



# Spring Inversion Of Control

# Agenda

1

**Inversion of Control (IoC)**

2

**Dependency Injection**

3

**Dependency Injection - Autowiring**

# Objectives

At the end of this session, you will be able to:

- Understand What is Inversion Of Control otherwise called as IOC
- Understand Implementation Of Dependency Injection
- Understand the usage of Autowiring in Dependency Injection

# Inversion of Control (IoC)

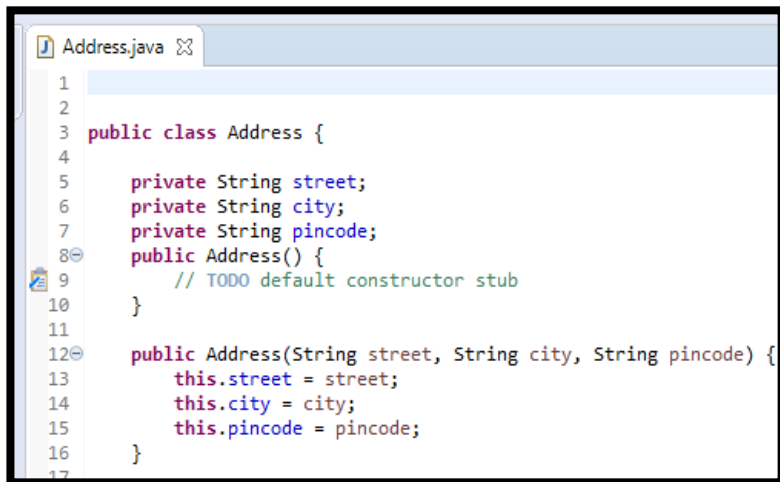


# Inversion of Control (IoC)

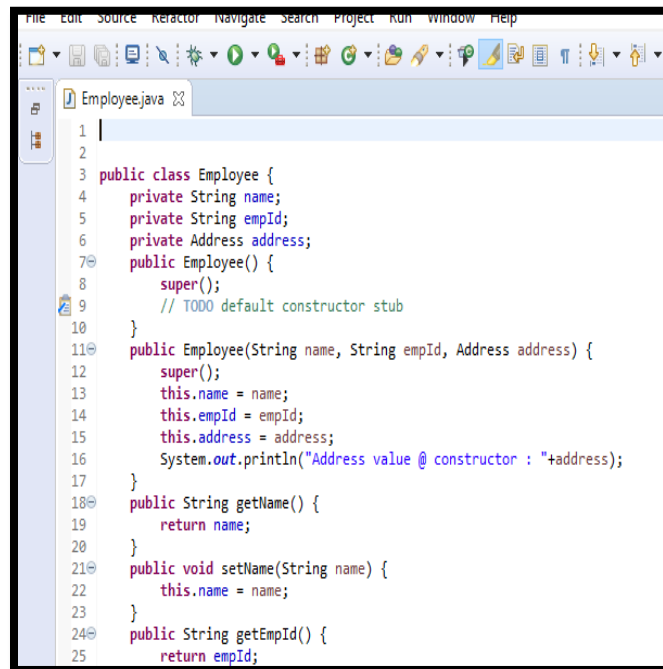
- Inversion Of Control, In Software Engineering is explained as a Programming Technique where the *object management* (creation & association between the objects) is *done by the framework* and not by the client
- In other words: Instead of Clients having the control to establish relationship between Components, now the Framework carries this job
- There are several mechanisms to implement IOC
  - Strategy design pattern
  - Factory Pattern
  - Dependency Injection

# Inversion of Control (IoC)

- Let us understand this in a simple form.
  - We have an Employee class which needs Employee address as a separate object association
  - In general we'll create the object in the Employee class itself or from client application

A screenshot of a code editor showing the implementation of an Address class in Java. The class has three private attributes: street, city, and pincode, all of type String. It includes a no-argument constructor with a TODO comment and a parameterized constructor that initializes the attributes from the provided arguments.

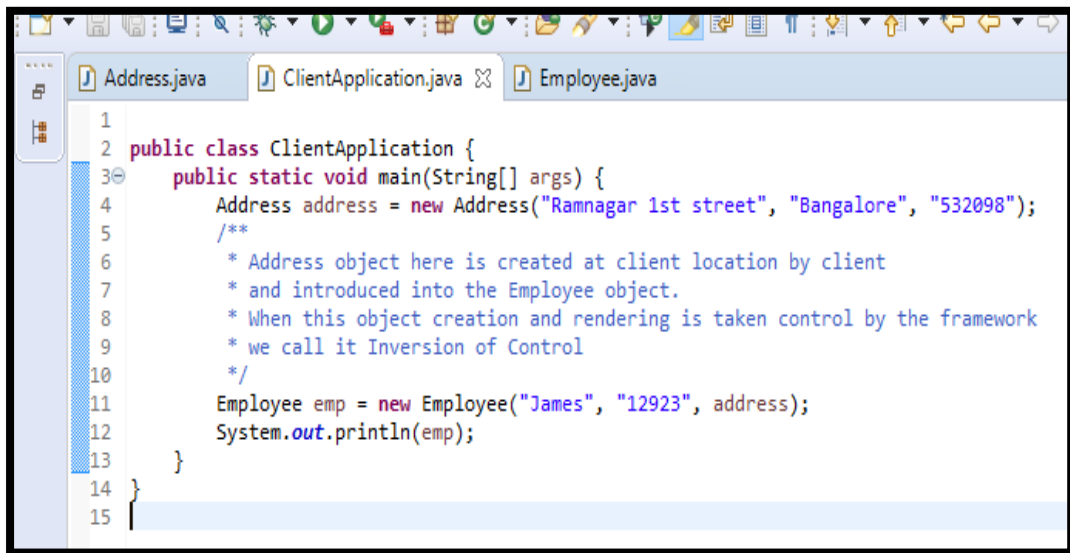
```
1  
2  
3 public class Address {  
4  
5     private String street;  
6     private String city;  
7     private String pincode;  
8     public Address() {  
9         // TODO default constructor stub  
10    }  
11  
12     public Address(String street, String city, String pincode) {  
13         this.street = street;  
14         this.city = city;  
15         this.pincode = pincode;  
16    }  
17
```

A screenshot of a code editor showing the implementation of an Employee class in Java. The class has three private attributes: name, empId, and address. name and empId are of type String, and address is of type Address. It includes a no-argument constructor with a TODO comment, a parameterized constructor that initializes the attributes and prints the address, and two getter methods for name and empId.

```
1  
2  
3 public class Employee {  
4     private String name;  
5     private String empId;  
6     private Address address;  
7     public Employee() {  
8         super();  
9         // TODO default constructor stub  
10    }  
11     public Employee(String name, String empId, Address address) {  
12         super();  
13         this.name = name;  
14         this.empId = empId;  
15         this.address = address;  
16         System.out.println("Address value @ constructor : "+address);  
17    }  
18     public String getName() {  
19         return name;  
20    }  
21     public void setName(String name) {  
22         this.name = name;  
23    }  
24     public String getEmpId() {  
25         return empId;  
26    }
```

# Inversion of Control (IoC)

- Let us understand this in a simple form.
  - Let us see an example of client application



```
1 public class ClientApplication {
2
3     public static void main(String[] args) {
4         Address address = new Address("Ramnagar 1st street", "Bangalore", "532098");
5         /**
6          * Address object here is created at client location by client
7          * and introduced into the Employee object.
8          * When this object creation and rendering is taken control by the framework
9          * we call it Inversion of Control
10        */
11        Employee emp = new Employee("James", "12923", address);
12        System.out.println(emp);
13    }
14 }
15 }
```

- Client or some class in between has to create the object and render it
- Here the Address Object is a data object
- hence we may require more than one object depending on the data
- When the business component is a service object then client should create a singleton object

- Such Dependencies are removed and taken care by the framework; which we call **Inversion Of Control**

# Dependency Injection (DI)





# Dependency Injection (DI)

- Let us learn how to Implement Spring Inversion of Control using Dependency Injection
- Dependency Injection is a form of IOC that removes explicit dependence on container APIs
  - ordinary Java methods are used to inject dependencies such as collaborating objects or configuration values into application object instances.
- The two major flavors of Dependency Injection are
  - *Setter Injection* (injection via JavaBean setters)
  - *Constructor Injection* (injection via constructor arguments).

# Dependency Injection (DI)

## ■ Setter Injection

- using setter methods in a bean class, the Spring IOC container will inject the dependencies
- Let us take the same Employee Bean and Address bean example
- Configuration file to express *setter injection* of Address object to Employee Bean

```
<bean id="addressBean" class="com.wipro.bean.Address">
    <property name="street" value="Pritech Park"></property>
    <property name="city" value="Bengaluru"></property>
    <property name="pincode" value="560037"></property>
</bean>

<bean id="employeeBean" class="com.wipro.bean.Employee">
    <property name="name" value="ALLEN"></property>
    <property name="empId" value="2000123"></property>
    <property name="address" ref="addressBean"></property>
</bean>
```

addressBean Object is injected as setter value to the EmployeeBean

# Dependency Injection (DI)

- **Constructor Injection**
- using parameterized constructor of the bean, the Spring IOC container will inject the dependencies while creating the object
- The constructor will take arguments based on number of dependencies required

```
<bean id="addressBean" class="com.wipro.bean.Address">  
  <property name="street" value="Pritech Park"></property>  
  <property name="city" value="Bengaluru"></property>  
  <property name="pincode" value="560037"></property>  
</bean>
```

addressBean Object is  
injected as constructor  
value to the EmployeeBean

```
<bean id="employeeBean" class="com.wipro.bean.Employee">  
  <constructor-arg index="0" type="java.lang.String" value="ALLEN"></constructor-arg>  
  <constructor-arg index="1" type="java.lang.String" value="2000123"></constructor-arg>  
  <constructor-arg index="2" type="com.wipro.bean.Address" ref="addressBean"></constructor-arg>  
</bean>
```

# Dependency Injection (DI) : Sample Demo

- Create the Bean class Employee and Address

Employee.java

```
public class Employee {  
    private String name;  
    private String empId;  
    private Address address;  
    public Employee(String name, String empId, Address  
    address){  
        this.name = name;  
        this.empId = empId;  
        this.address = address;  
    }  
  
    // TO DO Getters and Setters for all properties  
}
```

Address.java

```
public class Address {  
    public Address(){  
  
    }  
    private String street;  
    private String city;  
    private String pincode;  
  
    // TODO Getters and Setters for all Properties  
}
```

For sample demo refer :: [Spring IOC Demo Projects.pdf](#)

# Dependency Injection (DI) : Sample Demo

- Create the ApplicationContext.xml for *Constructor Injection*

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">
<bean id="addressBean" class="Address">
    <property name="street">
        <value>Main Street</value>
    </property>
    <property name="city">
        <value>Bangalore</value>
    </property>
    <property name="pincode">
        <value>567456</value>
    </property>
</bean>
<bean id="employeeBean" class="Employee">
    <constructor-arg index="0" type="java.lang.String" value="MyName"/>
    <constructor-arg index="1" type="java.lang.String" value="001"/>
    <constructor-arg index="2">
        <ref bean="addressBean"/>
    </constructor-arg>
</bean></beans>
```

# Dependency Injection (DI) : Sample Demo

- Create the main class to test the app

```
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;
import org.springframework.core.io.Resource;

public class ConstructorInjection {
    public static void main(String args[]){
        Resource xmlResource = new FileSystemResource("ApplicationContext.xml");
        BeanFactory factory = new XmlBeanFactory(xmlResource);
        Employee employee = (Employee)factory.getBean("employeeBean");
        Address address = employee.getAddress();
        System.out.println(employee.getName());
        System.out.println(employee.getEmpId());
        System.out.println(address.getCity());
        System.out.println(address.getStreet());
        System.out.println(address.getPincode());
    }
}
```

# Dependency Injection (DI) : Sample Demo

- Edit applicationContext.xml (*for Setter Injection*)
- Employee.java (for Setter Injection) : no constructor with 'address' as parameter, only setter & getter methods

```
<beans ...>
    <bean id="addressBean" class="Address">
        <property name="street" value="Normal Street" />
        <property name="city" value="Bangalore" />
        <property name="pincode" value="567456" />
    </bean>
    <bean id="employeeBean" class="Employee">
        <property name="name" value="MyName"/>
        <property name="empId" value="001"/>
        <property name="address" ref="addressBean"/>
    </bean>
</beans>
```

- Again execute the main application to experience the Setter Injection

# DI Autowiring





# Dependency Injection - Autowiring

- Object Injection can be automated by the concept of autowiring
- Instead of explicit referencing of the dependent object through setter or through constructor; it can be automated by setting *autowire* attribute of the bean
- Different values of autowire attribute are
  - `byType` - setter Injection
  - `byName` - setter Injection
  - `constructor` - Constructor Injection

```
<bean id="addressBean" class="com.wipro.bean.Address">
  <!-- Using Constructor for passing the values -->
  <constructor-arg index="0" type="java.lang.String" value="Pritech Park"></constructor-arg>
  <constructor-arg index="1" type="java.lang.String" value="Bengaluru"></constructor-arg>
  <constructor-arg index="2" type="java.lang.String" value="560037"></constructor-arg>
</bean>

<bean id="employeeBean" class="com.wipro.bean.Employee" autowire="byType">
  <property name="name" value="ALLEN"></property>
  <property name="empId" value="2000123"></property>
</bean>
```

# Dependency Injection - Autowiring

- byType looks for a bean definition of the required Object's Type
- If there are more than one Bean definition found for the required Object Type, it raises *NoUniqueBeanDefinitionException* Exception

```
<bean id="addressBean" class="com.wipro.bean.Address">
  <!-- Using Constructor for passing the values -->
  <constructor-arg index="0" type="java.lang.String" value="Pritech Park"></constructor-arg>
  <constructor-arg index="1" type="java.lang.String" value="Bengaluru"></constructor-arg>
  <constructor-arg index="2" type="java.lang.String" value="560037"></constructor-arg>
</bean>

<bean id="addressBean1" class="com.wipro.bean.Address">
  <!-- Using Constructor for passing the values -->
  <constructor-arg index="0" type="java.lang.String" value="new World"></constructor-arg>
  <constructor-arg index="1" type="java.lang.String" value="Delhi"></constructor-arg>
  <constructor-arg index="2" type="java.lang.String" value="450037"></constructor-arg>
</bean>

  <bean id="employeeBean" class="com.wipro.bean.Employee" autowire="byType">
    <property name="name" value="ALLEN"></property>
    <property name="empId" value="2000123"></property>
  </bean>
```

# Dependency Injection - Autowiring

- byName injects the bean id value matching to the property name of the required Object

```
<bean id="address" class="com.wipro.bean.Address">
  <!-- Using Constructor for passing the values -->
  <constructor-arg index="0" type="java.lang.String" value="Pritech Park"></constructor-arg>
  <constructor-arg index="1" type="java.lang.String" value="Bengaluru"></constructor-arg>
  <constructor-arg index="2" type="java.lang.String" value="560037"></constructor-arg>
</bean>

<bean id="employeeBean" class="com.wipro.bean.Employee" autowire="byName">
  <property name="name" value="ALLEN"></property>
  <property name="empId" value="2000123"></property>
</bean>
```

```
public class Employee {
    private String name;
    private String empId;
    private Address address; //bean id value is same as the Address variable name
    //Constructors and Getters and Setters are given
}
```

# Dependency Injection - Autowiring

For detailed demo Refer :

[Spring DI - Autowiring Demo.pdf](#)

# Summary

- In this module, we have learnt
  - Inversion of Control methodologies
  - Dependency Injection
    - Setter Injection
    - Constructor Injection
    - Autowiring



**Thank you**