Spring Basics + Spring Boot & Micro services

Pre-Requisites

1) Core Java

2) Adv Java (JDBC + Servlets)

3) Oracle (SQL)

4) Hibernate (Basics of ORM)

Course Content

1) Spring Core

2) Spring JDBC / DAO

3) Spring Boot

4) Spring Data JPA

5) Spring Web MVC

6) Restful Services (Spring REST)

7) Spring Cloud

8) Microservices

9) Spring Security

10) Spring Boot Integrations

- Apache kafka
- Redis cache
- Docker
- Unit Testing (Junit)
- Logging

Programming Language : Java

- Language Fundamentals
- Syntaxes

-> Standalone apps

Technologies: JDBC + Servlets + JSP

-> Database communication using JDBC

-> Servlets for web application development

-> JSP (Presentation logic)

=> Web applications

Java Frameworks: Hibernate + spring --> Spring Boot

Core Java: To develop Standalone applications

JDBC: To develop persistence logic

Servlets: To develop web applications

JSP: To develop Presentation logic

What is Framework?

=> Framework is a semi developed / readymade software

=> Frameworks will provide some common logics required for several projects development.

=> Frameworks provides re-usable components (classes & interfaces)

=> Frameworks will reduce burden on the developers

=> IN Java community we are having 2 types of frameworks

1) ORM frameworks (Ex: Hibernate) - Object Relational Mapping

2) Web Frameworks (Ex: Struts) - 2001

Note: Spring is called as Application development Framework (interface21)

1st spring version came into market 2003/2004

Current version of Spring is 6.x

-> Spring is not single framework

-> Spring Framework is collection of frameworks (Modules)

Spring 1.x ---> 7 Modules

Spring 2.x ---> 6 Modules

Spring 3.x ----> 20+ Modules

-> Spring Framework is versatile framework

-> Spring is loosely coupled framework

What is Spring?

-> Spring is a java based application development framework

-> By using spring we can develop end to end application

-> Spring is developed by using JSE & JEE

-> Spring framework developed by Rod Johnson

-> First version of spring came into market in 2004 (Spring 1.x v)

-> The current version of Spring is 6.x (2022)

-> Spring is developed in modular fashion

-> Spring Modules are loosely coupled

-> Spring is versatile framework (it can integrate with any other framework)

Spring Framework Modules

1) Spring Core

2) Spring Context

3) Spring DAO

4) Spring ORM

5) Spring AOP

6) Spring Web MVC

7) Spring REST

8) Spring Data JPA

9) Spring Cloud

10) Spring Security

11) Spring Social

12) Spring Batch

Spring Core

-> Base module of spring framework

-> Core Module providing fundamental concepts of Spring those are

IOC : Inversion Of Control

DI : Dependency Injection

-> IOC & DI are used to develop classes with loosely coupling

-> IOC will take care of java objects life cycle (Spring Beans)

Spring Context

-> TO manage configurations in Spring application we will use Context Module

-> It provides configuration support required for managing classes

Spring AOP

AOP : Aspect Oriented Programming

-> AOP is used to separate cross cutting logics in the application

Cross Cutting / Secondary / Helper Logics

Ex : Security, transaction, Logging, Auditing & exception handling etc...

Spring DAO / Spring JDBC

-> Spring JDBC is extension for Java JDBC

-> To simplify persistence logic development we can use Spring JDBC

JDBC Logic

---

1. load driver
2. get connection
3. Create statement
4. Execute query
5. process result
6. Close connection

spring jdbc

---

1. Execute query

2. process result

=> Spring JDBC provided predefined classes to perform DB operations

Ex: JdbcTemplate, NamedJdbcTemplate etc...

Spring ORM

=> SPring ORM module is extension for existing ORM frameworks

ORM - Object relational mappping

=> To suppport ORM integrations we have Spring ORM module

Spring ORM = Spring + ORM Framework (Ex: Hibernate)

Hibernate

---

1. Create Config obj

2. create session factory

3. create session

4. begin tx

5. execute methods

6. commit tx

7. close session

8. close sf

spring orm

HibernateTemplate.save(entityObj)

Spring Web MVC

-> To develop both web & distributed (web services) applications

-> It is used to simplify web layer development in applications

Spring REST

-> To simplify REST API development

Spring Data JPA

-> It is extension for Spring ORM

-> It is providing readymade methods to perform CRUD operations in DB

Spring Security

-> It is used to secure our spring based application

-> We can implement both Authentication & Authorization by using Spring Security

Authentication: Decide who can access our application?

Authorization: Identify logged in user having access for the functionality or not?

Spring Batch

=> Batch means bulk operation

Ex:

sending bulk email to customers

sending bulk sms to students regarding course update

sending bank statement to account holders

read records from file and store into DB

Spring Cloud

-> It provides configurations required for Micro services development

Service Registry

Admin Server

API Gateway

Spring Core

-> Base module of Spring Framework

-> Providing IoC & DI

-> IOC & DI are used to develop classes with loosely coupling

```
public class Engine {
	public int start() {
		// logic
		return 1;
	}
```

```
}
public class Car extends Engine {
	public void drive() {
		int status = super.start();
		if (status >= 1) {
			System.out.println("Journey Started..");
		} else {
			System.out.println("Engine Trouble..");
		}
	}
}
public class App {
	public static void main(String[] args) {
		Car c = new Car();
		c.drive();
	}
}
```

-> Car is extending Engine class

-> Car class can't use inheritance in future

-> Any changes in Engine class will effect on Car class

-> Car is tightly coupled with Engine

Note: It is not recommended to develop classes with Tightly coupling.

```
public class Car {
	public void drive() {
		Engine eng = new Engine();
		int status = eng.start();
		if (status >= 1) {
			System.out.println("Journey Started..");
```

```
                } else {
                        System.out.println("Engine Trouble..");
                }
        }
}
```

-> Car is directly creating Object for Engine

-> Any changes to Engine class will effect on Car class

-> Car is always talking to only one Engine

-> If i want to change from one Engine to another Engine then we should modify Car class code.

Note: Car is tightly coupled with Engine.

What is Dependency Injection?

The process of injecting dependent object into target object using target class variable / setter method / constructor is called as Dependency Injection.

Dependency Injection Types

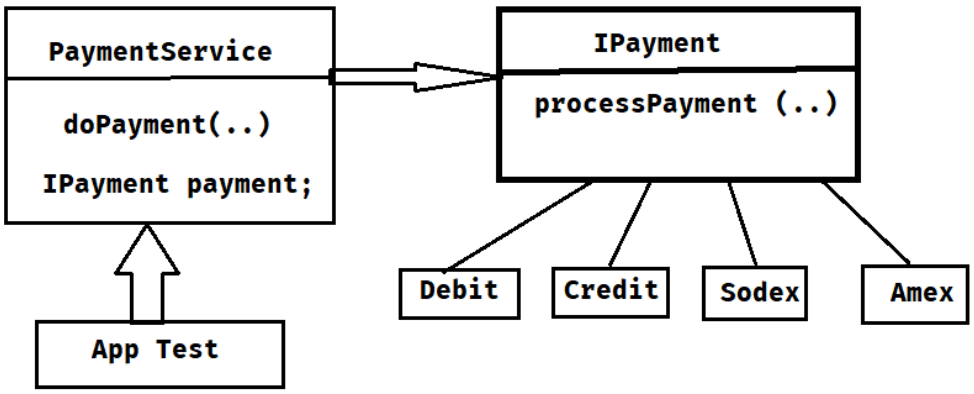
1) Field Injection (variable)

2) Setter Injection (setter method)

3) Constructor Injection (constructor)

Requirement:

Develop an application to perform bill payment. It should support for multiple Payment options (Debit card, Credit Card, Sodex & Amex....)



What will happen when we perform both constructor & setter injections on same variable ?

-> First Constructor injection will happen then it will initialize the variable then setter injection will happen and it will re-initialize the same variable so final value be setter injection value.

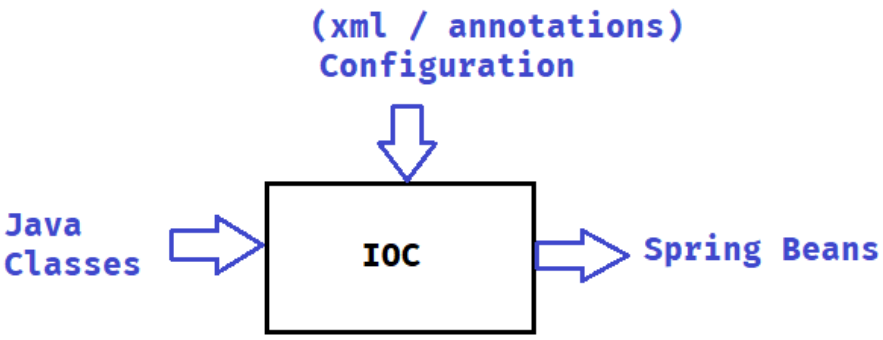
Note: Setter Injection will override Constructor injection.

What is IoC ?

IOC: Inversion of Control

-> IOC is a principle which is used to manage and collaborate dependencies among the objects in the application.

-> In Spring, IOC is responsible for Dependency Injection.



Note: For IOC we need to pass Java Classes + Configuration as input then IOC will perform DI and it will produce Spring Beans.

Spring Bean : The class which is managed by IOC is called as Spring Bean.

How start IOC container?

=> We can start in 2 ways

1) BeanFactory (I) (outdated)

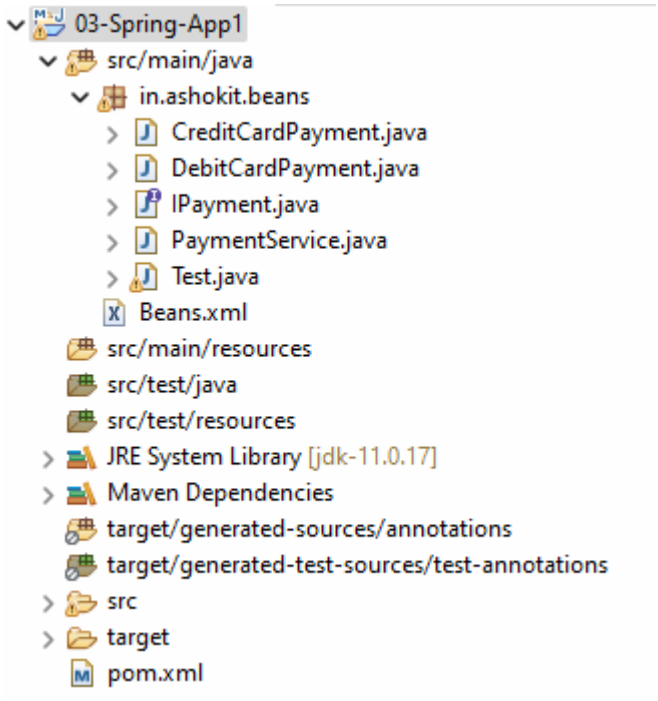
2) ApplicationContext (I) (recommended)

```
ApplicationContext ctxt = new ClassPathXmlApplicationContext(String xmlFile)
```

Creating First Spring Project

1) Open STS IDE

2) Create Maven Project



3) Open pom.xml file and add below dependency

```
<dependencies>
	<dependency>
		<groupId>org.springframework</groupId>
		<artifactId>spring-context</artifactId>
		<version>5.3.25</version>
	</dependency>
</dependencies>
```

Note: After adding dependency verify project Maven Dependencies folder (jars should be displayed)

4) Create Required Java classes

5) Create Bean Configuration File like below

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="
	   http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

	<bean id="credit" class="in.ashokit.beans.CreditCardPayment" />

	<bean id="debit" class="in.ashokit.beans.DebitCardPayment" />

	<bean id="payment" class="in.ashokit.beans.PaymentService">
		<constructor-arg name="payment" ref="credit" />
```

```
    </bean>

</beans>
```

5) Create Test Class and start IOC Container

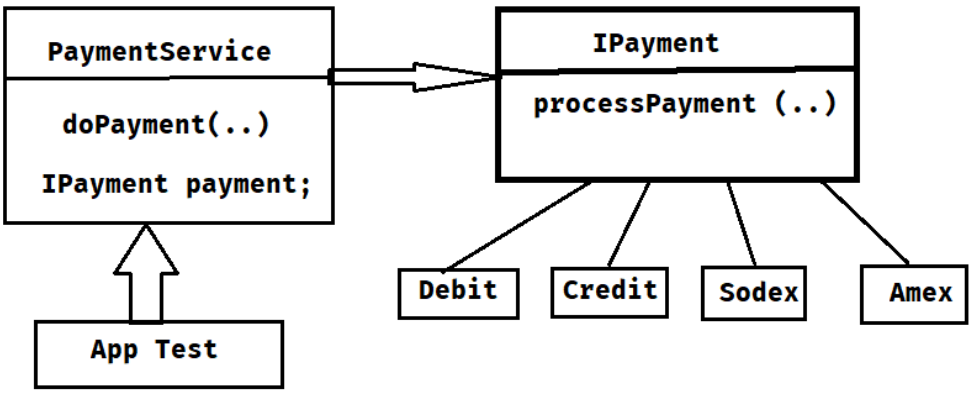
```
6  public class Test {
7
8      public static void main(String[] args) {
9
10         ApplicationContext ctxt =
11                 new ClassPathXmlApplicationContext("Beans.xml");
12
13         Car c = ctxt.getBean(Car.class);
14         c.drive();
15     }
16 }
17
```

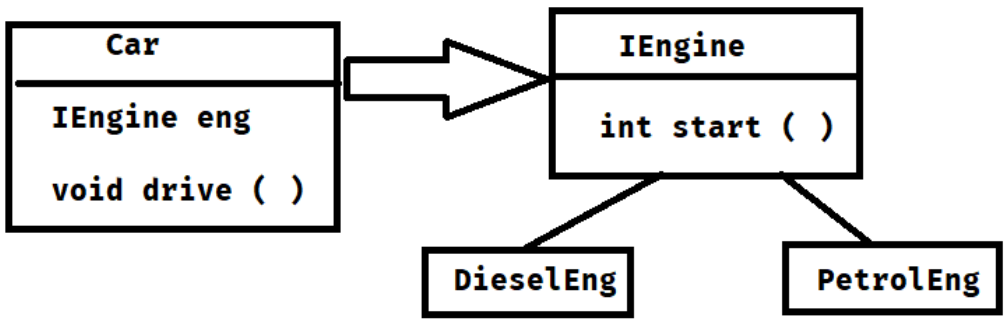
// Constructor Injection will be represented like below

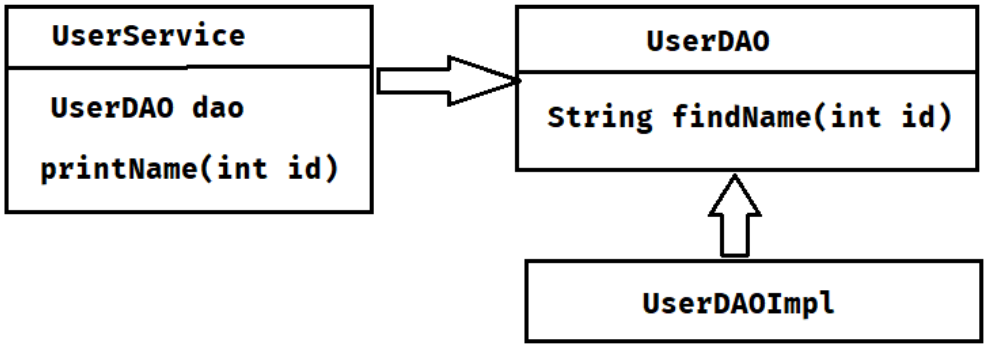
```
<constructor-arg name="payment" ref="credit" />

// setter injection
<property name="payment" ref="debit" />
```

Note: ref attribute represents which object should be injected.







Spring Bean Scopes

-> Bean Scope will decide how many objs should be created for Spring Bean class

-> We have 4 types of scopes

1) singleton (default)

2) prototype

3) request

4) session

-> Singleton means only one object will be created

-> Prototype means every time new object will be created

Note: request & session scopes we will use in spring web mvc.

```
<bean id="car" class="in.ashokit.beans.Car" scope="prototype">
    <property name="eng" ref="petrol" />
</bean>
```

Autowiring

=> Auto wiring is used to identify dependent objects and inject into target objects.

=> Autowiring works based on below modes

1) byName

2) byType

3) constructor

4) none

-> byName MODE will identify dependent bean based on variable name matching with bean id.

-> byType MODE will identify dependent bean based on variable data type.

Note: There is a chance of getting ambiguity in byType mode.

=> If variable data type is interface then we can have multiple implementation classes in this scenario IOC can't decide which bean it has to inject.

=> We can resolve byType ambiguity in 2 ways

1) auto-wire-candiate="false"

2) primary = "true"

```
<beans xmlns="http://www.springframework.org/schema/beans"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

     <bean id="eng12" class="in.ashokit.beans.DieselEngine"
          primary="true" />

     <bean id="eng123" class="in.ashokit.beans.PetrolEngine" />

     <bean id="car" class="in.ashokit.beans.Car" autowire="byType">

     </bean>

</beans>
```

Spring Annotations

@Configuration : To represent java class as config class

@Component : To represent java class as Spring Bean class

@Service : To represent java class as Spring Bean class

@Repository : To represent java class as Spring Bean class

@Scope : To represent scope of spring bean (default : singleton)

@Autowired : Inject dependent into target

@Bean : To customize bean object creation.

@Qualifier : To identify bean based on the given name for DI

@Primary : To give priority for the bean for auto wiring

What is Component Scanning ?

=> It is used to identify Spring Bean classes available in the Project.

=> It will start scanning from current pacakge.

se package name : in.ashokit

AppConfig

- @Configuration

- @ComponentScan

in.ashokit.service ----- scanned

in.ashokit.util ----- scanned

com.ashokit.dao --- not scanned

```
@Configuration
@ComponentScan
public class AppConfig {


}
@Component
public class Car {
	public Car() {
		System.out.println("Car::constructor");
	}
}
@Component
public class Chip {
	public Chip() {
		System.out.println("Chip :: Constructor");
	}
	public String chipType() {
```

```
            return "32-Bit";
        }
}


public class App {
        public static void main(String[] args) {
            ApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class);
        }
}
```

Autowiring with Annotation

=> To perform autowiring we will use @Autowired annotation

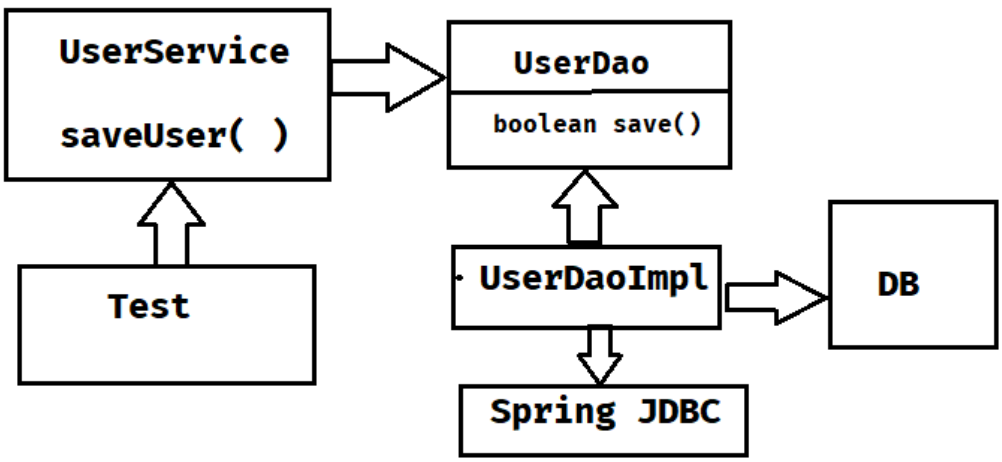
=> @Autowired annotation we can use at 3 places

1) variable level --- FI

2) constructor level -- CI

3) setter method level – SI

Assignment



Spring Bean Life Cycle

=> The class which is managed by IOC is called as Spring Bean.

=> We can perform some operations when bean object created and before bean object is removed.

- post construct
- pre destory

=> To achieve above requirement we can use bean life cycle methods

=> Bean Life cycle methods we can execute in 3 ways

1) Declarative approach (xml)

2) Programmatic approach

3) Annotation approach

Declarative Approach

```
public class UserDao {
	public void init() {
		System.out.println("getting db connection...");
	}
	public void getData() {
		System.out.println("getting data from db....");
	}
	public void destroy() {
		System.out.println("closing db connection...");
	}
}
```

```
<bean id="dao"
		class="in.ashokit.UserDao"
		init-method="init"
		destroy-method="destroy"
		/>
```

Programmatic Approach

=> We need to implement 2 interfaces here

1) InitializingBean ----> afterPropertiesSet ( )

2) DisposableBean ---> destory ( )

```
public class UserDao implements InitializingBean, DisposableBean{
	public void afterPropertiesSet() throws Exception {
		System.out.println("init method....");
	}
	public void destroy() throws Exception {
		System.out.println("destroy method....");
	}
	public void getData() {
		System.out.println("getting data from db....");
	}
}
```

Annotation Approach

=> We have below 2 annotations to achieve life cycle methods execution

1) @PostConstruct

2) @PreDestroy

@DependsOn: It is used to specify one class is indirectly dependent on another class.

```
@Component
public class UserDao {
	@PostConstruct
	public void init() throws Exception {
		System.out.println("init method....");
	}

	@PreDestroy
	public void destroy() throws Exception {
```

```
		System.out.println("destroy method....");
	}
	public void getData() {
		System.out.println("getting data from db....");
	}
}
@Component("userdao")
public class UserDao implements InitializingBean {
	@Override
	public void afterPropertiesSet() throws Exception {
		System.out.println("get data from db...");
		System.out.println("store data into redis...");
	}
}
@Service
@DependsOn("userdao")
public class UserService {
	public UserService() {
		System.out.println("getting data from redis...");
	}
}
```

=> UserService bean telling to IOC to create userdao object first.

Summary

1) What is IOC container ?

2) What is Dependency Injection ?

3) How to start IOC container ?

4) What is Bean Scope ?

5) Auto Wiring

6) what is primary attribute ?

7) what is ref attribute ?

8) How to configure Java class as Spring Bean ?

9) What is Spring Bean ?

10) How to get bean object from IOC ?

11) Spring Annotations

12) Component Scanning

13) @Autowired annotation

14) Bean Life Cycle

15) What is @DependsOn ?

Project Lombok

-> Project Lombok is a third party library

-> It is used to avoid boiler-plate-code in project

-> Project Lombok will generate below things for our classes

1) setter methods

2) getter methods

3) 0-param constructor

4) param-constructor

5) toString ( ) method

6) equals ( ) method

7) hashCode ( )

Project Lombok Setup

Step-1) Add Lombok Dependency in pom.xml file

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.26</version>
</dependency>
```

Step-2) Install Lombok jar in our IDE (Eclipse / STS / IntelliJ)

-> Goto lombok jar location

-> execute below command

```
$ java -jar <lombok-jar-file-name>
```

-> Specify IDE location (eclipse.exe / STS.exe)

-> Click on install

-> Close installer

-> Re-Start IDE

Note: Step-2 is required only first time.

Project Lombok Annotations

=> Project Lombok provided annotations to generate boiler plate code.

1) @Setter : To generate setter methods for variables

2) @Getter : To generate getter methods for variables

3) @ToString : To generate toString ( ) method

4) @NoArgsConstructor : To generate 0-param constructor

5) @AllArgsConstructor : To generate param-constructor

6) @EqualsAndHashCode : To generate equals () & hashCode ( ) methods

7) @Data =

@Setter + @Getter + @NoArgsConstructor + @ToString + @EqualsAndHashCode