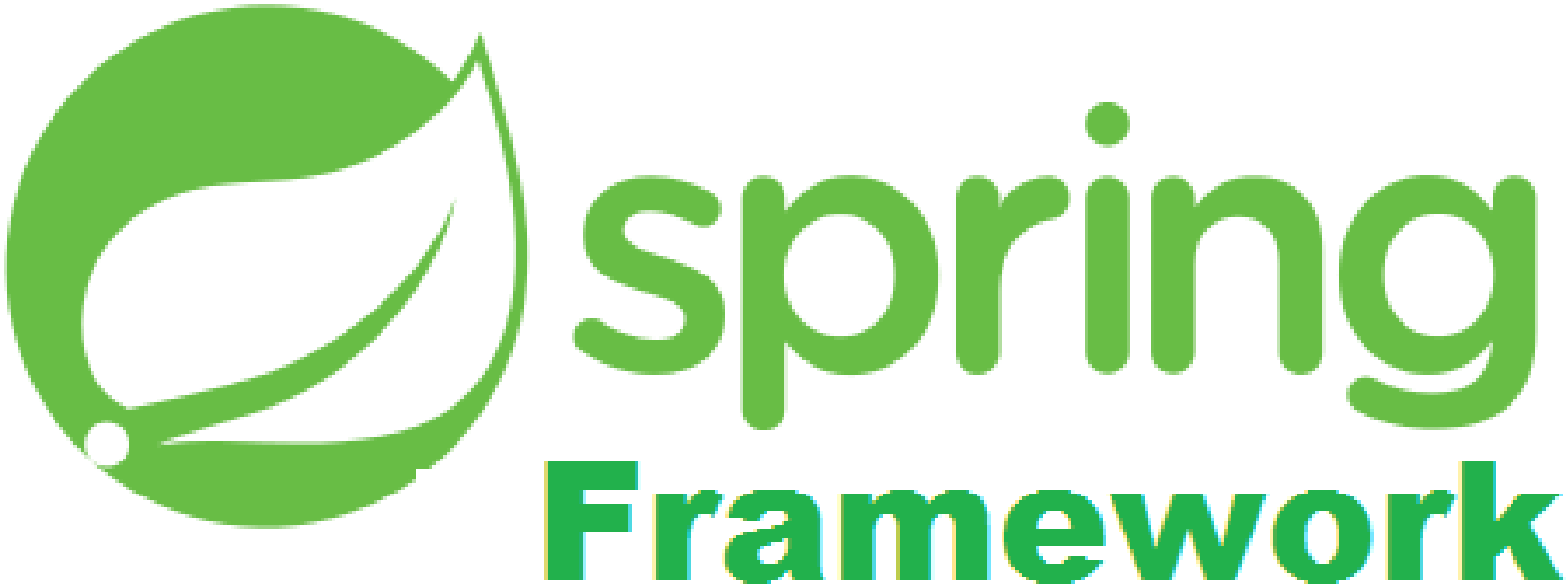

spring
Framework

# What is Spring?

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

Spring framework makes the easy development of JavaEE application.

It was developed by Rod Johnson in 2003

# Spring Core Concepts

1. Spring Framework
2. What is IOC?
3. Dependency Injection.
4. Types of dependency injections
   **1. Setter based dependency Injection**
   **(Data Types Dependency Injection)**
   a. ) Primitive types (**byte, short, long, int, boolean, char**)
   b.) Collection Types ( List, Set, Map and Properties)
   c.) Reference Type
   **2. Constructor based dependency Injection**
   **Bean[ Scope (Singleton, prototype, request, session, application, web socket)]**
5. Life Cycle of Bean & configuration techniques
   a.) XML
   b.) Spring Interface ( InitializingBean, DisposableBean
   c.) Annotation( @Bean, @Scope @Value)
6. Auto-wiring
   a.) XML
   ( **Auto wiring Modes**) → No, byName, byType, Constructor, auto detect(It is deprecated since spring 3)

   b.) Annotation (@Autowired, @Qualifier, @Value)

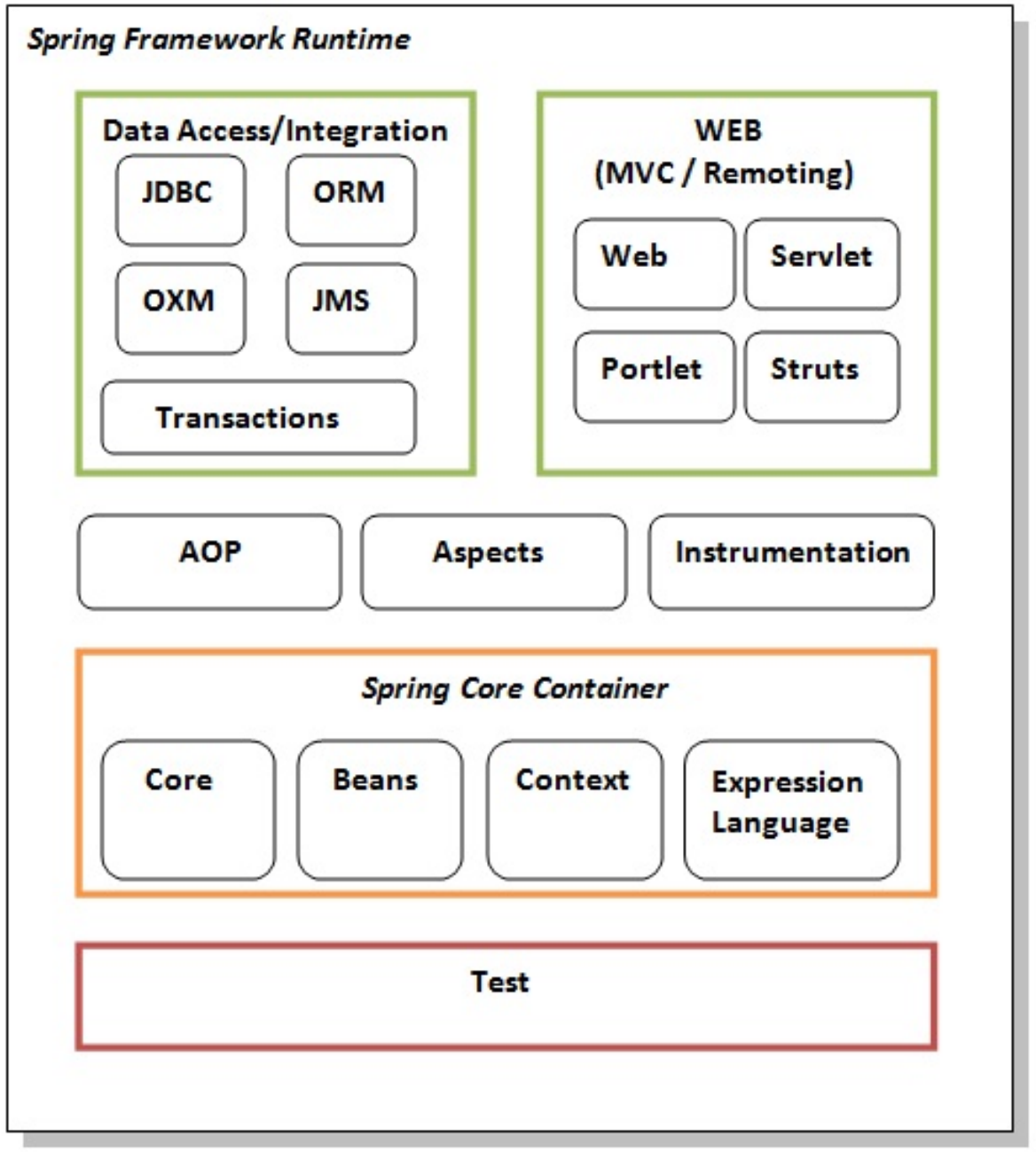
7. @Component, @Value, @Configuration,@Bean

The **Spring framework** comprises of many modules such as core, beans, context, expression language, AOP, Aspects, Instrumentation, **JDBC,** ORM, OXM, JMS, Transaction, **Web**, **Servlet**, Struts etc.

These modules are grouped into

Test,
Core Container,
AOP,
Aspects,
Instrumentation,

Data Access / Integration, Web (MVC / Remoting) as displayed in the following diagram.



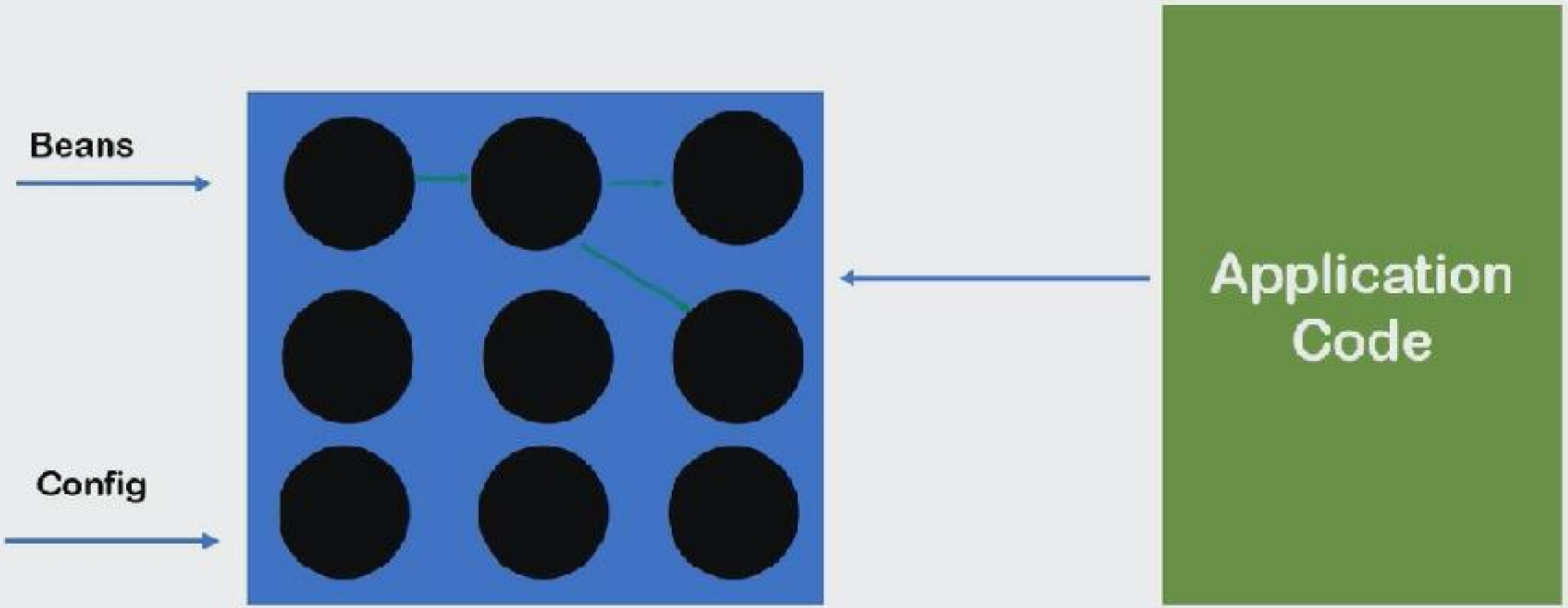
# IOC(Inversion of Control)

- Spring IOC Container

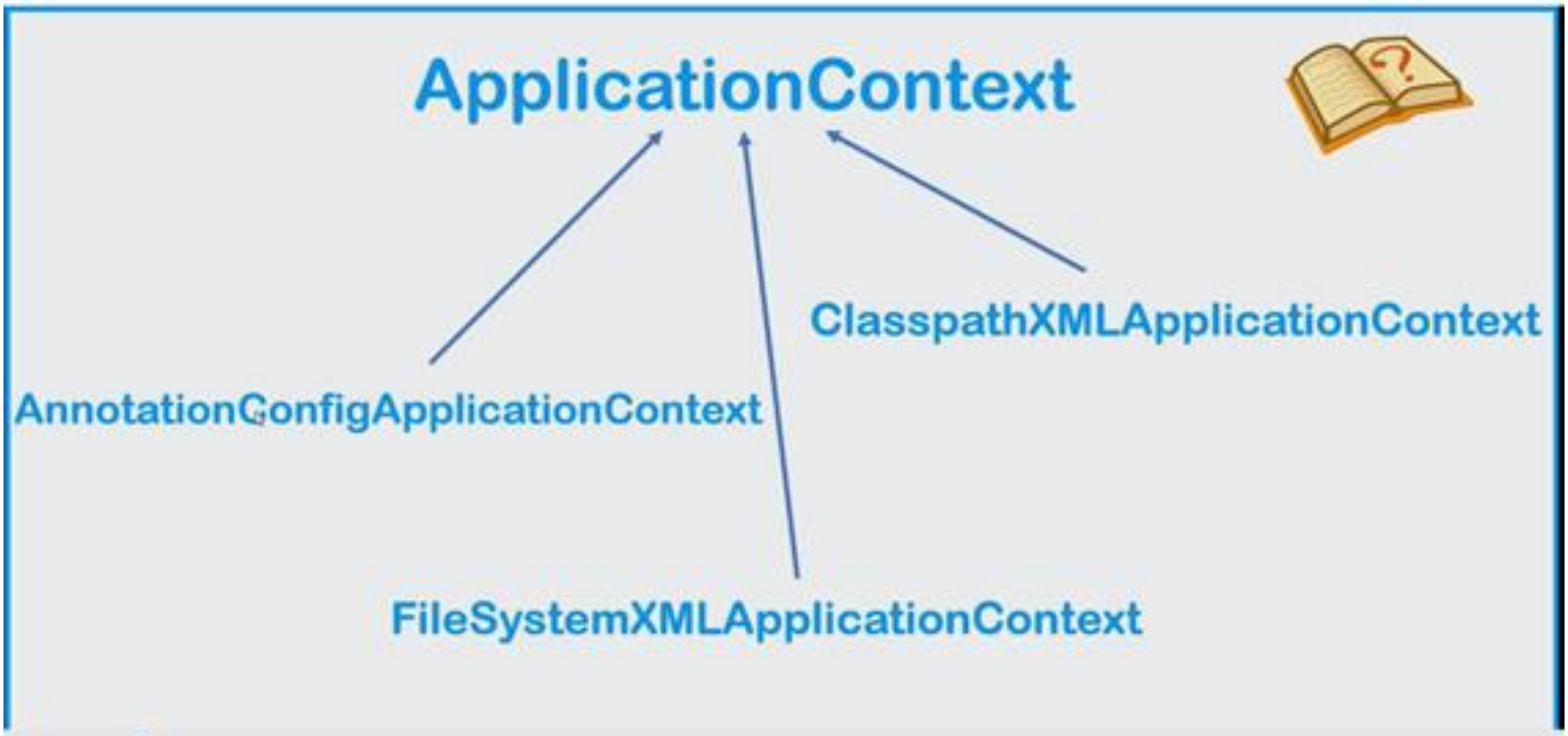
  It creates the object, hold them in memory, inject(Store) one object into another object as required.

  It maintains the life cycle of object.

# Spring IoC Container



**ApplicationContext** is a interface that represent IOC container.



# Dependency Injection done in two ways

1. Using Setter Injection

2. Using Constructor Injection

# Dependency Injection

It is design pattern

```
class Ramu
{
        Geeta ob;

        public void doWork()
        {

        }

}
```

```
class Geeta
{
        public void doWork()
        {

        }
}
```

# Configuration File

**Where we create beans and its Dependency**

# Data types Dependency

## 1. Primitive Data types

**Byte, short, int, long, char, float, double, boolean**

# Data types Dependency

## 2. Collections types

**List, Set, map and properties**

# Data types Dependency

## 3. Reference types

**Other classes Object**

# Dependency Injection

```
class Student
{

        int id;
        String name;
        Address address;

}
```

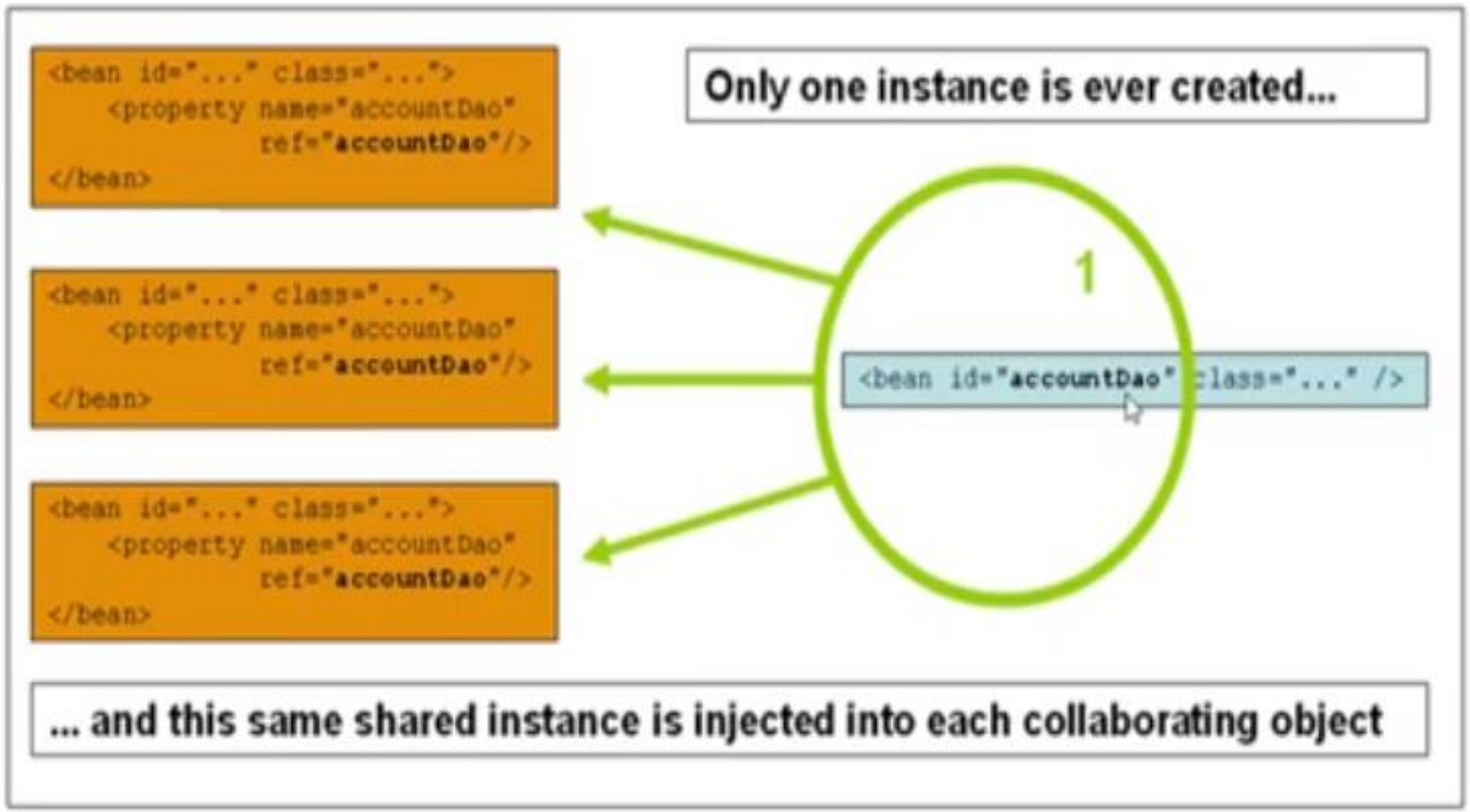
```
class Address
{
        String street;
        String city;
        String state;
        String country;
}
```

Softwares:
1=>Eclipse/Netbeans/IntelliJ
2=>TomcatServer
3=>Mysql for database
4=>Sqlyog , workbench or phpmyadmin for mysql gui

---

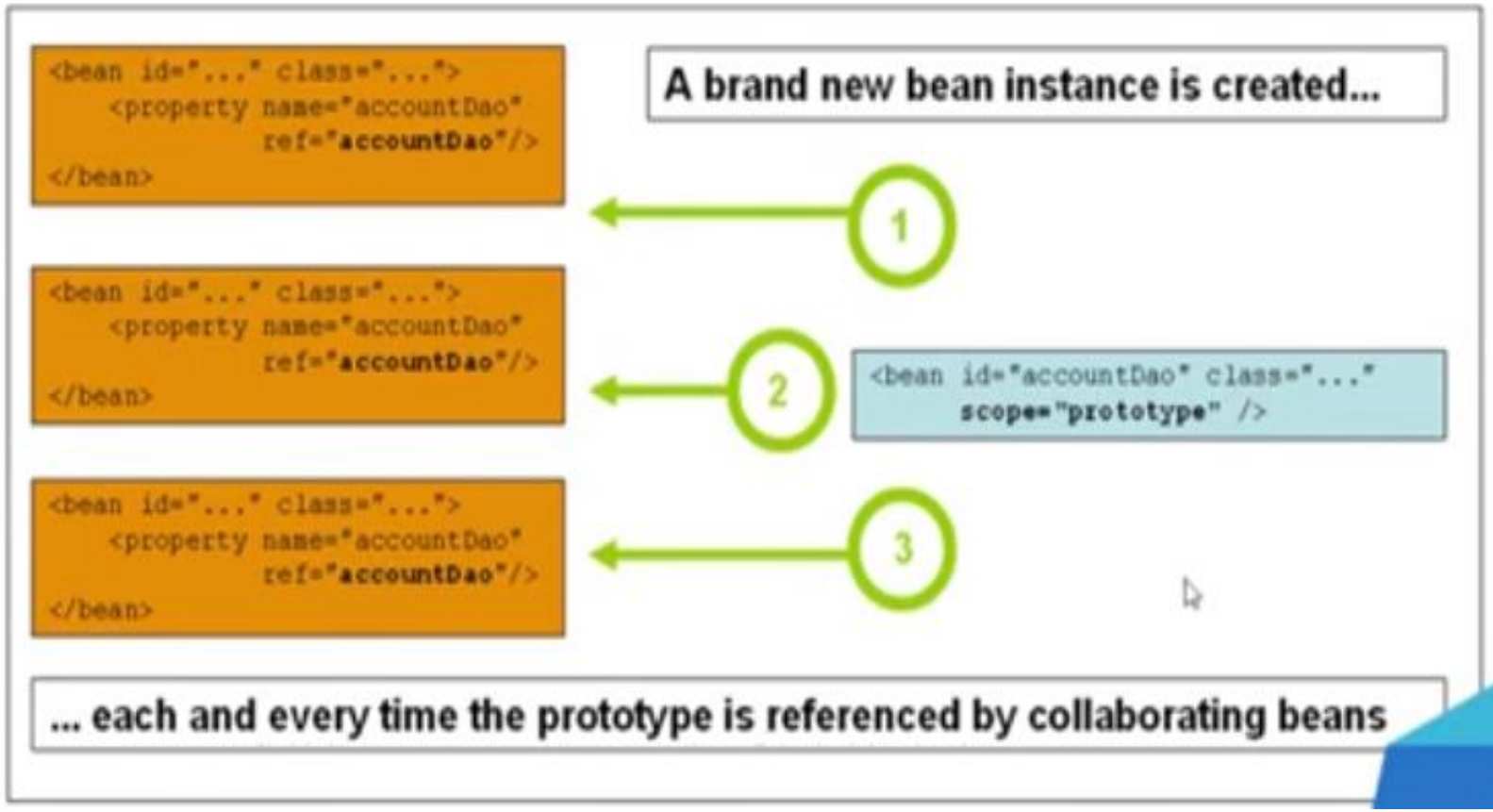
1-Create maven project
2-Adding dependencies =>spring core , spring context
3-creating beans
4-creating configuration file=>config.xml
5-Setting Injection
6-Main class : which can pull the object and use

# Singleton Scope




- The singleton scope is the default scope in Spring

# Prototype Scope




- You should use the prototype scope for all stateful beans and the singleton scope for stateless beans

---

What is an Annotation?

---

-> Annotation is used to provide meta data

-> From Java 1.5v onwards we have annotations support.

-> Annotations came into market as an alternative XML configurations

-> Spring framework also having support for Annotations

-> Spring Boot also having support for Annotations

---

## Frequently Used Annotations From Spring Framework

---

@Component
@Service
@Repository

@Configuration
@Bean

@Autowired
@Qualifier

What is Spring Bean?

The class which is managing by IOC is called as Spring Bean. How to represent java class as Spring Bean ?

1. Using Xml Configuration

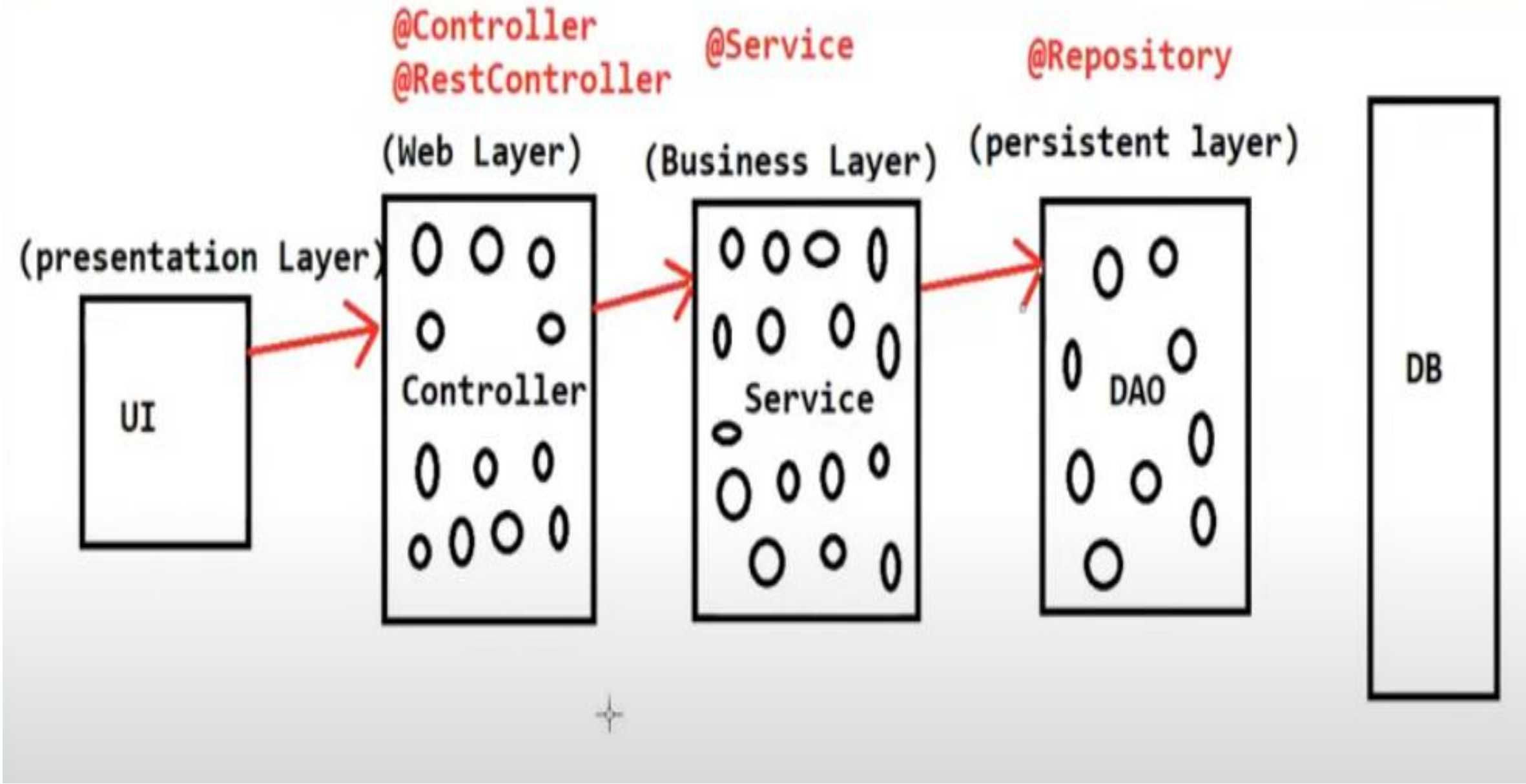
```
<bean id="car" class="rkg.boot.Car"/>
```

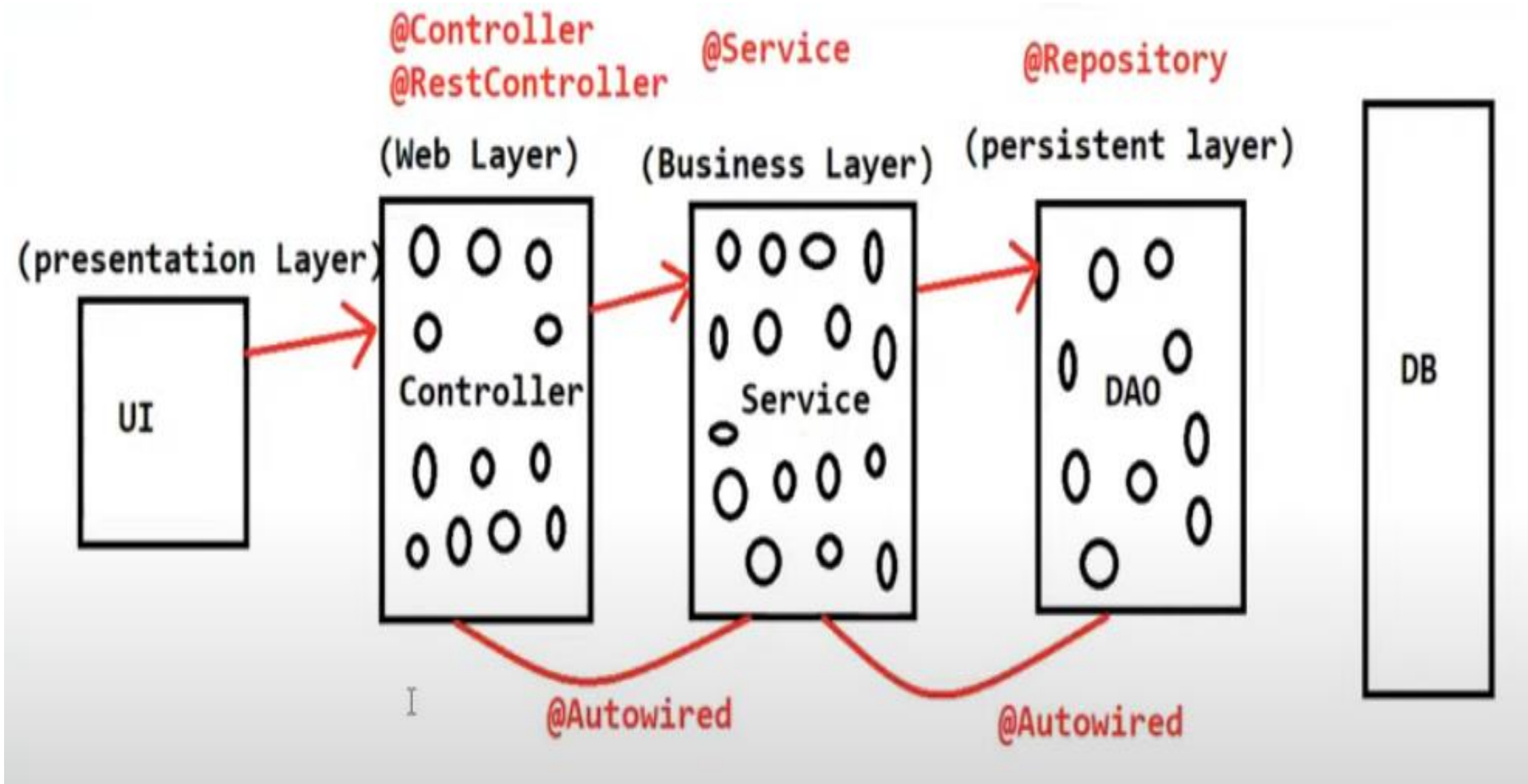
2. Using Annotation
   (@Component, @Service and @Repository)

```
@Component
public class Car{

    //properties
    //methods
}
```

```
@Service
public class UserService{

    //re-usable methods
    //with business logic
}
```

```
@Repository
public class UserDao{

    //methods with DB
Logic

}
```


@Controller
@RestController
@Service
@Repository
(Web Layer)
(Business Layer)
(persistent layer)
(presentation Layer)
UI
Controller
Service
DAO
DB


@Controller
@RestController
@Service
@Repository
(Web Layer)
(Business Layer)
(persistent layer)
(presentation Layer)
UI
Controller
Service
DAO
DB
@Autowired
@Autowired

# Life Cycle Methods

Spring provide two important methods to every bean

**Initialization code Loading config, Connecting db, Webservice etc**

public void init()

public void destroy()

**Clean up code**

We can change the name of these method
But signature must be same

# Configure Technique

- Xml
- Spring Interface
- Annotation

# Life Cycle

Spring bean

Configuration Xml File

25

Setter method

init()

Then we Read and use the bean

destroy()

# Using Interfaces

InitializingBean

DisposableBean

# Using Annotations

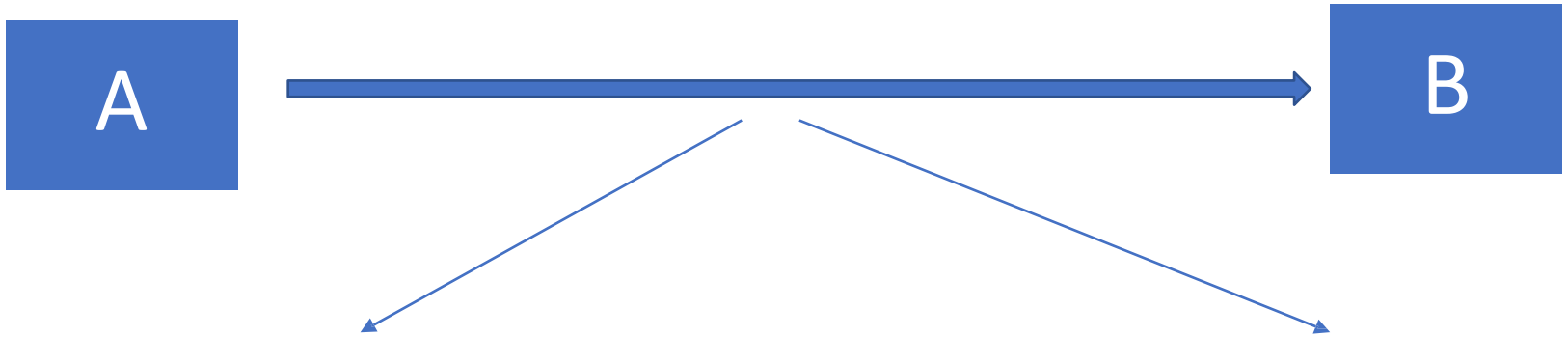
@PostConstruct

@PreDestory

# Autowiring in Spring

- Feature of spring Framework in which spring container inject the dependencies automatically .

- Autowiring can't be used to inject primitive and string values. It  works with reference only.




Manually

<ref bean=""/>

Programmer

Automatically

Spring container

# Autowiring

XML

Annotations

## Autowiring Modes

- no
- byName
- byType
- constructor
- autodetect

*It is deprecated since Spring 3.*

- @Autowired

# Autowiring Advantages

- **Automatic**
- **less code**

# Autowiring Disadvantages

- **No control of programmer.**
- **It can't be used for primitive and string values.**

# Stereotype Annotations

- XML <bean />

@Component Class Student
{

}

<context:component-scan base-package=“ ” />

# Stereotype Annotation

```
@Component  Class Student
{


}

Student student=new Student()
```

# Configure bean scope

```
<bean class=“ ” name=“ ” scope=“ ” />

@Component  @Scope( “        ”)  Class
Student
{


}
```

# @Configuration
# @Bean

# Bean Scope

Singleton

prototype

request

session

globalsession