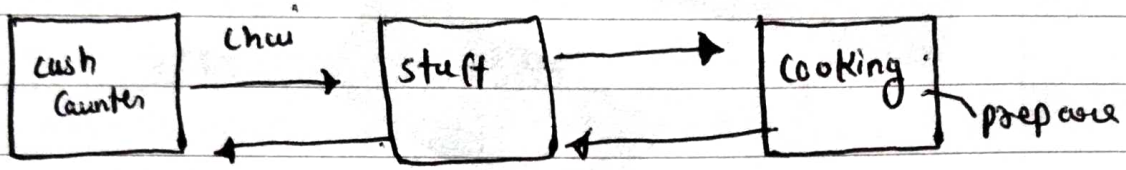
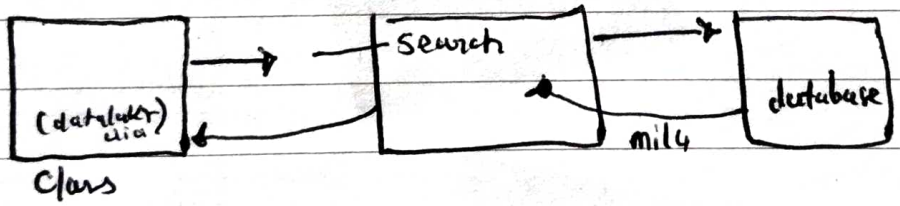
WEEK 53
001-364

Common problem ko solve kone ke liye ek predefined pattern bnaya use Design pattern bnaya.

Rest → ek architecture hai.



request aai counter pr use staff ko farward ki and staff ne farward ko cooking ko & usne prepare ki & staff ko farward ki. & staff ne cash counter ise hi API kehte hai.



Service → ek class ke andr multiple operation ko perform krna.

Business logic → logic lagana → ise ham service method me likhte hai.

Spring is a Dependency Injection framework to make java application loosely coupled.

It provide us IOC :- Inversion of Control / Container

Spring Framework makes the easy development of JavaEE.

It was Developed by Rod Johnson in 2003

Dependency Injection

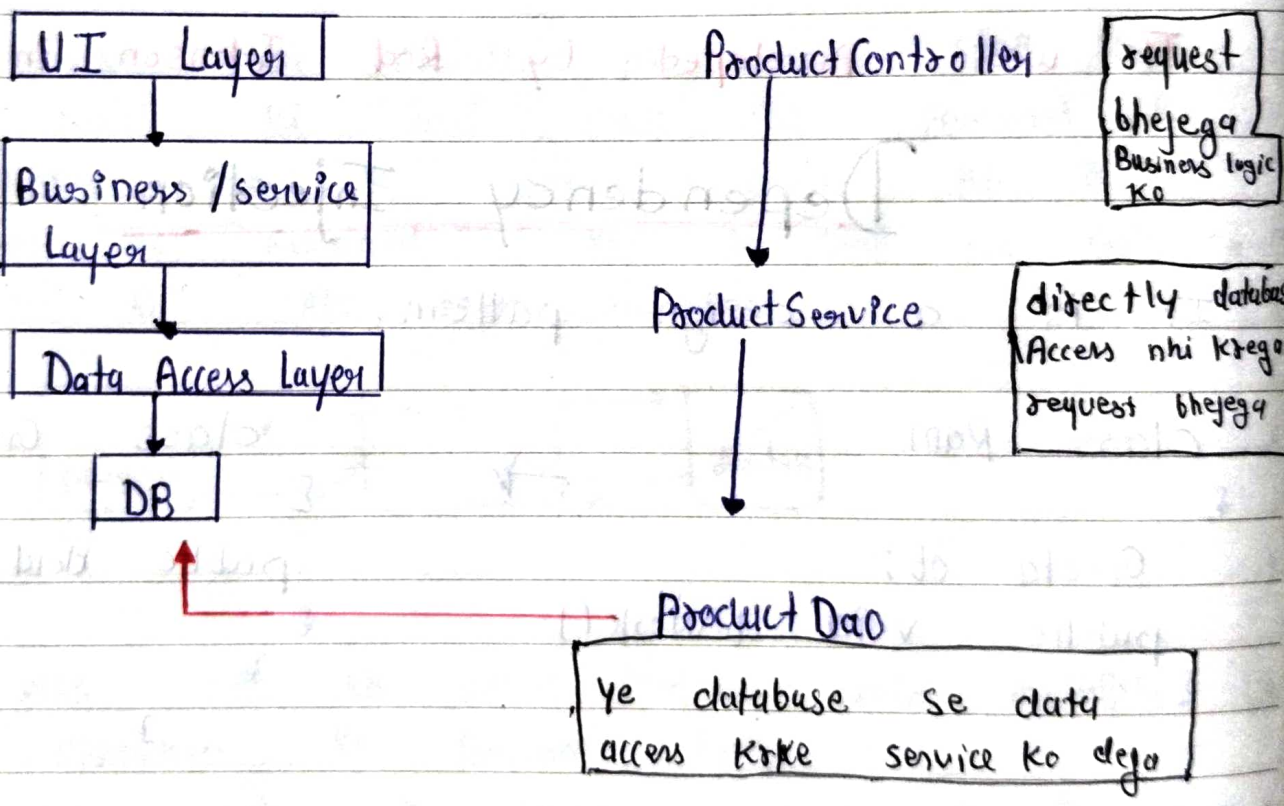
It is a design pattern.

```
class Ram                          class Geeta
{                                  {
   Geeta ob;                       public void doWork()
   public void doWork()            {
{ }                                }
}
```

yha pr Ram dependent hai Geeta pr work krne ke liye. Ek class dusre class pr depend kr rha hai work krne ke liye.
Spring khud dusri class ka object bnakr dusri class me inject kr deta hai

Object creation ka control spring ko dena hai spring sari dependency ka dynamically runtime pr object bna kr inject kr de use hi Inversion of Control kehte hai.

Spring and JEE

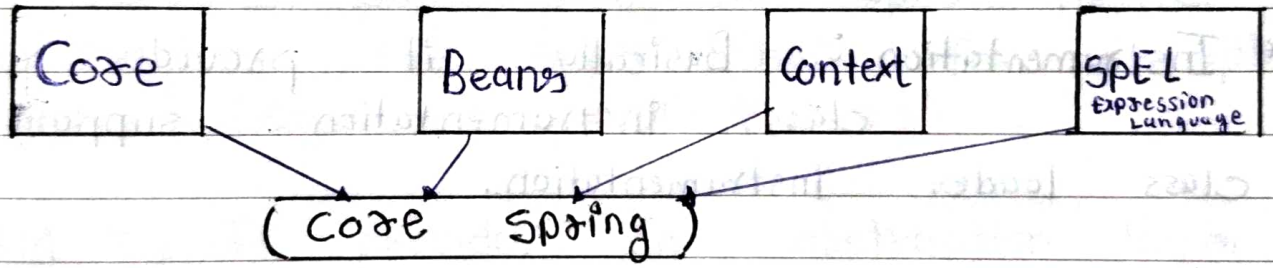


Agar ham productService me ProductDao ka directly new keyward se object bna dete hai to ye highly coupled ho jayega and hum future me productDao ko change krte hai to hume productservice me bhi change krna hoga.

Some Spring Modules / API

1) Spring JDBC → For data access layer
2) Spring ORM → For Hibernate & other ORM
3) Spring MVC → For UI Layer

Spring Core Modules :-



Ye 4 modules milkr spring ke sare fundamentals provide krte hai jese Dependency injection, IOC etc.

(*) Core and Beans fundamental part of an framework like IOC, DI.

(*) Context → Inherit features from Bean module like resource loading, event propogation, JEE

(*) SpEL → Query and manuplate object graph at runtime

5 THURSDAY
2023 JANUARY

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

WEEK 01
005-360

AOP | Aspect | Instrumentation | Messaging

* AOP :- Aspect Oriented Programming. This allow us to define intersepter and point cut to cleanly decouple the code.

And Baki teeno modules hame help krte hai to decouple code so that implement functionality should be seprated. Jisse hamari sari functionality alag - alag rahegi.

* Instrumentation :- Basically it provides us class, instrumentation support, or class loader instrumentation.

* Messaging :- This is to be we serve for as a foundation for messaging based application.

Data Access / Integration

JDBC | ORM

JMS | OXM

* **JDBC** :- Spring JDBC provides us JDBC abstraction layers that removes the need of feeding JDBC code and passing database vendor specific error code. Java DataBase Connectively

Ye hame ek abstract layer provide krta hai jisse hame purane JDBC code ka use nahi krna padta hai. Ye simple hota hai.

* **ORM** :- Spring ORM modules provides Integration layer. Iski help se ham kisi dusre ORM tool ko apne project me integrate kr sakte hai. for example Hibernate. (Object Relational Mapping)

* **OXM** :- It provides an abstraction layer that support object xml mapping.

* **JMS** :- Java Messaging Services. It helps us to produce and cosume the messages.

Ye sabhi module ki help se ham DataBase Manage kar sakte hai ya Ye sabhi module Data Integration layer ke liye.

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

7 SATURDAY
2023 JANUARY

WEEK 01
007-358

Spring Based on two most designing Patterns :-

1). Dependency Injection.
2). Inversion of Control (IOC).

Specification

It is a prototype of a product or guidline for creating a project.

Implementation

The product created using the given guidline are implemtation of specification

8 SUNDAY

Standardization

If we follow the specification / implementation model for creating product we will achieve standardization.

FEBRUARY 2023

M	T	W	T	F	S	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28					

MONDAY
JANUARY 2023

9

WEEK 02
009-356

have you ever observed :-

* How your mobile can accept sim from any company.

* How ATM machine accept any card of Bank.

In programming world, generally we use interface for creating the specification of a product (class).

* The classes will implement that interface to became an implementation of given specification.

Specification	Implementation

interface Car
{
void startPosition();
void Engine();
}
	class Tata implements Car
	{
	// we need to override method of car interface.
	}

10 TUESDAY
2023 JANUARY

WEEK 02
010-355

Spring Container

*) It is a component of Spring framework.

* Responsibilities of Spring container is to manage bean object.

* To manage life cycle of bean.

* Dependency Injection.

* AOP (Aspect Orient programming).

* Transaction Managment.

Dependency Injection

Dependency :- One class is dependent on another class object to perform some task.

Dependency Injection :- Instead of taking a fixed object, inserting the object from external word.

```
class Operation
{
    calc obj;
public void doAdd (int a, int b)
{
    obj.add (a,b);
}
public void doMod (int a, int b)
{
    obj.mod (a,b);
}
}
```

```
class calc
{
  void add (int a, int b)
  { }

  void mod (int a, int b)
  { }
}
```

here Operation class is depend on calc class object to perform the task.

```
Operation o = new Operation ();
o.setcalc (new Calc());
```

Injecting the calc object inside operation object (operation object is dependent on calc object).

This concept is called Dependency Injection.

12 THURSDAY
2023 JANUARY

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

WEEK 02
012-353

We can achieve Dependency Injection by two ways..

(1) Using setter method
(2) Using constructor

DI is used to achieve loose coupling

Inversion of Control

It is a core principle of Spring Framework that helps to manage the dependencies between object, promoting loose coupling and easier manageability.

Instead of objects creating their dependencies the control of creating and injecting dependencies is given to an external entity, typically the spring container.

---

* How object automatically created by Spring?
* We are providing the access of our objects to spring Framework.
*

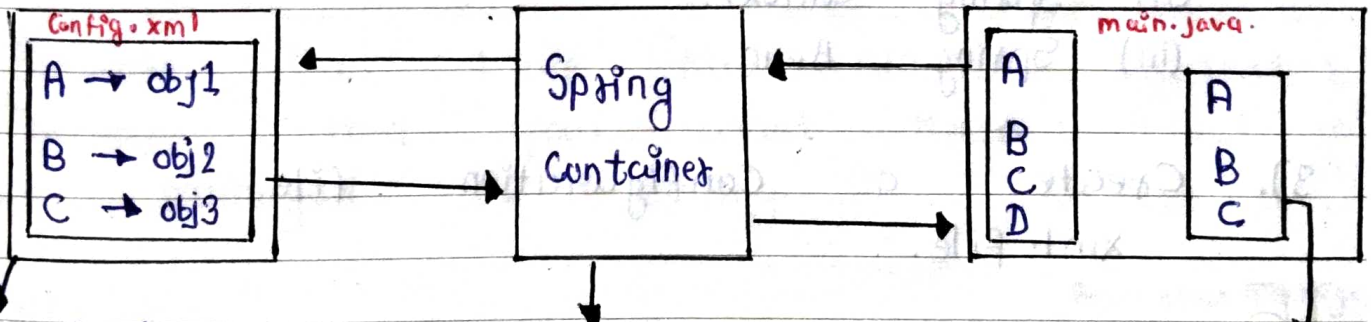
Now spring will create object for use, spring will inject one object into another object. Complete DI will be automated using IOC.

```
public class ServerDemo extends HttpServlet
{
    doGet() / doPost()
}
```

We never create object this ServerDemo class ourself. Servlet object is created and managed by Container only, which is called Servlet Container.

web.xml → It is a meta data file which contains the information about servlet meta data is configuration data.

DI Diagram



In this file we define the list of Bean object which spring needs to manage for you

Spring Container is responsible for creating object & providing you the object on demand

Whenever we need object of A,B,C class we just ask spring container to give us obj1, obj2, obj3.

14 SATURDAY
2023 JANUARY

WEEK 02
014-351

Containers are of two types

1). Bean Factory.

2). Application Context.

Steps to create simple Spring Application

1). Create a maven project.

2). Add following dependencies

(i) Spring Core.
(ii) Spring Context.
(iii) Spring Bean.

3). Create a Configuration file.
xml. file.

15 SUNDAY

4). Create a Bean class with data member and getter and setter

5). Configuration the bean in Configuration file.
that xml file

FEBRUARY 2023

M	T	W	T	F	S	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28					

MONDAY
JANUARY 2023 16

WEEK 03
016-349

Java Bean Class

A serializable class with no argument Constructor and with some getter and setter and data member, often used to encapsulation data is called Java Bean class

* In xml based configuration, file you have a pair of <bean> </bean> you need to configure all your bean object in this pair.

* <beans> all the object you want to spring to manage for you, you need to enter them here </beans>

Suppose you have two classes student & Employee and you want there object to be managed by spring.

```
<beans>                                   Bean class Package.
  <bean id = "S1" class = " com.beans.student" />
  <bean id = "E1" class = " com.beans.Employee" />
</beans>
```

Id's can be anything you want & this id only you will ask spring for an object.

```
Container.getBean("S1", student.class);
```

here we are demanding an object with id S1

FEBRUARY
MARCH
APRIL

17 TUESDAY
2023 JANUARY

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

WEEK 03
017-348

Automatic Object Creation

```
package com.beans;
Class Student
{
    int id;
    String name;

    — getter setter & Constructor —
}
```

Info.xml

```
<xml>
 <beans>
  <bean id = "S1" class = " com.beans.Student" />
</beans>
        </xml>
```

StudentInfo.Java.

```
import com.beans.*;
class StudentInfo
{
  public static void main (String ar[])
  {
ApplicationContext co = classPathXmlApplication Context ("Info.xml");
Student S1 = co.getBean ("S1", student.class);
        S1.setName ("Ram"); S1.setId(10);
        Sop(S1);
}
```

To ask spring for object following steps have to done:-

Step 1 → Create spring container object

```
ApplicationContext co = new ClassPathXMLApplicationContext("inform");
```

Step 2 → call the "getBean( )" method for demanding object.

```
Student S1 = co.getBean("S1", student.class);
S1.setId(10);
S1.setName(20);
System.out.println(S1);
```

Injecting Bean Properties

Setter Based DI

```
<beans>
    <bean id="S1" class="com.beans.Student">
  <property name="id" value="10" />
  <property name="name" value="ram" />
  </bean>
</beans>
```

StudentInfo.java

```
Student S1 = co.getBean("S1", student.class);
System.out.println(S1);
```

id = 10
name = ram

No need to set/init properties

19 THURSDAY
2023 JANUARY

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

WEEK 03
019-346

Injecting Collection type Property

Bean class

```
class Student
{
    int id;
    String name;
    List <String> emails;

    —getter / setters —
}
```

Type 1 → Manually

```
Student s1 = context.getBean("s1", Student.class);
List <String> emails = new ArrayList <> ();
 emails.add("ram@gmail.com");
  emails.add("sita4@gmail.com");

s1.setEmails(emails);

System.out.println(s1);
}
```

id = 10
name = ram
ram@gmail.com
sita4@gmail.com

Type 2 → xml based.

```
<bean id = "SL" class = "com.beans.Student" >
<property name = "id" value = "10" />
<property name = "name" value = "ram" />

<property name = "emails" >
  <list>
      <value> abc @gmail.com </value>
      <value> ram @gmail.com </value>
  </list>
      </property>
  </bean>
```

Now ↘ we do not need to set gmails (list) in this type, Spring will automatically inject the values in list.

Note :- Property tag uses setters to initialize members also know as setter based Dependency Injection.

21 SATURDAY
2023 JANUARY

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

WEEK 03
021-344

Set → Collection

It does not contain any duplicate value in it and if we try to insert any duplicate value it didn't add. (No error will be generate if we try).

POJO Class

```
private Set <String> emails ;
```

Info.xml

```
<property name = "emails" >
  <set>
    <value> abc @gmail.com </value>
    <value> bhanu @gmail.com </value>
  </set>
</property>
```

22 SUNDAY

Map - Collection

```
<property name = "mobileNo" >
  <map>
    <enter key = "Jio" value = "8827272702" />
    <enter key = "Airtel" value = "9770745999" />
  </map>
</property>
```

private Map<String, String> mobile

Constructor Based Dependency Injection

info.xml

```
<xml>
 <beans>
    <bean id = "s1" class = " com.beans.Student" >
  <constructor-arg  value  =  "20" />
  <constructor-arg  value  =  "Bhanu" />
  </bean>
          </beans>
                  </xml>
```

Student (Bean class)

```
public  class  Student
{
   private int id;
   private string name;

 public  Student (int id, string name)
 {
     this.id =  id;
     this.name  =  name;
  }
}

 public  string toString()
 {
   return " id = " +id + " name = " +name ;
}}
```

24 TUESDAY
2023 JANUARY

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

WEEK 04
024-341

Bean - Scope

Suppose you have entry for "S1" bean in info.xml. You request spring for an object.

--- spring will return you object

Suppose you can request for same object -now what spring will do?

will it return same object
or it will create new object and return it.

In Java code you are asking for same object multiple times.

```
Student s1 = context.getBean("S1", Student.class);
Student s2 = context.getBean("S1", Student.class);
```

what container will do?
will it return same object
or it will return/create new object.

This things depends on the scope of Bean Configured.

If you configure scope = "singleton", it will create just one instance & each time return same object.

If you configure

scope = "prototyp" >

It will create new object for each demand / request.

Code

```
<xml>
 <beans>

 <bean id = "s" class = " Com.beans.Student"
   scope = " prototype" />

   </beans>
</xml>
```

Scope = "singleton" :- Same object every time

Scope = "prototype" :- new object every time

By default Scope is Singleton

If we don't clearify scope bydefault scope will be singleton.

26 THURSDAY
2023 JANUARY

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

WEEK 04
026-339

@Component Annotation

In spring framework @component is used to declare a class as a spring Bean. which manages component in spring application context.

It helps spring to automatically detect and manage these Bean during application startup making them available for dependency injection and other spring feature.

mark @component on Bean class.

Steps to use @Component

Step 1 → mark bean class as @component

```
@component
public class Student
{
    private int id;
    private string name;
}
```

– getter setter Constructor. –

Step 1 → Take new schema of xml file from google with context and paste it into info.xml file.

Step 3 → Inform spring Framework to look for beans into particular package.

```
<context : component-scan base package = "com.beans" />
```

Step 4 → Ask for an object to spring.

```
ApplicationContext co = new ClassPathXMLApplicationContext("info.xml");
Student s = co.getBean("student", student.class);
```

↑ in small because id will be same name as bean class name.

Student (Bean) Class

```
@component
public class student
{
    private int id;
    private string name;

    — getter setter Constructor —

}
```

Info.xml

```
<xml>
        -new  schema-
<context : component-scan base package = " com.beans " />
</xml>
```

StudentInfo.java

```
public class StudentInfo
{
  public static void main (String args[])
  {
ApplicationContext co = ClassPathXMLApplicationContext("Info.xml");
Student s =  co.getBean ("student", student.class);

        S.setId (10);
        S.setName ("Bhanu");
```



```
        System.out.println (S);
  }
}
```

id = 10
name = Bhanu

It marks a class as a spring component. Spring component are java object that are managed by spring container and can be used to perform various tasks within the application.

When a class is marked with @component annotation, spring creates an instance of that class and register it as an spring Bean in the application context.

Such spring Bean are managed by spring container which provides feature such as dependency injection, life cycle, managment and IOC.

Note :- @component likhe se hame xml file me bean configuration nahi karna padta hai.

31 TUESDAY
2023 JANUARY

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

WEEK 05
031-334

Autowiring in Spring

DI is a process in which one bean is injected in another bean object, it can be done by 2 ways

(1) Setter Based DI
(2) Constructor Based DI

Till now we performing DI manually but Spring Framework have feature of automatic DI.

Autowiring :- DI is achieved explicitly by user through some code.

Autowiring is a spring feature through which DI is automated, we can achieve DI automatically.

Auto :- Automatically manages the bean or connection between beans.

Wiring :- Linked those object together. (objects created by spring).

WEDNESDAY
2023 FEBRUARY

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

WEEK 05
032-333

Autowiring Types

Autowiring can be achieved by two way

(1) Annotation Based (@Autowired)
(2) xml Based (we have a attribute "autowire")

@Autowired

It is a annotation in Spring which is used for automatic dependency injection.

It tells Spring : just create the object of bean and inject it as right place. Spring is handling all your connection for you.

Internally Working

Student -

```
<bean id = "student" class = " com. beans. Student" >
<property name =" address " ref = " address " />
</bean>
```

Address -

```
<bean id = "address" class =" com. beans. Address"/>
</bean>
```

Code for Autowiring

Student class

```
@Component
public class Student
{
    private int id;
    private String name;

    @Autowired
    private Address address;

    → Constructor & getter setters ←
}
```

Address Class

```
@Component
public class Address
{
    private int houseNo;
    private String city;
    private String state;

    → Constructor and Getter setters ←
}
```

3 FRIDAY
2023 FEBRUARY

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

WEEK 05
034-331

StudentEntry Class

```
public class StudentEntry
{
public static void main (String arr [])
{
ApplicationContext context = new ClassPathXml
                             ApplicationContext ("info.xml");

Student S1 = (Student) context.getBean ("student");
      S1.setId (101);
      S1.setName ("ram");

Address ad = S1.getAddress ();
          ad.setCity ("Indore");
          ad.HauseNo (10);
          ad.setState ("MP");

    S1.setAddress (ad);
   System.out.println (S1);
}
}
```

XML File

```
<xml>  <beans>
    <context : component-scan base package =
                              " com.beans " />
    </beans>
```

Xml Based Autowiring

Autowire in xml is based on modes of autowire attribute it has different modes.

1) **byName** :- In this mode spring is checking for the name of property mentioned in bean class Id of bean must be equal to name of that property. (isme bean ki id and bean class me instance variable ka name same hona chaiye.)

Student class

```
public class Student
{
    Address ad1;
}
```

Xml File

```
<bean id = "S1" class = "com.beans.Student" autowire = "byName" />

<bean id = "ad1" class = "com.beans.Address" />
```

isme ham chahe to property to initialize bhi kar sakte hai & Student class ke variable name & us bean ki id same honi chaiye.

6 MONDAY
2023 FEBRUARY

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

WEEK 06
037-328

2.) **by Type :-** In byType mode it will look for the type of property and search for bean of same type.

Isme sirf vo variable ki configuration ko dhundega class ko check krke. like Student class me Address type ka variable hai to vo sirf Address class ko dhundega.

Student Class

```
public class Student
{
    Address add;
    →getter setter, Cons ←
}
```

Xml file

```
<bean id = "s1" class = "com.beans.student" autowire = "byType" />
<bean id = "ad12" class = "com.beans.Address" />
```

Address

byType me vo variable ke type ki bean search krega. Id ko nhi dekhega. byName me variable name and bean id same dekhega.

If there are multiple bean of same class then if we want to provide different id for different object?

In this case (only in byTye) ambigiaus error will occur. Kyuki vo confuse ho jayega ki konsi bean inject kru. But there is an solution for this.

Xml File

```
<bean id = "st" class = " com.beans. student " autowire ="byTye"/>

<bean id = " add1" class = " com.beans. Address " />
```

Multiple bean for address

```
<bean id = "ad2" class = "com.beans. Address "
                    autowire candidate = "false" />
```

Now ye vali bean id autowire me use nhi hogi jisse ambiguity nhi aayegi.

And ye wali bean id ko koi dusri student class ki id use kr skti hai,

```
<property name =" address " ref =" ad2 " />
```

kr ke.

(sirf ye autowiring me use nhi hogi)

8 WEDNESDAY
2023 FEBRUARY

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

WEEK 06
039-326

3) Constructor :- In this the object was injected through the constructor, (Parameterized).

Student class

```
public class Student
{
    int id;
    String name;
    Address address;

    public Student (int id, string name, Address address)
    {
        this.id = id;
        this.name = name;
        this.address = address
    }
}
```

Address Class

```
public class Address
{
    int hauseNo;
    string city

    public Address (int hauseNo, string city)
    {
        this.hauseNo = hauseNo;
```

```
    this.city = city;
}
}
```

Xml File

```
<bean id = "s1" class = "com.beans.student"
                autowire = "constructor" />

<constructor-arg value = "101" index = "0" />
<constructor-arg value = "ram" index = "1" />

</bean>

<bean id = "address" class = "com.beans.Address" />

<constructor-arg value = "10" />
<constructor-arg value = "Indore" />
</bean>
```

Isme variable name and bean id same hona chaiye jise ham inject kar rhe hai.

Constructor (parameterized) same field ka hona chaiye.

10 FRIDAY
2023 FEBRUARY

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

WEEK 06
041-324

Java Based Configuration

The difference in xml and Java based Configuration is :-

1). We will replace somefile.xml with the Java annated with @Configuration.

2). To configure a bean instead of writing a bean tag, we will write one method for each bean configuration with an annotation @Bean.

3). Properties of a Bean will be set directly using constructor or setter methods

4). Component-scan, annotation-config which you configured through tags will be configured using annotation.

5). Stereotype annotation (@component) will remain same in xml and Java both

6). All other annotation like @Autowired etc. will remain same in xml and Java Configuration both.

Xml Based

```
<context : component-scan
base package = "com.beans" >

  <beans>

  <bean id = "s1" class
      = "com.beans.student" >

<property name = "id"
  value = "101" />

<property name = "name"
  value = "ram" />

  </bean>
        </beans>
```

Java Based.

```
@componentScan (basePackages
  = "com.beans")

@Configuration
public class ProConfig.
{

  @Bean
  public student s1()
  {
    student s = new student();
    s.setId(101);

    s.setName("ram");
    return s;

  }
}
```

* Name of method will be id of bean (s1).

* Component scan will be replaced by ComponentScan annotation.

```
ApplicationContext ap = new AnnotationConfigApplicationContext
                        (Proconfig.class);
```

13 MONDAY
2023 FEBRUARY

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

WEEK 07
044-321

Java Configuration

Student class

```
public class Student
{
    private int id;
    private Address address;

    → Getter & setters, Cons ←
}
```

Address class

```
public class Address
{
    private string street;
    private string city;
    private string state;

    → getter, setter Cons ←
}
```

ProjectConfig class

```
@Configuration
@ComponentScan (base package = "com.beans")

public class ProjectConfig
{
```

```
@Bean
public Student s1()
{
    Student s = new Student();
    s.setId(101);
    s.setAddress(adr1());
    return s;
}
@Bean
public Address adr1()
{
    Address ad = new Address("Abro", "Indore", "MP");
    return ad;
}
}
```

Student Entry Class

```
public class StudentEntry
{
psvm(String ar[])
{
ApplicationContext ap = new AnnotationConfigContext(Project
                                                    config.class);

Student s1 = ap.getBean("s1", student.class);
System.out.println(s1);
}
}
```

15 WEDNESDAY
2023 FEBRUARY

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

WEEK 07
046-319

AutoWire in Java Configuration

Student class

```
public class Student
{
    private int id;
    Private Address add;

    @Autowired
    private Address add;

        → Const. getter & setter ←
}
```

Address class

→ same as Before ←

ProjectConfig class

→ annotations

```
public class ProjectConfig
{
    @Bean
  public student s1()
  {
      Student s1 = new student();
          s1.setId(101);
  }   return s;
```

→ Not initializing add because its autowired

```
@Bean
public Address add()
{
    return new Address("Abc", "Indore", "MP");
}
}
```

Using @Component & Autowire in Java Confi

Student class

```
@component
public class student
{
    private int id;
    @Autowired
    Address add;
    → cons, getter & setter ←
}
```

Address Class

```
@component
public class Address
{
    private string city;
    private string state;
    → getter setter Const ←
}
```

ProjectConfig Class

```
@Configuration
@ComponentScan (basePackage = "com.beans")
public class ProjectConfig
{
}
```

StudentEntry Class

```
public class StudentEntry
{
    public static void main (String ar[])
{
ApplicationContext ap = AnnotationConfigApplicationContext (ProjectConfig.class)

Student s = ap.getBean ("S1", Student.class);
    S1.setId(101);
~~Address a=s1.getAddd();~~
                Address a = S1.getAdd();
                a.setCity ("Indore");
                a.setState("MP");
    S1.setAdd(a);
}}
```

Hamne student & Address class ko @component bnaya means unki bean hame nhi bnani pdegi spring bnayega. just like xml component me hota hai