrate with any other framework or library
4. Easy for testing

Etc.

Q. What is dependency Injection?

Dependency injection means if particular method or class or constructor is dependent on some specified parameter called as dependency injection.

Q. What is inversion of control?

Inversion of Control is a principle in software engineering which transfers the control of objects or portions of a program to a container or framework. We most often use it in the context of object-oriented programming.

TYPES OF DEPENDENCY INJECTION

1. Setter injection

setter injection means if we have setter method which is dependent on specified parameter called as setter injection and the best example of setter injection is POJO class.

```
class Employee
{  private int id;
   private String name;
   private int sal;
  public void setId(int id)
   { this.id=id;
   }
  public int getId(){
    return id;
  }
  public void setName(String name)
   { this.name=name;
   }
  public String getName(){
     return name;
  }
  public void setSal(int sal)
    {this.sal=sal;
    }
  public int getSal(){
    return sal;
  }
}
```

setter injection

```
Employee emp= new Employee();
emp.setName("ABC"); //here we can say setName()
is depedent on string type of data means we cannot call setName() method
without parameter so we can say it is setter injection.
```

2. Constructor injection

constructor injection means if class constructor is dependent on parameter called as constructor injection shown in following code.

```
class Employee
{    Employee(String name,int id,int sal)
     {
     }
}
```

it is constructor dependency

```
Employee emp = new Employee("ABC",1,10000); //must be pass parameter
//when we create object of class.
```

3. Object dependency

object dependency means if particular method or constructor is dependent on specified object called as object dependency.

```
class Employee{
   private int id;
   private String name;
   private int sal;
 //setter and getter methods
}
```

```
class Company
{
    public void setEmployee(Employee employee)
    {
    }
}
```

Note: if we think about setEmployee() methtod we cannot call this method without employee object means we can say this method is 100% dependent on Employee object so we can say it is object dependency .

4. Collection Dependency

Collection dependency means if we pass collection as parameter in function or in constructor called as collection dependency.

```
class Employee
{   void setEmployees(List<String> empNames)
     {
     }
}
```

Note: if we think about above code we can say setEmployees() method is 100% dependent on List Collection so we can say it is called as collection dependency

5. Map dependency

Map dependency means if we pass Map as parameter in method or constructor called as Map dependency.

```
class Company
{

    void setEmployeeList(Map<Integer,String> map)
     {
     }
}
```

Note: if we think about setEmployeeList() method we must be pass map as parameter without map we cannot call this method means this method is dependent on Map so it is called as Map dependnecy

6. Property dependency

If we pass property class as parameter in method or in constructor called as Property dependency.

IF WE WANT TO WORK WITH SPRING FRAMEWORK WE NEED TO KNOW SOME IMPORTANT MODULES OF SPRING FRAMEWORK

1. Spring Core

Spring core module helps us to design console base application and provide the all Features and concept implemented of core java by spring framework.

2. Spring DAO

Spring DAO stands for Data Access Object and using this module we can develop the application spring with database using JDBC concept.

3. Spring MVC

Spring MVC is module using this module we can develop the web application by using spring framework with the help of Model View and controller concept.

4. Spring ORM

Spring ORM (Object Relationship Mapping) is module and using this module we can connect your spring application with database by using ORM concept or using hibernate.

5. Spring AOP

We can develop the application using Aspect Oriented Programming approach with the help of spring.

6. Spring Security

This module provides security API to us for application.

NOW WE WANT TO SEE HOW TO DEVELOP THE APPLICATION BY USING SPRING CORE MODULE.

1. Open eclipse

2. Create Maven Project

Q. What is maven?

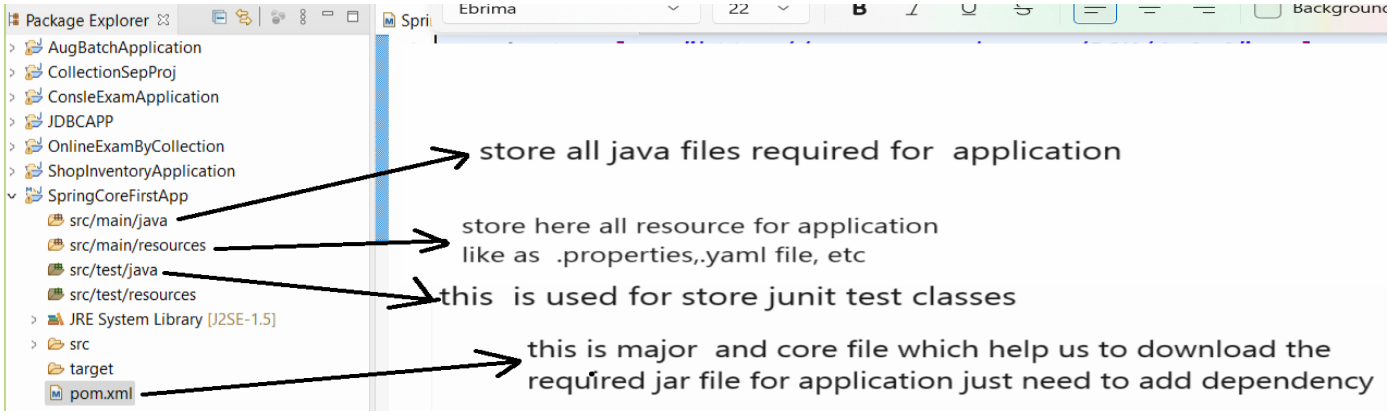
Maven is POM base tool or project object model tool which helps us to download the required APIS for application or Library or .jar file required for application just we need to add dependency in POM.XML file

Note: when we work with maven project then your machine must be connected with internet.

Steps to create maven project

File -- new -- select maven option --- select maven project --- click on next button --- select create sample project --- click on next button --- give group id (group id indicate

package name) --- give artifact id (artifact id means project name) and click on finish button



Dependency

Dependency just XML tag which is used for download the required libraries from internet and put in java application shown in following diagram.

Once we create maven project and if we want to work with spring core application we need to add following dependencies.

a) Spring core.

b) Spring context.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.techhub</groupId>
  <artifactId>SpringCoreFirstApp</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <dependencies>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.3.27</version>
</dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
```

```
   <version>5.3.27</version>
</dependency>
  </dependencies>
</project>

//loose coupling, interface, collection, wrapper classes, POJO class
```

3. Create POJO class

```
package org.techhub;
public class Employee {
	private int id;
	private String name;
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public int getSal() {
		return sal;
	}
	public void setSal(int sal) {
		this.sal = sal;
	}
	private int sal;
}
```

4. Create Bean XML Configuration file

When we work with spring for achieve dependency injection and inversion of control we required to inject the spring bean or configure spring bean or POJO class as spring bean class

THERE ARE TWO WAYS TO CONFIGURE SPRING BEAN CLASS

a) Using XML

b) Using Annotation

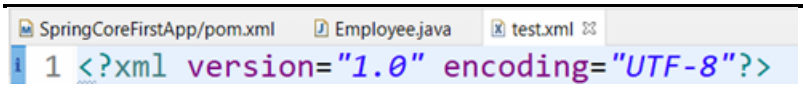
Now we want to configure spring bean class using XML

Steps to configure spring bean using XML

a) Create XML file under src/resources folder

Right click on src/resources' folder ---- select new --- other --- select XML option ---- select xml file ----click on next --- next and finish.

Example: test.xml



b) Copy the required DOC type for bean configuration

Q. What is DOC type?

Doc type it like as header file in c language because XML is custom tag language means developer can design own tags using XML and can reuse it so if we want to reuse some tags then we required to create doc type for that tag means doc type contain all information of user tags so when we want to reuse tag then we required copy the doc type and put top on xml file and if we think java framework then every java framework XML doc type present in his .jar files.

Steps to copy the doc type

If we think spring core your doc type present here.
open the project --- maven dependencies --- open spring-beans.jar ---- open the package org.springframework.beans.factory.xml ----- open the file name as spring-beans.dtd --- copy the doctype from commented part and paste in XML file.

This is your doc type use in XML file

```
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                    "https://www.springframework.org/dtd/spring-beans-2.0.dtd">
```

c) Write the following tags for configure POJO class as bean class.

If we want to configure your POJO class as bean we have to use following tags.

```
<beans>
  <bean id="ref"  class="classname">
    <property name="propertyname" value="value for property"/>
  </bean>
</beans>
```

Here

<bean>: Indicate class as per the rule of java.
id=”ref”: This attribute indicate reference of POJO class.
class: This attribute indicates name of class whose bean or object want to create.
property: Indicate setter method.
name: Indicate name of method which we want to use or field name whose setter method want to create.
value: Value indicate actual parameter which we want to pass.

test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                    "https://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
  <bean id="e" class="org.techhub.Employee">
      <property name="id" value="1"/>
      <property name="name" value="ram"/>
      <property name="sal" value="10000"/>
  </bean>
</beans>
```

6. Create Client Application

ClientApplication means class where we call the XML file and call bean instance here or ClientApplication means class which contain main method from user provide input and get results.

Steps to work with client application.

1. Create reference of Resource interface

Here Resource indicate XML file where we configure all bean classes.

Syntax: Resource r = new ClassPathResource(String xmlFileName);

Example

```
package org.techhub.xmllifecycle;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
public class ClientApplication {
```

```
	public static void main(String[] args) {
		Resource r=new ClassPathResource("test.xml");
	}
}
```

2. Create Reference of BeanFactory

BeanFactory is spring container.

Q. What is spring container?

Spring container is internal API which is used for or which help to maintain all activity of spring framework like as bean object creation, perform dependency injection as well as manage life cycle bean object etc.

THERE ARE TWO TYPES OF SPRING CONTAINER?

a) BeanFactory b) ApplicationContext

So, if we want to create reference of BeanFactory we have class name as XmlBeanFactory means BeanFactory is interface internally and XmlBeanFactory is an implementer class of BeanFactory.

Syntax: BeanFactory ref = new XmlBeanFactory(Resource);

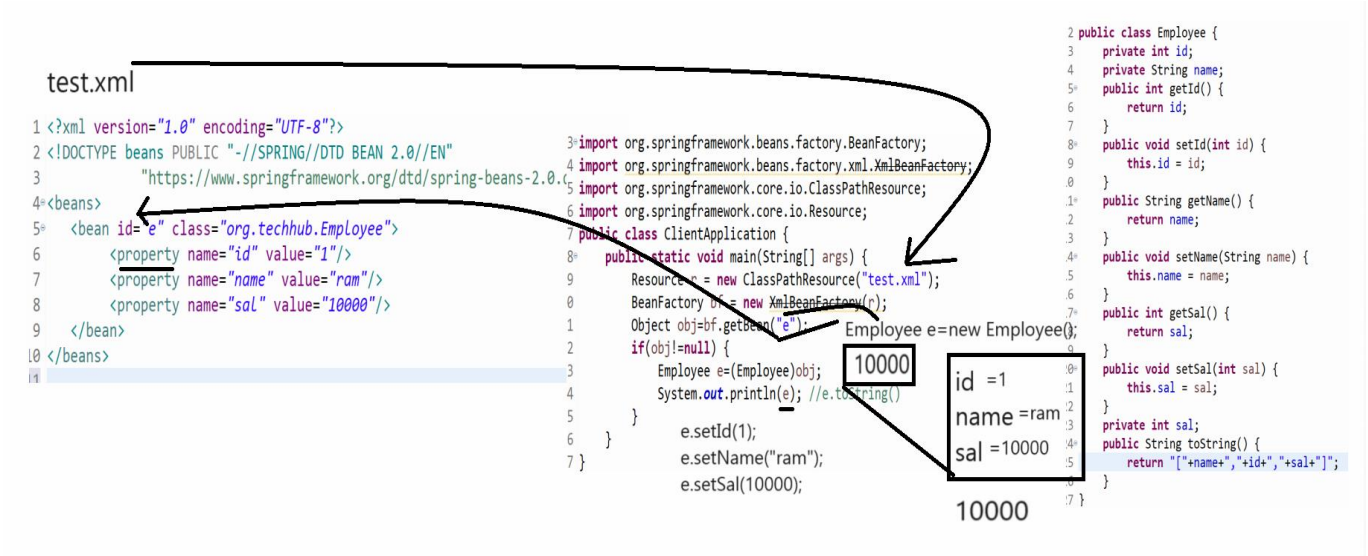
Example

```
package org.techhub;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
public class ClientApplication {
	public static void main(String[] args) {
		Resource r = new ClassPathResource("test.xml");
		BeanFactory bf = new XmlBeanFactory(r);
	}
}
```

3. Call its getBean() method

getBean() method is used for create instance of bean of class or object of bean class and call setter method internally means getBean() is used for perform setter injection by default and return object of bean whose id we pass in it.

Syntax: Object getBean(String beanid);



Source code Example

test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                "https://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
   <bean id="e" class="org.techhub.Employee">
        <property name="id" value="1"/>
        <property name="name" value="ram"/>
        <property name="sal" value="10000"/>
   </bean>
</beans>
```

POJO class

```
package org.techhub;
public class Employee {
        private int id;
        private String name;
        public int getId() {
                return id;
        }
        public void setId(int id) {
                this.id = id;
        }
```

```
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public int getSal() {
		return sal;
	}
	public void setSal(int sal) {
		this.sal = sal;
	}
	private int sal;
	public String toString() {
		return "["+name+","+id+","+sal+"]";
	}
}
```

ClientApplication.java

```
package org.techhub;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
public class ClientApplication {
	public static void main(String[] args) {
		Resource r = new ClassPathResource("test.xml");
		BeanFactory bf = new XmlBeanFactory(r);
		Object obj=bf.getBean("e");
		if(obj!=null) {
			Employee e=(Employee)obj;
			System.out.println(e); //e.toString()
		}
	}
}
```

Constructor Injection

Constructor Injection means when we pass parameter to class constructor by using spring container called as constructor injection.

Note: when we have default constructor in class then we not need to perform constructor injection.
It is automatically performed by spring container.

Source code

test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                    "https://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
  <bean id="t" class="org.techhub.Test"></bean>
</beans>
```

Test.java

```
package org.techhub;
public class Test {
        public Test() {
                System.out.println("I am default constructor");
        }
}
```

ClientApplication.java

```
package org.techhub;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
```

```
public class ConsClientApplication {
        public static void main(String[] args) {
          Resource r= new ClassPathResource("test.xml");
          BeanFactory bf= new XmlBeanFactory(r);
          Test t1=(Test)bf.getBean("t");
        }
}
```

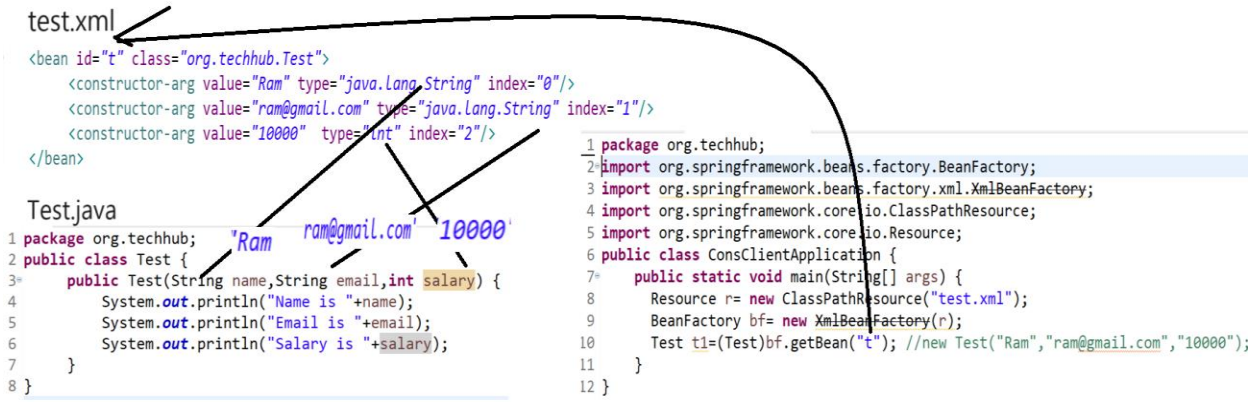
When we have parameterized constructor in class then we need to pass parameter to constructor by spring container then we need to use constructor injection so if we want to pass parameter to constructor using XML bean then we have tag name as <constructor-arg value=”value” type=”data type of parameter “index=” index number of parameter”/>

constructor-arg tag indicate we want to pass parameter to constructor using spring container

value: this parameter indicate value to constructor.

type: this parameter indicates data type of parameter.

index: index means sequence of parameter means 0^{th} index 1^{st} parameter, 1^{st} index second parameter and so on….



Source code.

Test.java

```
package org.techhub;
public class Test {
        public Test(String name,String email, int salary) {
                System.out.println("Name is "+name);
                System.out.println("Email is "+email);
                System.out.println("Salary is "+salary);
        }
```

```
}
```

test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                    "https://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
  <bean id="e" class="org.techhub.Employee">
      <property name="id" value="1"/>
      <property name="name" value="ram"/>
      <property name="sal" value="10000"/>
  </bean>
  <bean id="t" class="org.techhub.Test">
       <constructor-arg value="Ram" type="java.lang.String" index="0"/>
       <constructor-arg value="ram@gmail.com" type="java.lang.String" index="1"/>
       <constructor-arg value="10000"  type="int" index="2"/>
  </bean>
</beans>
```

ClientApplication

```
package org.techhub;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
public class ConsClientApplication {
       public static void main(String[] args) {
         Resource r= new ClassPathResource("test.xml");
         BeanFactory bf= new XmlBeanFactory(r);
         Test t1=(Test)bf.getBean("t"); //new Test("Ram","ram@gmail.com","10000");
       }
}
```

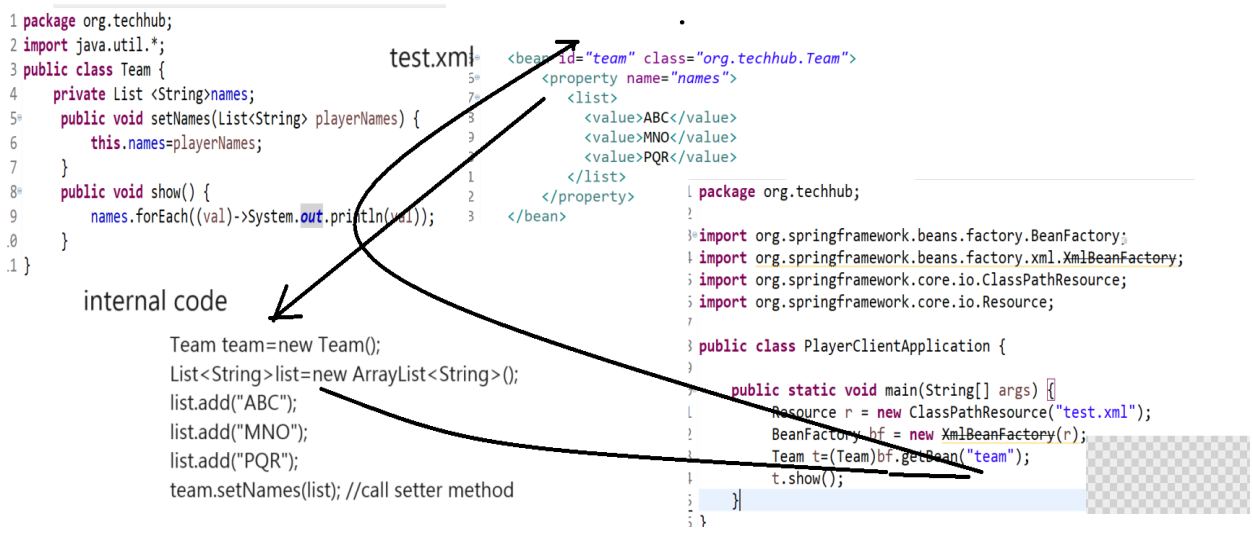
Collection dependency

Collection dependency means if we pass collection as parameter to the method or constructor by using spring framework called as collection dependency.
If we want to work with Collection dependency by using Spring Framework we have following tag.

```
<bean id=”idname”  class=”classname”>
    <property name=”fieldname”>
       <parameter collectionname>
         <value>value of parameter</value>
       </parameter collectionname>
  </property>
</bean>
```

Example: Suppose consider we have Team class and in team we have method name as

void setNames(List<String> playerNames): This method can work with collection dependency means we required to pass List Collection as parameter in method but if we pass this list collection with data by spring framework called as collection dependency shown in following diagram and code.



Source code

```
package org.techhub;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
public class PlayerClientApplication {
        public static void main(String[] args) {
                Resource r = new ClassPathResource("test.xml");
                BeanFactory bf = new XmlBeanFactory(r);
                Team t=(Team)bf.getBean("team");
                t.show();
```

```
	}
}
```

POJO class

```
package org.techhub;
import java.util.*;
public class Team {
   private List <String>names;
	public void setNames(List<String> playerNames) {
		this.names=playerNames;
	}
	public void show() {
		names.forEach((val)->System.out.println(val));
	}
}
```

Test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                    "https://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
<bean id="team" class="org.techhub.Team">
	<property name="names">
	  <list>
	    <value>ABC</value>
	    <value>MNO</value>
	    <value>PQR</value>
	  </list>
	</property>
  </bean>
</beans>
```

Map dependency

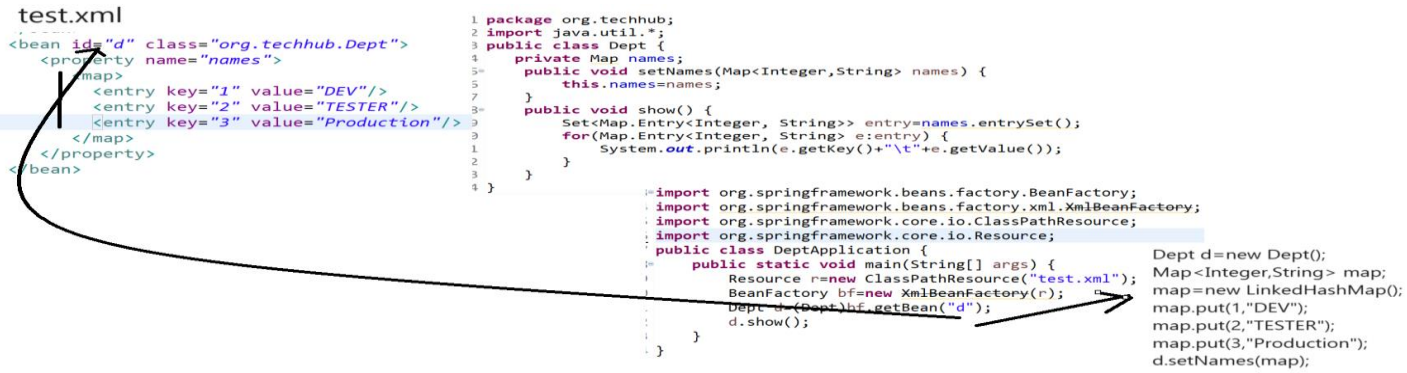
When we pass Map as parameter to method or constructor called as Map dependency. When we pass Map as parameter to method using spring framework using XML bean configuration we have to use following tags.

```
<bean id=”beanname” class=”classname”>
<property name=”propertyname”>
```

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

Example

Suppose consider we have class Name as Dept and we have method name as
void setName(Map<Integer, String> names): This method should accept employee data by using key and value pair.
void show(): This method can show the all employee details.



Object dependency

Object dependency means when we have method or constructor is dependent on particular class object called as object dependency.

How to manage the object dependency by Spring Framework

```
<bean id="ref"  class="classname">
   <property name="propertyname"  ref="parameter class object  id"/>
</bean>
```

```
class Employee
{
 private int id;
 private String name;
 private int sal;
//setter and getters
}
```

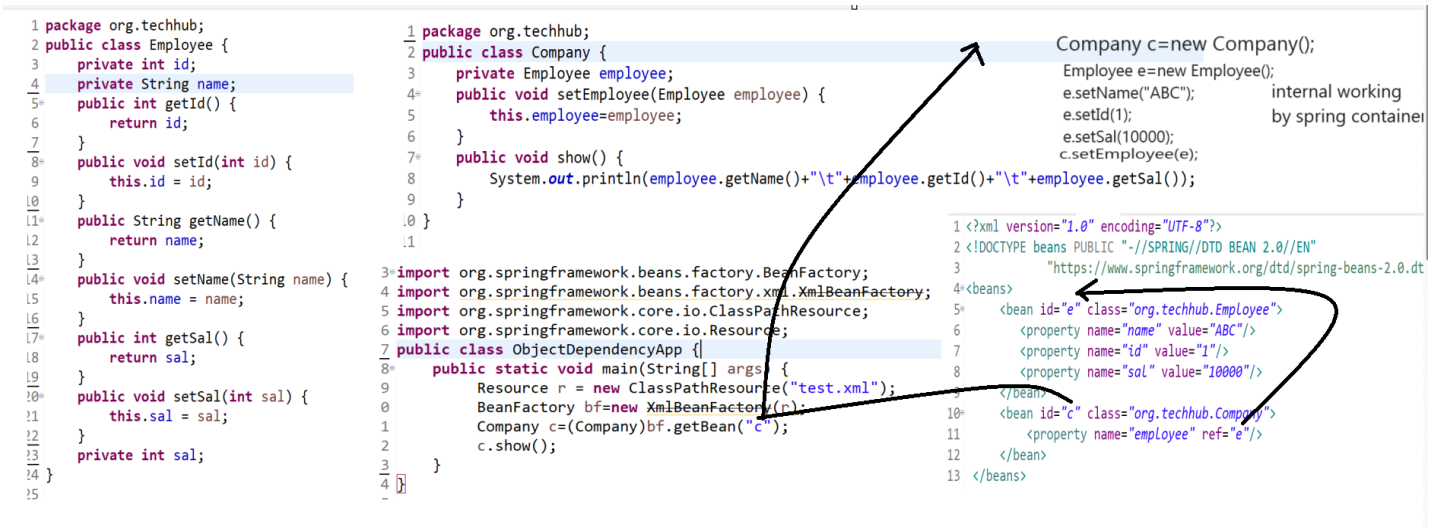
```
class Companny
{ Employee employee;

   void setEmployee(Employee employee)
   { this.employee=employee;
   }
}
```

```
<bean  id="e" class="org.techhub.Employee">
  <property name="name" value="Ram"/>
   <property name="id" value="1"/>
   <property name="sal" value="10000"/>
</bean>
<bean id="c"  class="org.techhub.Company">
   <property name="employee"  ref="e"/>
</bean>
```

Note: if we think about above code we have two classes name as Employee with three field id name and sal and we have one more class name as Company with method void setEmployee(Employee employee) here setEmployee() method required object of Employee class so this method is dependent on Employee class object so we can say it is object dependency.

If we work with object dependency if we call the bean id of dependent class then parameter class object gets created automatically.

Shown in Following Example and Diagram



Source code

Employee.java

```
package org.techhub;
public class Employee {
	private int id;
	private String name;
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public int getSal() {
		return sal;
	}
	public void setSal(int sal) {
		this.sal = sal;
	}
	private int sal;
```

```
}
```

Company.java

```
	package org.techhub;
public class Company {
	private Employee employee;
	public void setEmployee(Employee employee) {
		this.employee=employee;
	}
	public void show() {
	System.out.println(employee.getName()+"\t"+employee.getId()+"\t"+employee.
getSal());
	}
}
```

test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                    "https://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
    <bean id="e" class="org.techhub.Employee">
      <property name="name" value="ABC"/>
      <property name="id" value="1"/>
      <property name="sal" value="10000"/>
    </bean>
    <bean id="c" class="org.techhub.Company">
       <property name="employee" ref="e"/>
    </bean>
 </beans>
```

ObjectDependencyApp.java

```
package org.techhub;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
public class ObjectDependencyApp {
	public static void main(String[] args) {
		Resource r = new ClassPathResource("test.xml");
```

```
            BeanFactory bf=new XmlBeanFactory(r);
            Company c=(Company)bf.getBean("c");
            c.show();
      }
}
```

AUTOWIRNG

Q. What is Autowiring concept?

Auto wiring concept works with object dependency only means if we call the parameter object from dependent class object or parameter bean from dependent bean then we need to use <property name=" propertyname" ref=" parameter class object id"/> means need to call manually parameter object by using ref attribute but if we use auto wiring concept then we not need to call parameter bean using ref attribute it is internally call by spring container.

TYPES OF AUTO WIRING

byName

If we use the byName auto wiring concept then your parameter bean id configures in XML file and parameter reference name declare in dependent class object must be same.

Syntax: <bean id=" ref" class=" dependentclassname" auto-wired=" byName">
</bean>

Example

```java
package org.techhub;
public class Employee {
    private int id;
    private String name;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getSal() {
        return sal;
    }
    public void setSal(int sal) {
        this.sal = sal;
    }
    private int sal;
}
```

```java
package org.techhub;
public class Company {
    private Employee employee;
    public void setEmployee(Employee employee) {
        this.employee=employee;
    }
    public void show() {
        System.out.println(employee.getName()+"\t"+employee.getId()+"\t"+employee.getSal());
    }
}
```

must be same when we work with byName auto wiring concept

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
        "https://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
    <bean id="employee" class="org.techhub.Employee">
        <property name="name" value="ABC"/>
        <property name="id" value="1"/>
        <property name="sal" value="10000"/>
    </bean>
    <bean id="c" class="org.techhub.Company" autowire="byName">
    </bean>
</beans>
```

Source code

```
package org.techhub;
public class Employee {
        private int id;
        private String name;
        public int getId() {
                return id;
        }
        public void setId(int id) {
                this.id = id;
```

```
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public int getSal() {
		return sal;
	}
	public void setSal(int sal) {
		this.sal = sal;
	}
	private int sal;
}
```

Company.java

```
package org.techhub;
public class Company {
	private Employee employee;
	public void setEmployee(Employee employee) {
		this.employee=employee;
	}
	public void show() {
	System.out.println(employee.getName()+"\t"+employee.getId()+"\t"+employee.
getSal());
	}
}
```

test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                    "https://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
    <bean id="employee" class="org.techhub.Employee">
      <property name="name" value="ABC"/>
      <property name="id" value="1"/>
      <property name="sal" value="10000"/>
    </bean>
```

```
    <bean id="c" class="org.techhub.Company" autowire="byName">
    </bean>
 </beans>
```

ObjectDependencyApp.java

```
package org.techhub;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
public class ObjectDependencyApp {
	public static void main(String[] args) {
		Resource r = new ClassPathResource("test.xml");
		BeanFactory bf=new XmlBeanFactory(r);
		Company c=(Company)bf.getBean("c");
		c.show();
	}
}
```

byType

byType auto wiring internally check the data type of object configured in bean as well as declare in dependent class.

Employee.java

```
package org.techhub;
public class Employee {
	private int id;
	private String name;
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
```

```
	}
	public int getSal() {
		return sal;
	}
	public void setSal(int sal) {
		this.sal = sal;
	}
	private int sal;
}
```

Company.java

```
package org.techhub;
public class Company {
	private Employee employee;
	public void setEmployee(Employee employee) {
		this.employee=employee;
	}
	public void show() {
	System.out.println(employee.getName()+"\t"+employee.getId()+"\t"+employee.
getSal());
	}
}
```

test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                    "https://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
    <bean id="e" class="org.techhub.Employee">
      <property  name="name"  value="ABC" />
      <property name="id" value="1"  />
      <property name="sal" value="10000"/>
    </bean>
    <bean id="c" class="org.techhub.Company" autowire="byType">
    </bean>
 </beans>
```

ObjectDependencyApp.java

```
package org.techhub;
```

```
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
public class ObjectDependencyApp {
	public static void main(String[] args) {
		Resource r = new ClassPathResource("test.xml");
		BeanFactory bf=new XmlBeanFactory(r);
		Company c=(Company)bf.getBean("c");
		c.show();
	}
}
```

byConstructor

This auto wiring helps us to perform object dependency when we have object as parameter in constructor means object dependency with constructor.

Employee.java

```
package org.techhub;
public class Employee {
	private int id;
	private String name;
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public int getSal() {
		return sal;
	}
	public void setSal(int sal) {
		this.sal = sal;
```

```
	}
	private int sal;
}
```

Company.java

```
package org.techhub;
public class Company {
	private Employee employee;
	public void setEmployee(Employee employee) {
		this.employee=employee;
	}
	Public Company() {
	System.out.println(employee.getName()+"\t"+employee.getId()+"\t"+employee.
getSal());
	}
}
```

test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                    "https://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
    <bean id="e" class="org.techhub.Employee">
      <property  name="name"  value="ABC" />
      <property name="id" value="1"  />
      <property name="sal" value="10000"/>
    </bean>
    <bean id="c" class="org.techhub.Company" autowire="constructor">
    </bean>
 </beans>
```

ObjectDependencyApp.java

```
package org.techhub;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
public class ObjectDependencyApp {
	public static void main(String[] args) {
```

```
            Resource r = new ClassPathResource("test.xml");
            BeanFactory bf=new XmlBeanFactory(r);
            Company c=(Company)bf.getBean("c");
            c.show();
        }
}
```

autodetect

This auto wiring work with by default auto wiring technique of spring framework and by default spring framework use the byType auto wiring concept.

Employee.java

```
package org.techhub;
public class Employee {
        private int id;
        private String name;
        public int getId() {
                return id;
        }
        public void setId(int id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public int getSal() {
                return sal;
        }
        public void setSal(int sal) {
                this.sal = sal;
        }
        private int sal;
}
```

Company.java

```
package org.techhub;
```

```
public class Company {
	private Employee employee;
	public void setEmployee(Employee employee) {
		this.employee=employee;
	}
	public void show() {
	System.out.println(employee.getName()+"\t"+employee.getId()+"\t"+employee.
getSal());
	}
}
```

test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                    "https://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
    <bean id="e" class="org.techhub.Employee">
      <property  name="name"  value="ABC" />
      <property name="id" value="1"  />
      <property name="sal" value="10000"/>
    </bean>
    <bean id="c" class="org.techhub.Company" autowire="autodetect">
    </bean>
 </beans>
```

ObjectDependencyApp.java

```
package org.techhub;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
public class ObjectDependencyApp {
	public static void main(String[] args) {
		Resource r = new ClassPathResource("test.xml");
		BeanFactory bf=new XmlBeanFactory(r);
		Company c=(Company)bf.getBean("c");
		c.show();
	}
}
```

How to configure spring framework by using Annotation

Q. What is annotation?

Annotations in Java provide additional information to the compiler and JVM. An annotation is a tag representing additional information about classes, interfaces, variables, methods, or fields. Annotations do not impact the execution of the code that they annotate and it represent in java using @ Symbol.

Q. Why need to configure spring application by using annotation?

The major goal of annotation is avoiding the XML configuration of spring application so, if we want to work with spring core have some common annotation required for spring application.

1.@Configuration 2.@Bean 3.@Value 4.@Required 5.@Component 6.@Autowired 7.@Qualifier 8.@Lookup 9.@Scope 10.@PostConstructor 11.@PreDestroy 12.@Service 13.@Repository etc.

How to configure spring application by using annotation

1. Create POJO class

```
import org.springframework.beans.factory.annotation.Value;
public class Employee {
	@Value("1")
	private int id;
	@Value("ABC")
	private String name;
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
}
```

Note: if we think about above POJO class we provide value to property of POJO class by using @Value annotation.

2. Create Configuration

If we want to mark any class as spring bean configuration class we have annotation name as @Configuration and using this annotation we can mark class as configuration class.

```
package org.techhub;
import org.springframework.context.annotation.Configuration;
@Configuration
public class Config {

}
```

Note: if we think about Configuration class it acts as XML file.

3. Configure bean in Configuration class by using @Bean annotation

```
package org.techhub;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
@Configuration
public class Config {
	@Bean("emp")
	public Employee getEmployee() {
		return new Employee();
	}
}
```

@Bean annotation help us to create bean class object by spring container and perform dependency injection and inversion of control.

4. Create client application and call the Configuration and bean which we want to use

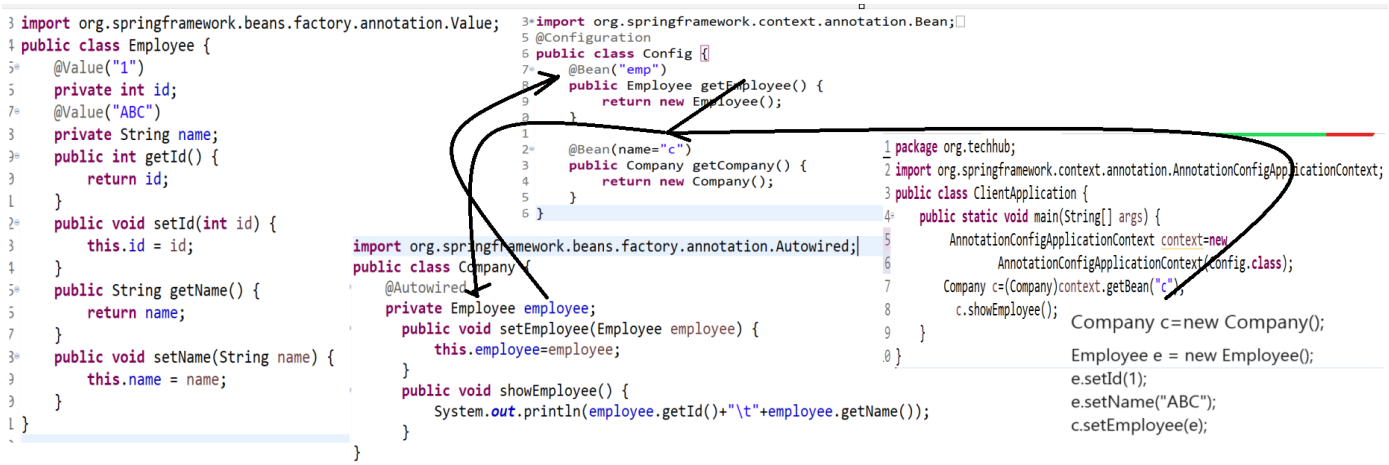
When we work with annotation then we need to use following spring container

```
package org.techhub;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
public class ClientApplication {
	public static void main(String[] args) {
		AnnotationConfigApplicationContext context=new
AnnotationConfigApplicationContext(Config.class);
```

```
        Employee e=(Employee)context.getBean("emp");
        System.out.println(e.getId()+"\t"+e.getName());
    }
}
```

Q. How to perform auto wiring concept by using Spring Framework with annotation?

If we want to perform auto wiring using annotation we have to use @Autowired Annotation.



Example with source code

Employee.java

```
package org.techhub;
import org.springframework.beans.factory.annotation.Value;
public class Employee {
        @Value("1")
        private int id;
        @Value("ABC")
        private String name;
        public int getId() {
                return id;
        }
        public void setId(int id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
```

```
	}
}
```

Company.java

```
package org.techhub;
import org.springframework.beans.factory.annotation.Autowired;
public class Company {
	@Autowired
   private Employee employee;
	 public void setEmployee(Employee employee) {
		this.employee=employee;
	 }
	 public void showEmployee() {
		System.out.println(employee.getId()+"\t"+employee.getName());
	 }
}
```

Config.class

```
package org.techhub;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
@Configuration
public class Config {
	@Bean("emp")
	public Employee getEmployee() {
		return new Employee();
	}

	@Bean(name="c")
	public Company getCompany() {
		return new Company();
	}
}
```

ClientApplication.java

```
package org.techhub;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
public class ClientApplication {
	public static void main(String[] args) {
```

```
            AnnotationConfigApplicationContext context=new
                        AnnotationConfigApplicationContext(Config.class);
            Company c=(Company)context.getBean("c");
             c.showEmployee();
      }
}
```

@Qualifier annotations

@Qualifier annotation help us to perform dynamic polymorphism using spring application means @Qualifier annotation work with @Autowired annotation means suppose we have interface and we implement it in multiple classes or bean classes and if we use @Autowired annotation with interface then we cannot create object of interface but we can create its reference and if we want to create reference of interface we need to create object of its implementer class and for which implementer object should be create with interface reference decide by @Qualifier annotation means we need to pass bean id in @Qualifier annotation whose implementer class object we want to create.

Vehicle.java

```
package org.techhub;
public interface Vehicle {
  void engine();
}
```

Bike.java

```
package org.techhub;
public class Bike implements Vehicle {
      @Override
      public void engine() {
            System.out.println("I am Bike class bean object");
      }
}
```

Car.java

```
package org.techhub;
public class Car implements Vehicle {
      @Override
      public void engine() {
            System.out.println("I am Car bean object");
      }
}
```

Shop.java

```
package org.techhub;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
public class Shop {
   @Autowired
   @Qualifier("c")   //Vehicle vehicle=new Car();
      Vehicle vehicle;
      public void setVehicle(Vehicle vehicle) {
            this.vehicle=vehicle;
      }
      public void show() {
            vehicle.engine();
      }
}
```

Config.java

```
package org.techhub;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
@Configuration
public class Config {
      @Bean(name="b")
      public Bike getBike() {
            return new Bike();
      }
      @Bean(name="c")
      public Car getCar() {
            return new Car();
      }
      @Bean(name="s")
      public Shop getShop() {
            return new Shop();
      }
}
```

DynamicClientApplication.java

```
package org.techhub;
```

```
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
public class DynamicClientApplication {
	public static void main(String[] args) {
		AnnotationConfigApplicationContext context=new
AnnotationConfigApplicationContext(Config.class);
		Shop s=(Shop)context.getBean("s");
		s.show();
	}
}
```

Spring Bean Scope

Bean scope decide how to create bean object

TYPES OF BEAN SCOPE

1. singleton

singleton scope means bean can create only one object in whole application.
Note: by default, spring bean use the singleton bean scope.

2. prototype

proto type bean means bean can create new object every time when we pass its id in getBean() method means proto type scope create normal bean object.

How to set the scope to the spring bean

if we want to set scope to the spring bean using xml we have scope attribute in bean shown in following or we can use @Scope annotation with bean

Example with singleton scope

```
package org.techhub.demo;
public class Test {
	public Test() {
		System.out.println("I am constructor");
	}
}
```

test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
```

```
                "https://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
 <bean id="t" class="org.techhub.demo.Test" scope="singleton">
 </bean>
</beans>
```

ClientApplication.java

```
package org.techhub.demo;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
public class ClientApplication {
	public static void main(String[] args) {
	   Resource r = new ClassPathResource("test.xml");
	   BeanFactory bf= new XmlBeanFactory(r);
	   Test t=(Test)bf.getBean("t");
	   Test t1=(Test)bf.getBean("t");
	   Test t2=(Test)bf.getBean("t");
	   Test t3=(Test)bf.getBean("t");
	}
}
```

Output

```
<terminated> ClientApplication (2) [Java Ap
I am constructor
```

Example with prototype

```
package org.techhub.demo;
public class Test {
	 public Test() {
		System.out.println("I am constructor");
	 }
}
```

test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                "https://www.springframework.org/dtd/spring-beans-2.0.dtd">
```

```
<beans>
 <bean id="t" class="org.techhub.demo.Test" scope="prototype">
 </bean>
</beans>
```

ClientApplication.java

```
package org.techhub.demo;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
public class ClientApplication {
	public static void main(String[] args) {
	   Resource r = new ClassPathResource("test.xml");
	   BeanFactory bf= new XmlBeanFactory(r);
	   Test t=(Test)bf.getBean("t");
	   Test t1=(Test)bf.getBean("t");
	   Test t2=(Test)bf.getBean("t");
	   Test t3=(Test)bf.getBean("t");
	}
}

//we will discuss this in web application using spring
3. session
4. application
5. request
6. page
Etc.
```

SPRING BEAN LIFE CYCLE

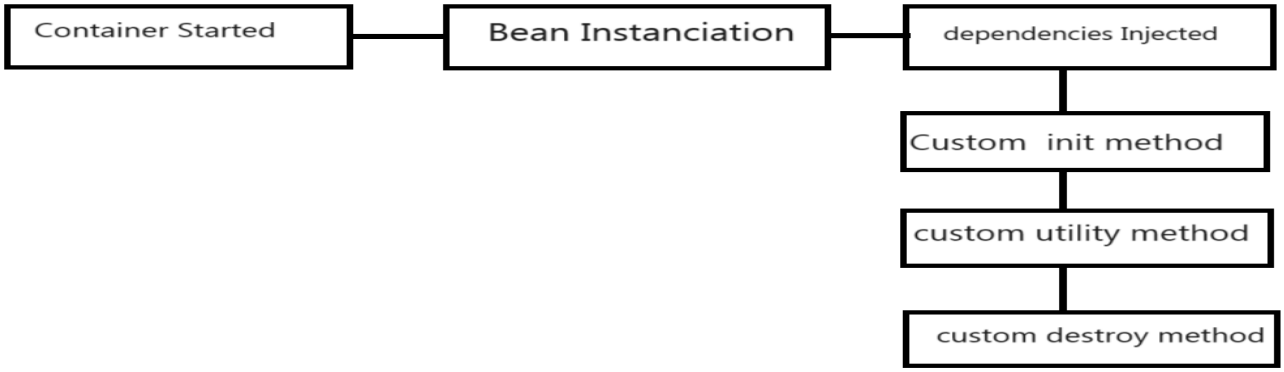
Spring bean life cycle

spring bean life decide how to spring bean object get created by spring container, perform dependency injection other activity of spring and how bean destroy by spring container.

THERE ARE THREE WAYS TO IMPLEMENT THE SPRING BEAN LIFE CYCLE

a) using XML configuration
b) using Annotation
c) using programming approach

If we want to implement spring bean life cycle practically we have to know its stages



Container Started: means when we create object BeanFactory or ApplicationContext called as container started.

Bean Instantiation and dependencies injected: this two stages get executed when user execute the getBean() method means when we call getBean() method of Spring container then we accept the bean id in getBean() then internally bean object get created by spring container as well as call internally setter methods.

Custom init: here we user can define own method or own custom method which is execute automatically after bean object creation and after dependency injection
Custom Utility method: this is user define method where user can write its own logics.

Custom destroy method: this method call automatically when we close the spring container by using close() method

Spring bean life cycle using XML Configuration

Steps

1) Create POJO class and define custom init and custom destroy method in it.

```
package org.techhub.beanslifeapp;
public class Employee {
	private int id;
	private String name;
	public Employee() {
		System.out.println("I am constructor");
	}
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public int getSal() {
		return sal;
	}
	public void setSal(int sal) {
		this.sal = sal;
	}
	public void init() {
		System.out.println("I am custom init method");
	}
	public void destroy() {
		System.out.println("I am custom destroy method");
	}
	private int sal;
}
```

2) Configure bean XML file and call custom init method and custom destroy method from bean tag

If we want to call custom init method we have attribute in bean tag name as

init method=”methodname” and if we want to call custom destroy method we have one attribute in bean tag name as destroy-method=”methodname”

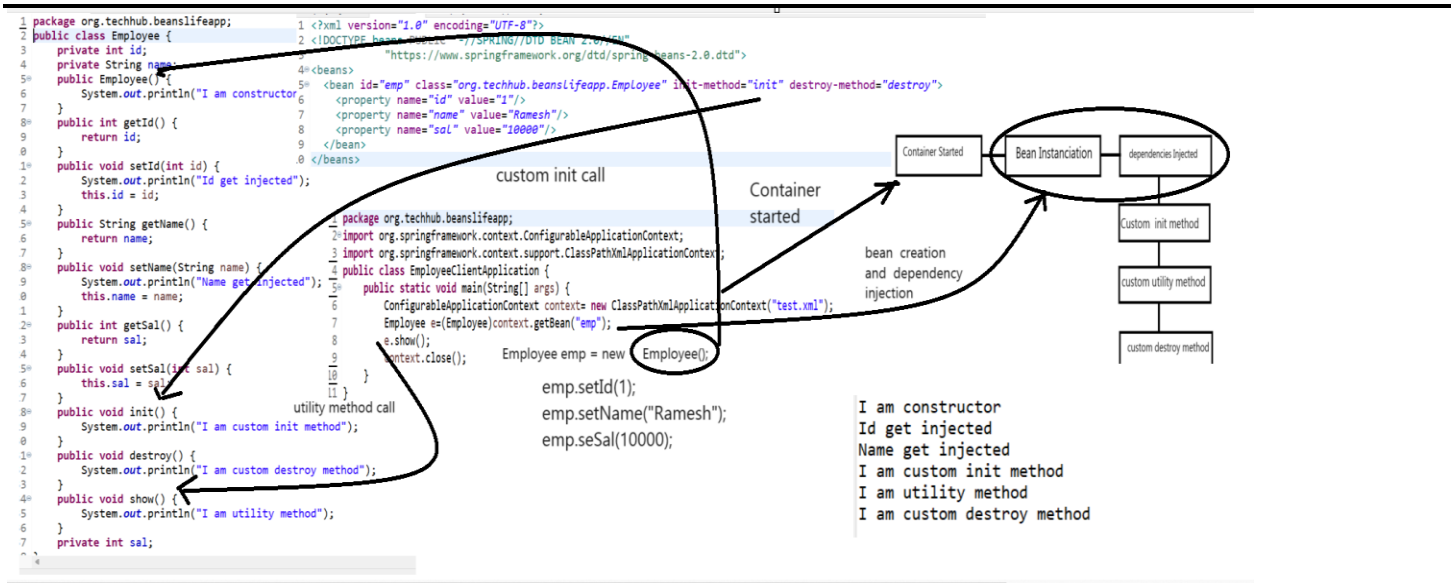
test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                        "https://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
  <bean id="emp" class="org.techhub.beanslifeapp.Employee" init-method="init"
destroy-method="destroy">
    <property name="id" value="1"/>
    <property name="name" value="Ramesh"/>
    <property name="sal" value="10000"/>
  </bean>
</beans>
```

3) Write Client application and create spring container and call XML file
EmployeeClientApplication.java

```
package org.techhub.beanslifeapp;
import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class EmployeeClientApplication {
	public static void main(String[] args) {
		ConfigurableApplicationContext context= new
ClassPathXmlApplicationContext("test.xml");
		Employee e=(Employee)context.getBean("emp");
		e.show();
		context.close();
	}
}
```

Following Diagram shows the working of bean life cycle



Bean Life cycle implementation by using programming approach

If we want to implement bean life cycle by using programming approach we have to implement two interfaces in your bean class name as InitializingBean and DisposableBean.

InitializingBean interface provide one method name as afterPropertiesSet() work as init() method so we need to override this method and write init method logics and DisposableBean interface provide one method to us name as destroy() which work as destroy method so we need to override it.

Source code

```
package org.techhub.beanslifeapp;
import org.springframework.beans.factory.DisposableBean;
import org.springframework.beans.factory.InitializingBean;
public class Employee implements InitializingBean, DisposableBean {
	private int id;
	private String name;
	public Employee() {
		System.out.println("I am constructor");
	}
	public int getId() {
		return id;
	}
	public void setId(int id) {
		System.out.println("Id get injected");
```

```
            this.id = id;
        }
        public String getName() {
            return name;
        }
        public void setName(String name) {
            System.out.println("Name get injected");
            this.name = name;
        }
        public int getSal() {
            return sal;
        }
        public void setSal(int sal) {
            this.sal = sal;
        }
        public void show() {
            System.out.println("I am utility method");
        }
        private int sal;
        @Override
        public void destroy() throws Exception {
            System.out.println("Work as destroy method");
        }
        @Override
        public void afterPropertiesSet() throws Exception {
             System.out.println("Work as init method or I am init method");
        }
}
```

test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                    "https://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
  <bean id="emp" class="org.techhub.beanslifeapp.Employee">
    <property name="id" value="1"/>
    <property name="name" value="Ramesh"/>
    <property name="sal" value="10000"/>
  </bean>
```

```
</beans>
```

EmployeeClientApplication.java

```
package org.techhub.beanslifeapp;
import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class EmployeeClientApplication {
	public static void main(String[] args) {
ConfigurableApplicationContext context= new
ClassPathXmlApplicationContext("test.xml");
		Employee e=(Employee)context.getBean("emp");
		e.show();
		context.close();
	}
}
```

Implement the Bean life cycle by using annotations

If we want to implement spring bean by using annotations we have to use two annotations name as

@PostConstruct and @PreDestroy

1. If we use @PostConstruct annotation with any method then method work as init() method
2. If we use the annotation with @PreDestroy then method work as destroy() method shown in following code.

Employee.java

```
package org.techhub.beanslifeapp;
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import org.springframework.beans.factory.DisposableBean;
import org.springframework.beans.factory.InitializingBean;
import org.springframework.beans.factory.annotation.Value;
public class Employee {
	@Value("1")
	private int id;
	@Value("ABC")
	private String name;
	public Employee() {
		System.out.println("I am constructor");
```

```
	}
	public int getId() {
		return id;
	}
	public void setId(int id) {
		System.out.println("Id get injected");
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		System.out.println("Name get injected");
		this.name = name;
	}
	public int getSal() {
		return sal;
	}
	public void setSal(int sal) {
		this.sal = sal;
	}
	@PostConstruct
	public void init() {
		 System.out.println("I am init method");
	 }
	@PreDestroy
	public void destroy() {
		System.out.println("I am destroy method");
	}
	public void show() {
		System.out.println("I am utility method");
	}
	@Value("10000")
	private int sal;

}
```

Config.java

```
package org.techhub.beanslifeapp;
```

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
@Configuration
public class Config {
	@Bean(name="emp")
	public Employee getEmployee() {
		return new Employee();
	}
}
```

EmployeeClientApplication.java

```
package org.techhub.beanslifeapp;
import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class EmployeeClientApplication {
	public static void main(String[] args) {
		AnnotationConfigApplicationContext context=new
AnnotationConfigApplicationContext(Config.class);
		Employee e=(Employee)context.getBean("emp");
		e.show();
		context.close();
	}
}
```

Q. What is spring JDBC?

Spring JDBC or Spring DAO it is used for connect your java application with database application by using spring framework and this is part of spring module built on basis of plain JDBC means internally spring JDBC using API of plain JDBC.

Q. Why use Spring JDBC?

1. Not need to handle the checked exceptions if we work with Plain JDBC we need to handle checked exception compulsory.
2. Spring JDBC Provide central DB configuration by using XML or by using Config class as single tone bean.
3. JdbcTemplate help us to provide all DDL and DML Operation.
4. Not need to close the connection it is internally manage by JdbcTemplate class.

Steps to work with Spring JDBC

1. Create Maven project

2. Add the Following maven dependencies

a) Spring core
b) Spring context
c) Spring JDBC
d) MYSQL Connector maven dependency

```
<dependencies>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-core</artifactId>
   <version>5.3.27</version>
</dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-context</artifactId>
   <version>5.3.27</version>
</dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-jdbc</artifactId>
   <version>5.3.27</version>
</dependency>
```

```
    <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
   <groupId>mysql</groupId>
   <artifactId>mysql-connector-java</artifactId>
   <version>8.0.28</version>
</dependency>
  </dependencies>
```

3. Create XML file and Config Database Credential or using annotation also.

If we want to configure database credential using spring JDBC we have inbuilt class name as.
DriverManagerDataSource from org.springframework.jdbc.core.datasource package so, we need to configure its bean and it four setter method for configure database credential

conn.xml

```
<?xml version="1.0" encoding="UTF-8"?>
       <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                     "https://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
  <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
     <property name="username" value="root"/>
     <property name="password" value="root"/>
     <property name="url" value="jdbc:mysql://localhost:3306/mysql"/>
     <property name="driverClassName" value="com.mysql.cj.jdbc.Driver"/>
  </bean>
</beans>
```

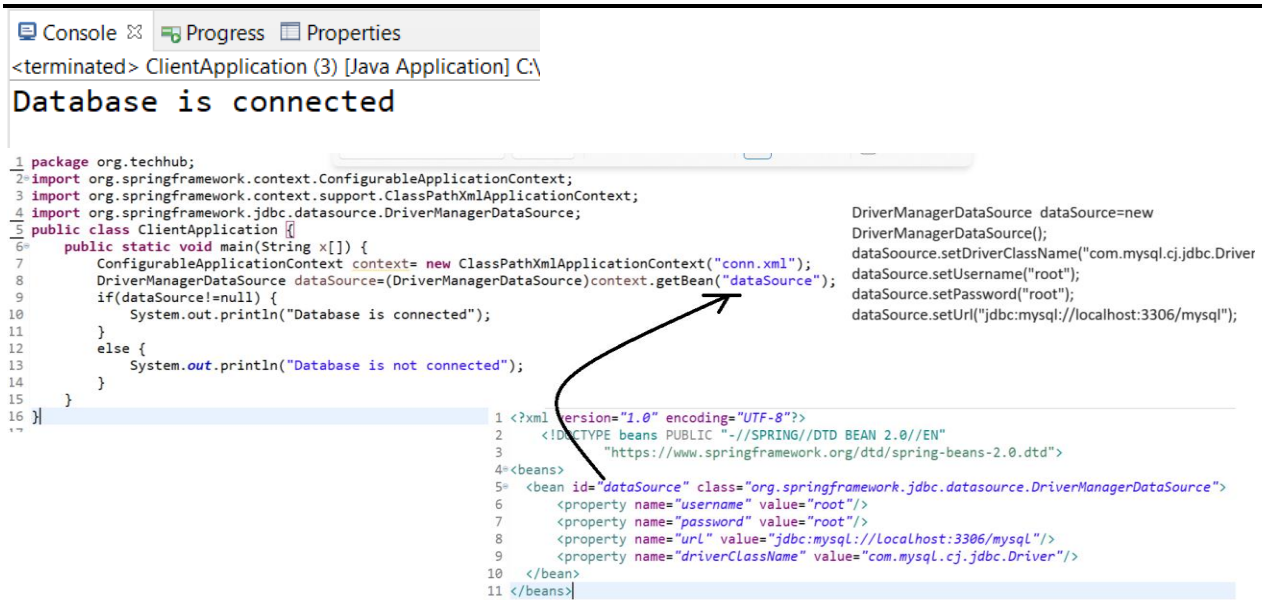
Once we configure Database using XML you can Create client application and check your database is connected or not.

Example

```
package org.techhub;
import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
public class ClientApplication {
       public static void main(String x[]) {
```

```
        ConfigurableApplicationContext context= new
ClassPathXmlApplicationContext("conn.xml");
        DriverManagerDataSource
dataSource=(DriverManagerDataSource)context.getBean("dataSource");
        if(dataSource!=null) {
            System.out.println("Database is connected");
        } else {
            System.out.println("Database is not connected");
        }
    }
}
```

Output



You can configure database credential by using annotations means you can create bean for DriverManagerDataSource class shown in following code.

Example database connection using annotation

```
package org.techhub.dbconfig.annot;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
@Configuration
public class DBConfig {
    @Bean(name="dataSource")
    public DriverManagerDataSource getDataSource() {
```

```
		DriverManagerDataSource dataSource= new DriverManagerDataSource();
		dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
		dataSource.setUsername("root");
		dataSource.setPassword("root");
		dataSource.setUrl("jdbc:mysql://localhost:3306/mysql");
		return dataSource;
	}
}
```

ClientApplication.java

```
package org.techhub.dbconfig.annot;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
public class DBConfigClientApplication {
	public static void main(String[] args) {
		AnnotationConfigApplicationContext context=new
AnnotationConfigApplicationContext(DBConfig.class);
		DriverManagerDataSource
dataSource=(DriverManagerDataSource)context.getBean("dataSource");
		if(dataSource!=null) {
			System.out.println("Database is connected");
		} else {
			System.out.println("Some problem is there.....");
		}
	}
}
```

4. Configure bean of JdbcTemplate

JdbcTemplate is used for perform all DDL and DML operation on database like as PreparedStatement and JdbcTemplate and DriverManagerDataSource work with object dependency means JdbcTemplate class is dependent on DriverManagerDataSource object.
Means we need to call DriverManagerDataSource bean id from JdbcTemplate.

JdbcTemplate configuration by using XML

```
package org.techhub;
import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
public class ClientApplication {
    public static void main(String x[]) {
        ConfigurableApplicationContext context= new ClassPathXmlApplicationContext("conn.xml");
        JdbcTemplate template=(JdbcTemplate)context.getBean("template");
        if(template!=null) {
            System.out.println("Databse is connected");
        }
        else {
            System.out.println("Database is not connected");
        }
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
    <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
            "https://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
  <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name="username" value="root"/>
      <property name="password" value="root"/>
      <property name="url" value="jdbc:mysql://localhost:3306/mysql"/>
      <property name="driverClassName" value="com.mysql.cj.jdbc.Driver"/>
  </bean>
  <bean id="template" class="org.springframework.jdbc.core.JdbcTemplate">
    <property name="dataSource" ref="dataSource"/>
  </bean>
</beans>
```

Source code of above diagram

```
<?xml version="1.0" encoding="UTF-8"?>
        <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                        "https://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
  <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name="username" value="root"/>
      <property name="password" value="root"/>
      <property name="url" value="jdbc:mysql://localhost:3306/mysql"/>
      <property name="driverClassName" value="com.mysql.cj.jdbc.Driver"/>
  </bean>
  <bean id="template" class="org.springframework.jdbc.core.JdbcTemplate">
    <property name="dataSource" ref="dataSource"/>
  </bean>
</beans>
```

ClientApplication.java

```
package org.techhub;
import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
public class ClientApplication {
        public static void main(String x[]) {
```

```
        ConfigurableApplicationContext context= new
ClassPathXmlApplicationContext("conn.xml");
        JdbcTemplate template=(JdbcTemplate)context.getBean("template");
        if(template!=null) {
            System.out.println("Database is connected");
        } else {
            System.out.println("Database is not connected");
        }
    }
}
```

Configuration of JdbcTemplate by using annotation

```
package org.techhub.dbconfig.annot;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
public class DBConfigClientApplication {
    public static void main(String[] args) {
        AnnotationConfigApplicationContext context=new AnnotationConfigApplicationContext(DBConfig.class);
        JdbcTemplate template=(JdbcTemplate)context.getBean("template");
        if(template!=null) {
            System.out.println("Database is connected");
        }
        else {
            System.out.println("Database is not connected");
        }
    }
}
```

```
package org.techhub.dbconfig.annot;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
@Configuration
public class DBConfig {
    @Bean(name="dataSource")
    public DriverManagerDataSource getDataSource() {
        DriverManagerDataSource dataSource= new DriverManagerDataSource();
        dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
        dataSource.setUsername("root");
        dataSource.setPassword("root");
        dataSource.setUrl("jdbc:mysql://localhost:3306/mysql");
        return dataSource;
    }
    @Bean(name="template")
    public JdbcTemplate getTemplate() {
        JdbcTemplate template= new JdbcTemplate();
        template.setDataSource(getDataSource());
        return template;
    }
}
```

Source code

DBConfig.java

```
package org.techhub.dbconfig.annot;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
@Configuration
public class DBConfig {
    @Bean(name="dataSource")
    public DriverManagerDataSource getDataSource() {
        DriverManagerDataSource dataSource= new DriverManagerDataSource();
        dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
        dataSource.setUsername("root");
        dataSource.setPassword("root");
```

```
		dataSource.setUrl("jdbc:mysql://localhost:3306/mysql");
		return dataSource;
	}
	@Bean(name="template")
	public JdbcTemplate getTemplate() {
		JdbcTemplate template= new JdbcTemplate();
		template.setDataSource(getDataSource());
		return template;
	}
}
```

DBConfigClientApplication.java

```
package org.techhub.dbconfig.annot;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
public class DBConfigClientApplication {
	public static void main(String[] args) {
		AnnotationConfigApplicationContext context=new
AnnotationConfigApplicationContext(DBConfig.class);
		JdbcTemplate template=(JdbcTemplate)context.getBean("template");
		if(template!=null) {
			System.out.println("Database is connected");
		}
		else {
			System.out.println("Database is not connected");
		}
	}
}
```

5. Work with Database

If we want to perform any database operation like as DDL or DML we have JdbcTemplate class and JdbcTemplate class provide some inbuilt method to us for work with database.

void execute(String sqlStatement): This method can perform all DDL and DML operation except select.

int update(String sqlStatement,PreparedStatementSetter): This method can execute the all DDL and DML operation except select and use for execute dynamic query and if operation get executed successfully return 1 otherwise return 0.

int update(String sqlStatement, Object []): This method can execute the all DDL and DML operation except select and use for execute dynamic query and if operation get executed successfully return 1 otherwise return 0.
List query(String selectStatement, RowMapper): This method can execute the select without parameter or where clause
List query(String selectStatement, PreparedStatementSetter, RowMapper): This method can execute the select statement with where clause or parameter.

Example: we want to create Spring JDBC program for insert record in table name as marchspringjdbc

```
package org.techhub.dbconfig.annot;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
public class DBConfigClientApplication {
	public static void main(String[] args) {
		AnnotationConfigApplicationContext context=new
AnnotationConfigApplicationContext(DBConfig.class);
		JdbcTemplate template=(JdbcTemplate)context.getBean("template");
		if(template!=null) {
			System.out.println("Database is connected");
			template.execute("insert into marchspringjdbc
values('ram','ram@gmail.com','1234567')");
			System.out.println("Record save Success.....");
		} else {
			System.out.println("Database is not connected");
		}
	}
}
```

Output

```
mysql> select *from marchspringjdbc;
+------+---------------+---------+
| name | email         | contact |
+------+---------------+---------+
| ram  | ram@gmail.com | 1234567 |
+------+---------------+---------+
1 row in set (0.03 sec)
```

Example: we want to accept name email and contact from keyboard and store in database table.

```
package org.techhub.dbconfig.annot;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import java.util.*;
public class DBConfigClientApplication {
	public static void main(String[] args) {
		AnnotationConfigApplicationContext context=new
AnnotationConfigApplicationContext(DBConfig.class);
		JdbcTemplate template=(JdbcTemplate)context.getBean("template");
		if(template!=null) {
			System.out.println("Database is connected");
			Scanner xyz =new Scanner(System.in);
			System.out.println("Enter name email and contact");
			String name=xyz.nextLine();
			String email=xyz.nextLine();
			String contact=xyz.nextLine();
			template.execute("insert into marchspringjdbc
values('"+name+"','"+email+"','"+contact+"')");
			System.out.println("Record save Success.....");
		} else {
			System.out.println("Database is not connected");
		}
	}
}
```

int update(String sqlStatement,PreparedStatementSetter): This method is used for pass sql statement as well as pass run time parameter to SQL Statement PreparedStatementSetter is a functional interface so you can use it by using lambda expression or by using anonymous inner class.

Example using Anonymous inner class

```
package org.techhub;
import org.springframework.context.ConfigurableApplicationContext;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.*;
```

```
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.PreparedStatementSetter;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
public class ClientApplication {
	public static void main(String x[]) {
		ConfigurableApplicationContext context= new
ClassPathXmlApplicationContext("conn.xml");
		JdbcTemplate template=(JdbcTemplate)context.getBean("template");
		if(template!=null) {
			System.out.println("Database is connected");
		System.out.println("Enter name email and contact");
		Scanner xyz  = new Scanner(System.in);
		final String name=xyz.nextLine();
		final String email=xyz.nextLine();
		final String contact=xyz.nextLine();
			PreparedStatementSetter pstmt=new PreparedStatementSetter() {
			public void setValues(PreparedStatement ps) throws SQLException {
					ps.setString(1, name);
					ps.setString(2,email);
					ps.setString(3, contact);
				}
			};
int value=template.update("insert into marchspringjdbc values(?,?,?)",pstmt);
			if(value>0) {
				System.out.println("Record Save Success.....");
			} else {
				System.out.println("Some problem is there.....");
			}
		} else {
			System.out.println("Database is not connected");
		}
	}
}
```

Example using Lambda expression with PreparedStatementSetter

```
package org.techhub;
import org.springframework.context.ConfigurableApplicationContext;
import java.sql.PreparedStatement;
```

```
import java.sql.SQLException;
import java.util.*;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.PreparedStatementSetter;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
public class ClientApplication {
	public static void main(String x[]) {
		ConfigurableApplicationContext context= new
ClassPathXmlApplicationContext("conn.xml");
		JdbcTemplate template=(JdbcTemplate)context.getBean("template");
		if(template!=null) {
			System.out.println("Database is connected");
		System.out.println("Enter name email and contact");
		Scanner xyz  = new Scanner(System.in);
		final String name=xyz.nextLine();
		final String email=xyz.nextLine();
		final String contact=xyz.nextLine();
			PreparedStatementSetter pstmt=(PreparedStatement ps) ->{
					ps.setString(1, name);
					ps.setString(2,email);
					ps.setString(3, contact);
				};
int value=template.update("insert into marchspringjdbc values(?,?,?)",pstmt);
			if(value>0) {
				System.out.println("Record Save Success.....");
			} else {
				System.out.println("Some problem is there.....");
			}
		} else {
			System.out.println("Database is not connected");
		}
	}
}
```

Example using lambda expression

```
package org.techhub;
import org.springframework.context.ConfigurableApplicationContext;
import java.sql.PreparedStatement;
```

```
import java.sql.SQLException;
import java.util.*;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.PreparedStatementSetter;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
public class ClientApplication {
	public static void main(String x[]) {
		ConfigurableApplicationContext context= new
ClassPathXmlApplicationContext("conn.xml");
		JdbcTemplate template=(JdbcTemplate)context.getBean("template");
		if(template!=null) {
			System.out.println("Database is connected");
		System.out.println("Enter name email and contact");
		Scanner xyz  = new Scanner(System.in);
		final String name=xyz.nextLine();
		final String email=xyz.nextLine();
		final String contact=xyz.nextLine();
	int value=template.update("insert into marchspringjdbc
values(?,?,?)",(PreparedStatement ps) ->{
				ps.setString(1, name);
				ps.setString(2,email);
				ps.setString(3, contact);
			});
			if(value>0) {
				System.out.println("Record Save Success.....");
			} else {
				System.out.println("Some problem is there.....");
			}
		} else {
			System.out.println("Database is not connected");
		}
	}
}
```

Example Input email and delete from database table using Spring JDBC

```
package org.techhub;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
@Configuration
public class Config {
	@Bean(name="dataSource")
	public DriverManagerDataSource getDataSource() {
		DriverManagerDataSource dataSource=new DriverManagerDataSource();
		dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
		dataSource.setUsername("root");
		dataSource.setPassword("root");
		dataSource.setUrl("jdbc:mysql://localhost:3306/mysql");
		return dataSource;
	}
	@Bean(name="template")
	public JdbcTemplate getTemplate() {
		JdbcTemplate template=new JdbcTemplate(getDataSource());
		return template;
	}
}
```

ClientApplication.java

```
package org.techhub;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.PreparedStatementSetter;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.*;
public class ClientApplication {
	public static void main(String[] args) {
		AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(Config.class);
		JdbcTemplate template = (JdbcTemplate) context.getBean("template");
		Scanner xyz = new Scanner(System.in);
		System.out.println("Enter email");
		final String email = xyz.nextLine();
int value = template.update("delete from marchspringjdbc where email=?", new
PreparedStatementSetter() {
			public void setValues(PreparedStatement ps) throws SQLException {
```

```
                        ps.setString(1, email);
                    }
                });
                if (value > 0) {
                    System.out.println("Success.....");
                } else {
                    System.out.println("failed.....");
                }
        }
}
```

How to select record from database table using spring JDBC

If we want to select record from database table using spring JDBC we have query() method.

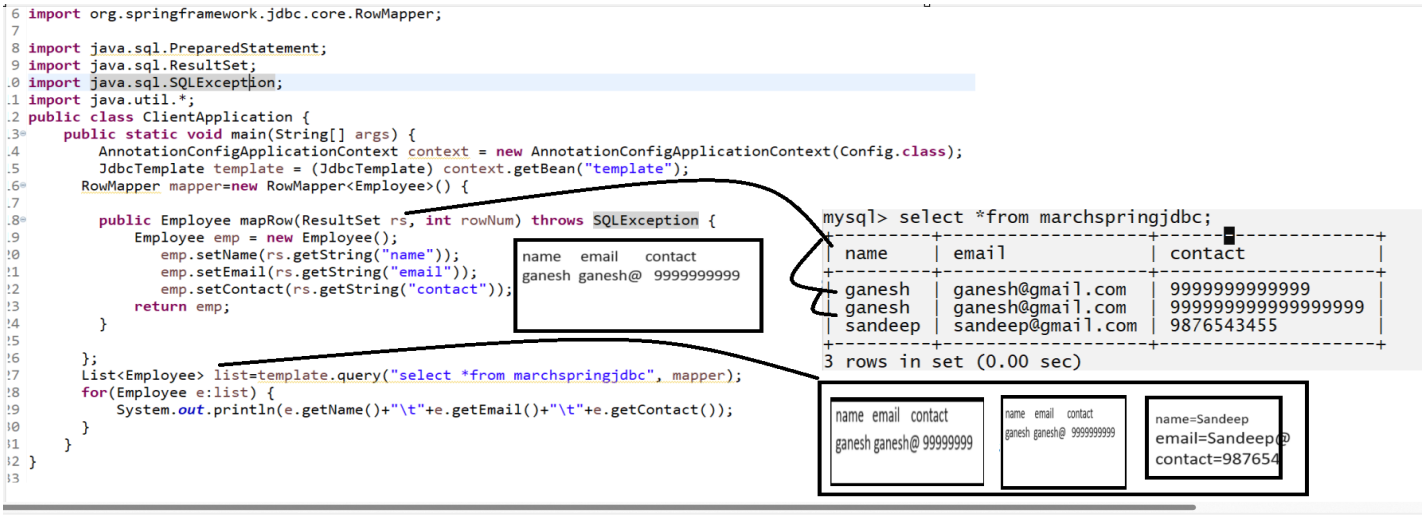
Steps

1. Create POJO class as per the table structure
2. Call the query() method like as

List query(String selectStatement, RowMapper):

String selectStatement: this parameter indicates we can write here select statement.

RowMapper: this is functional interface which is used for fetch data from database table. It contain mapRow(ResultSet rs) this method can fetch single row at time from database table.



Source code

package org.techhub;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

```
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.PreparedStatementSetter;
import org.springframework.jdbc.core.RowMapper;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.*;
public class ClientApplication {
	public static void main(String[] args) {
		AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(Config.class);
		JdbcTemplate template = (JdbcTemplate) context.getBean("template");
	  RowMapper mapper=new RowMapper<Employee>() {
	     public Employee mapRow(ResultSet rs, int rowNum) throws SQLException {
			Employee emp = new Employee();
			  emp.setName(rs.getString("name"));
			  emp.setEmail(rs.getString("email"));
			  emp.setContact(rs.getString("contact"));
			return emp;
		}
	  };
	  List<Employee> list=template.query("select *from marchspringjdbc", mapper);
	  for(Employee e:list) {
		System.out.println(e.getName()+"\t"+e.getEmail()+"\t"+e.getContact());
	  }
	}
}
```

How to fetch particular column from table using Spring JDBC

If we want to fetch particular column from table using Spring JDBC then we create array of Object class and store all columns data in it and store Object array in List Collection.

Example: we want to fetch only name and email from table

```
package org.techhub;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.PreparedStatementSetter;
import org.springframework.jdbc.core.RowMapper;
import java.sql.PreparedStatement;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.*;
public class ClientApplication {
	public static void main(String[] args) {
		AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(Config.class);
		JdbcTemplate template = (JdbcTemplate) context.getBean("template");
	  List<Object[]> list=template.query("select name, email from marchspringjdbc",
new RowMapper<Object[]>() {
		public Object[] mapRow(ResultSet rs, int rowNum) throws SQLException {
			Object obj[]=new Object[] {
				rs.getString("name"),rs.getString("email")};
				return obj;
			}
		});
		for(Object obj[]:list) {
			System.out.println(obj[0]+"\t"+obj[1]);
		}
	}
}
```

How to use where clause in select query using Spring JDBC

If we want to pass parameter to select query using where clause we have following syntax of query method**.**

Syntax: List query(String selectStatement, RowMapper, Object[])

String selectStatement: this parameter indicate we have select query here
RowMapper: using RowMapper you can fetch data from database table.
Object[]: this is used for pass run time parameter to where clause.
or
List query(String selectStatement PreparedStatementSetter, RowMapper):
String selectStatement: this parameter indicate we have select query here
PreparedStatementSetter: this is used for pass run time parameter to where clause.
RowMapper: using RowMapper you can fetch data from database table.

Example: we want to input email and contact of employee and get its detail.

```
package org.techhub;
import java.util.*;
```

```
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.PreparedStatementSetter;
import org.springframework.jdbc.core.RowMapper;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.*;
public class ClientApplication {
	public static void main(String[] args) {
		AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(Config.class);
		JdbcTemplate template = (JdbcTemplate) context.getBean("template");
		Scanner xyz  = new Scanner(System.in);
		System.out.println("Enter email and contact");
		String email=xyz.nextLine();
		String contact=xyz.nextLine();
	    List<Employee> list=template.query("select *from marchspringjdbc where
email=? and contact=?",new RowMapper<Employee>() {
			public Employee mapRow(ResultSet rs, int rowNum) throws
SQLException {
				Employee emp1 = new Employee();
				emp1.setName(rs.getString("name"));
				emp1.setEmail(rs.getString("email"));
				emp1.setContact(rs.getString("contact"));
				return emp1;
			}
	    }, new Object[] {email, contact});
	    for(Employee e:list) {
		System.out.println(e.getName()+"\t"+e.getEmail()+"\t"+e.getContact());
	    }
	}
}
```

Or

```
package org.techhub;
import java.util.*;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
```

```
import org.springframework.jdbc.core.PreparedStatementSetter;
import org.springframework.jdbc.core.RowMapper;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.*;
public class ClientApplication {
	public static void main(String[] args) {
		AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(Config.class);
		JdbcTemplate template = (JdbcTemplate) context.getBean("template");
		Scanner xyz  = new Scanner(System.in);
		System.out.println("Enter email and contact");
		final String email=xyz.nextLine();
		final String contact=xyz.nextLine();
	   List<Employee> list=template.query("select *from marchspringjdbc where
email=? and contact=?",new PreparedStatementSetter() {
			public void setValues(PreparedStatement ps) throws SQLException {
				// TODO Auto-generated method stub
				ps.setString(1,email);
				ps.setString(2, contact);
			}
		}, new RowMapper<Employee>() {
			public Employee mapRow(ResultSet rs, int rowNum) throws
SQLException {
				 Employee emp1 = new Employee();
				 emp1.setName(rs.getString("name"));
				 emp1.setEmail(rs.getString("email"));
				 emp1.setContact(rs.getString("contact"));
				return emp1;
			}
		});
		for(Employee e:list) {
		   System.out.println(e.getName()+"\t"+e.getEmail()+"\t"+e.getContact());
		}
	}
}
```

Spring MVC?

Q. What is spring MVC?

Spring MVC is module of spring framework which is used for develop the web application by using spring framework by using model view and controller concept or using MVC.

Q. What is MVC?

MVC stands for model view controller.

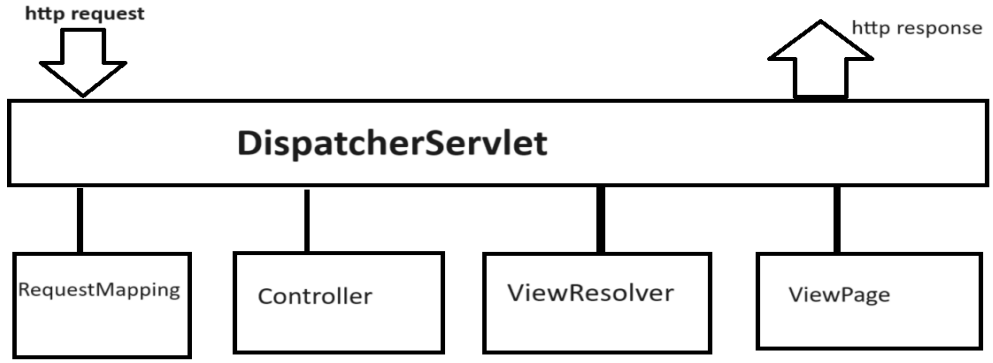
View: View means presentation layer from user provide input and get the results called as view.
Normally we can design view using html / CSS or JS.
Model: Model class which is used for accepts the data send by view and pass to controller.
Means model help us to store data of view and pass to different layer in application means to controller from controller to service and from service to repository layer.
Controller: Controller is class mark with @Controller annotation in spring framework which is used for accept data from view and pass to service and get results from service layer and send to the view.
Means using controller we can accept request data send by view and send response to view pages accept by server.

Q. Why use MVC or what is benefit of MVC?

If we use the traditional Servlet and JSP then we need to write UI code and business logics at same place.
but it is very complicated to maintain code when your application is very large means it is not maintainable so if we use MVC then the benefit is we separate the design logics and business logics in different layers the so the benefit is when we want to modify the code then it is very easier to maintain code.

How Spring MVC works?

If we want to identify how spring MVC work we need to know the Architecture of Spring MVC.



If we think about above architecture we have first http request

Http Request: when run spring MVC application then by default home page get executed or when click on some URL or when submit form to server it is an example of Http Request.

DispatcherServlet: DispatcherServlet is front controller means this is inbuilt class from org.springframework.web.servlet package and this configured in web.xml file and this responsible manage the all request send by view to controller and responses send by controller to view.

```
<servlet>
   <servlet-name>spring</servlet-name>
   <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>spring</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

Controller: Controller is class which is annotated by @Controller annotations in spring MVC.

```
@Controller
public class TestCtrl {

}
```

RequestMapping: RequestMapping is function which work as Request URL at server side means using RequestMapping we set URL where user can send request and get response means we need to function with @RequestMapping annotation under controller.

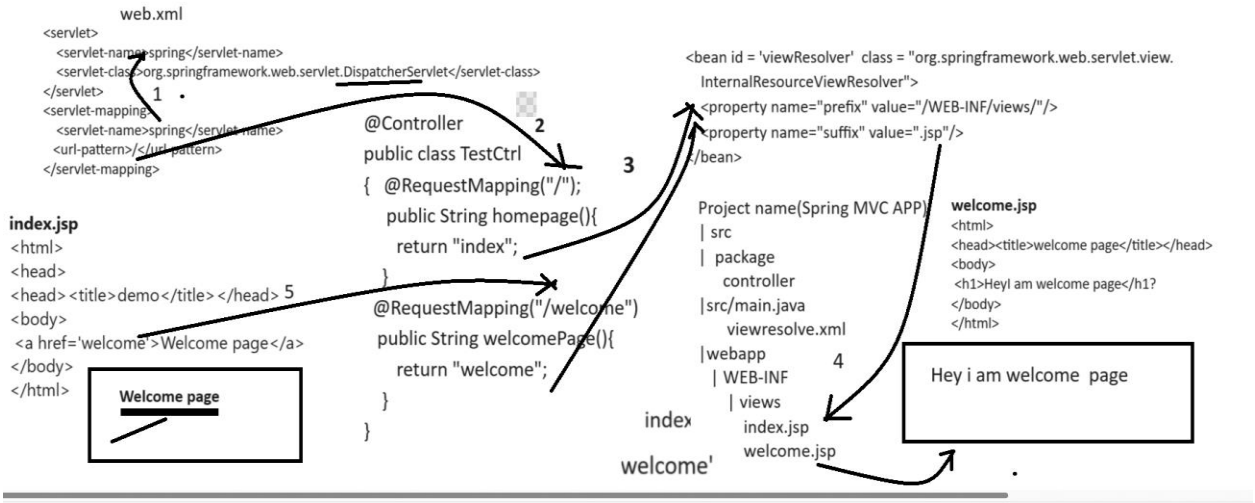
```
@Controller
public class TestCtrl {
```

```
    @RequestMapping("/");
     public String homepage(){
        return "index";
    }
  @RequestMapping("/welcome")
   public String welcomePage(){
        return "welcome";
    }
}
```

ViewResolver: ViewResolver is XML file or it may configuration class and ViewResolver indicate the view page extension and its path where is present in project means using ViewResolver DispatcherServlet can identify the location view for accept request and send response view RequestMapping.

```
<bean id = 'viewResolver'  class =
"org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/views/"/>
    <property name="suffix" value=".jsp"/>
</bean>
```

View: View is JSP page from user can provide input and get results working of spring MVC Architecture shown in following diagram.



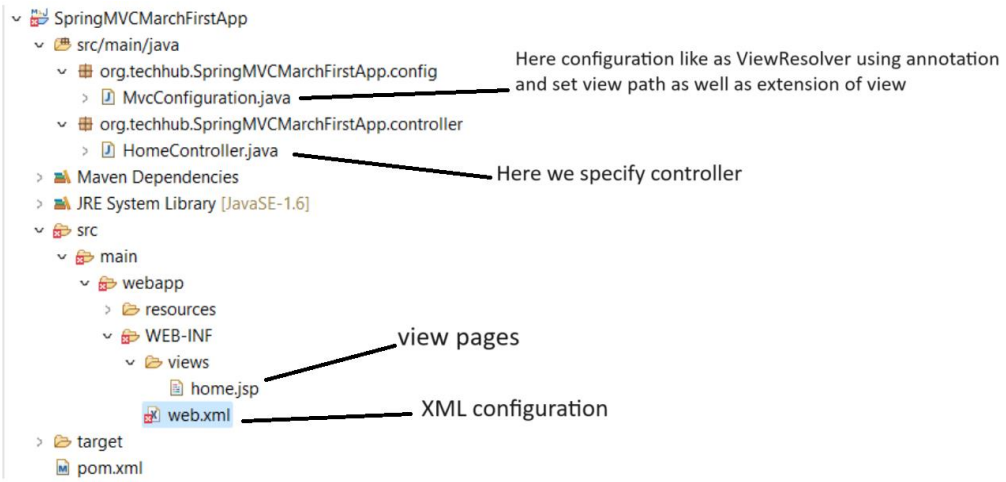
How to create Spring MVC application by using eclipse

Steps to create Spring MVC Application by using eclipse

1) Open eclipse and create maven project

File -- new project --- (select use default workspace) --- we list of inbuilt repository --- select repository name co.ntier -- click on next --- give group id and artifact id and click on next --- next and finish.

Once we create project using inbuilt repository we have inbuilt project structure shown in following diagram



Inbuilt Code Structure provided by co.ntier repository

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.ViewResolver;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
@Configuration
@ComponentScan(basePackages="org.techhub.SpringMVCMarchFirstApp")
@EnableWebMvc
public class MvcConfiguration extends WebMvcConfigurerAdapter {
	@Bean
	public ViewResolver getViewResolver(){
		InternalResourceViewResolver resolver = new
InternalResourceViewResolver();
		resolver.setPrefix("/WEB-INF/views/");
		resolver.setSuffix(".jsp");
		return resolver;
	}
```

```
	@Override
	public void addResourceHandlers(ResourceHandlerRegistry registry) {
registry.addResourceHandler("/resources/**").addResourceLocations("/resources/");
	}
}
```

Code description

```
public ViewResolver getViewResolver(){
		InternalResourceViewResolver resolver = new
InternalResourceViewResolver();
		resolver.setPrefix("/WEB-INF/views/");
		resolver.setSuffix(".jsp");
		return resolver;
	}
```

If we think about above code we configure ViewResolver using annotation here we create object of InternalViewResolver class and we set its prefix and suffix manually and return resolver object.

HomeController.java

```
package org.techhub.SpringMVCMarchFirstApp.controller;
import java.io.IOException;
import javax.servlet.http.HttpServletResponse;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;
@Controller
public class HomeController {
	@RequestMapping(value="/")
	public String test(HttpServletResponse response) throws IOException{
		return "home"; //home.jsp
	}
}
```

home.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
   <head>
```

```
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Home</title>
    </head>
    <body>
        <h1>Hello World!</h1>
        <p>This is the homepage!</p>
    </body>
</html>
```

Example: we want to call page using hyperlink in spring MVC

Source code of above diagram

home.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
 <meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
        <a href='wel' >Call welcome page</a>
</body>
</html>
```

HomeController.java

```
package org.techhub.SpringMVCMarchFirstApp.controller;
```

```
import java.io.IOException;
import javax.servlet.http.HttpServletResponse;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;
@Controller
public class HomeController {
	@RequestMapping(value="/")
	public String test(HttpServletResponse response) throws IOException{
		return "home"; //home.jsp
	}
	@RequestMapping("/wel")
	public String callWelcomePage() {
		return "welcome";
	}
}
```

welcome.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
 <h1>I am welcome page</h1>
</body>
</html>
```

How to submit form using spring MVC

If we want to submit form using spring MVC we have following steps.

1) Design form using view pages like as home.jsp

Note: we want to design registration page on home.jsp using name, email and contact

Home.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Home</title>
      <style>
        input{
          width:400px;
          height:40px;
        }
      </style>
   </head>
   <body>
       <input type='text' name='name' value=''/><br/><br/>
       <input type='text' name='email' value=''/><br/><br/>
       <input type='text' name='contact' value=''/><br/><br/>
       <input type='submit' name='s' value='Register'/>
   </body>
</html>
```

2) Create URL in controller where we want to submit form

Here we required to create URL function with HttpServletRequest parameter for accept form data and pass Map as parameter in function for forward the requested on another view page

```
package org.techhub.SpringMVCMarchFirstApp.controller;
import java.io.IOException;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;
@Controller
public class HomeController {
	@RequestMapping(value="/")
	public String test(HttpServletResponse response) throws IOException{
		return "home"; //home.jsp
```

```
	}
	@RequestMapping("/save")
	public String callWelcomePage(HttpServletRequest request,Map<String, String>
map) {
			String name=request.getParameter("name");
			String email=request.getParameter("email");
			String contact=request.getParameter("contact");
			map.put("n", name);
			map.put("e", email);
			map.put("c", contact);
		return "welcome";
	}
}
```

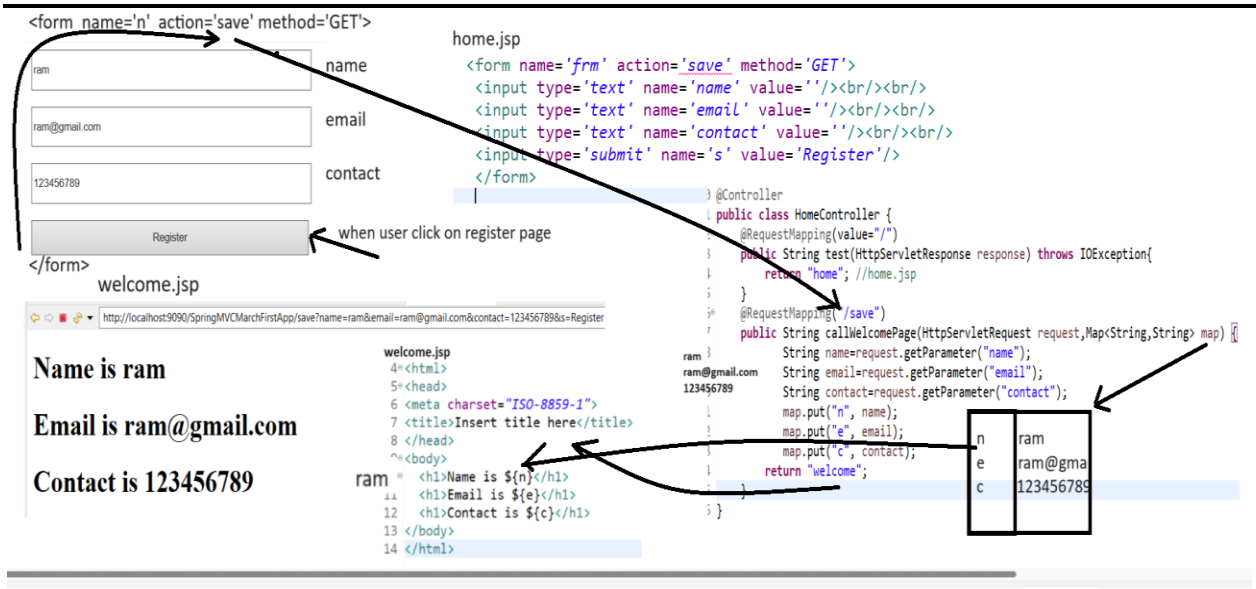
3) Submit form to destination URL where we accept its data as request parameter

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Home</title>
      <style>
        input{
          width:400px;
          height:40px;
        }
      </style>
   </head>
   <body>
      <form name='frm' action='save' method='GET'>
       <input type='text' name='name' value=''/><br/><br/>
       <input type='text' name='email' value=''/><br/><br/>
       <input type='text' name='contact' value=''/><br/><br/>
       <input type='submit' name='s' value='Register'/>
       </form>
   </body>
</html>
```

4) Accept the data on destination view page send by controller via Map using expression language.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
  <h1>Name is ${n}</h1>
  <h1>Email is ${e}</h1>
  <h1>Contact is ${c}</h1>
</body>
</html>
```

Working of all above code shown below.



If we think about above code we pass HttpServletRequest as parameter in /save request mapping and accept single field of form at time as requested data using request.getParameter() method but the limitation is if we have form with 50 fields then we need to write request.getParameter() 50 times manually for accept each and every control data of form so it is very complicated task in real time scenario so spring MVC provide facility to us called as model object means you can create Model class with setter and getter method same name as form field name and pass as parameter as

replacement of HttpServletRequest so Spring container accept all form data internally and store in model object.