**Spring**

-Framework to build java application

-Lightweight

-Provide lot helper class to make development easier

**J2EE**

Client side presentation-server side presentation—server side business logic -- Database

**EJB**

Complex ,multiple deployment descriptor

multiple interface , poor performance of Entity

Spring Framework

Goals

Lightweight development with java POJOs

Dependency injection to promote loose coupling

Declarative Program with AOP

minimized boilerplate java code

**Architecture**

Core Container

Beans,core,SPEL,Context

**It is heart**

**Object creation and management and holds the object in core container memory**

**Object factory for creating beans and Manage bean Dependency**

**Infrastructure**

AOP,Aspects,,Instrumentation,Messaging

Logging , trnasaction ,security Aop add fuctionilty to object declartivley etc..,

**Data Access Layer**

JDBC,ORM,TRANSACTION,OXM,JMS

JDBC helper classs reduce 50% jdbc code

Orm -object to realation mapping interagtion with hibernate and JPA

JMS ajava async messaeg sending broker

**Web Layer**

Servelt , websocket, web ,portlet

Web related class

Home for spring MVC

**Test Layer**

Unit,Integration,Mock

Out of container

**Inverion Of Control**

**Approach of outsourcing construction and management of objects**

Outsource to an object factory

**Spring provide Object factory -**

**Spring Container**

Primary function

**1 Create and Manage object(IOC)**

**2 Inject Object Dependencies(DI)**

**Configuring Spring Container**

1 XML Congif File (Leagcy)

ClassPathXmlApplicationContext cxml=**new** ClassPathXmlApplicationContext("applicationContext.xml");

2 java Annotaions

2 java Source Code

**Spring Development Process**

1 Config Spring Bean

<bean id=*"myApp"* class=*"com.basepgrm.ComputerEngieer"*>

</bean>

2 Create Spring Contaier

Generally know as ApplicationContext

Specalized Impleation : ClassPathXmlApplicationContext ,AnnotaionConfigApplicationContext etc.,

ClassPathXmlApplicationContext cxml=**new** ClassPathXmlApplicationContext("applicationContext.xml");

3 Retrive Bean from Spring container

**Task 1**

**My App ----BaseBallCoach(get Task)**

**App Schould be configurable**

**Easily change coach of diff sport**

"**Spring Bean**" is simply a Java object.

When Java objects are created by the Spring Container, then Spring refers to them as "Spring Beans".

Spring Beans are created from normal Java classes .... just like Java objects

In Spring 5.1, the Spring Development team changed the logging levels internally.

As a result, by default you will no longer see the red logging messages at the INFO level.

LOGG Creation

**Overview of the steps**

1. Create a bean to configure the parent logger and console handler

This class will set the parent logger level for the application context. It will also set the logging level for console handler. It sets the logger level to FINE. For more detailed logging info, you can set the logging level to level to FINEST

2. Configure the bean in the Spring XML config file

**Spring Dependency Injection**

Dependency inversion principle

The client delegates to calls to the another Object the responsibility of providing its dependencies

Dependency some of this helping objects

**Ex to understand**

**Car Factory**

When we require for new Car its not build initially

Parts by parts it is built in factory then we will get (so they will inject the objects)

Dependency=helper

**Task 2**

**Coach already provides daily workouts**

**Now will also provide daily fortunes**

New helper : FortuneService -> this is dependency

Coach dependes on fortune

**Injection Types**

-Constructor Injection

-Setter Injection

**Developemnt process -Constructor**

1 Define the dependency Interface and class

2 create Constructor in your class for injections

3 config dependency in spring config file

**Spring is responsible for creating object also injecting dependencies**

**Setter Injection**

1 Create Setter Methdos in ypur class for injection

2 config dependency in spring config file

**Injecting Literal Values**

1 Create Setter Methdos in your class for injection

2 config dependency in spring config file

**Injecting Literal Values using properties files**

1 creating property file

2 loading property file in spring config file

3 reference values from property files

**Spring Bean Scope and Life Cycle**

Scope refer to life cycle of bean

How long does been live

How many instance created

How is bean shared

**By Deafault Spring Bean Scope is Singleton**

**Singleton**

Spring Container creates only one instanceof the bean by default

It will be cached in memory

All req for bean will return a shared ref to the same bean

**Spring Scopes**

singleton--Create single shared instance of a bean , Default scope

prototype --Create a new bean instance for each container req

request -- scoped http web request

session-- scoped http web session

global-session -- scoped global http web session

**Bean Life cycle**

Container starts

1 Bean Instantiated

2 Dependencies Injections

3 Internal Spring processing

4 custom inti method --now bean ready to use

5 Container/ Application shut down Custom destoy method

**Bean Life cycle Methods/ Hooks**

**A Custom code during bean intilaization**

1 calling custom bussiness logic

2 setting handle resuource like (db , socket , file etc)

**B Custom code during bean destruction**

1 calling custom bussiness logic

2 close up handle resuource like (db , socket , file etc)

**Developemnt Process**

1 Define your mehtods for init and destroy

2 config methos names in spring config file

**For "prototype" scoped beans, Spring does not call the destroy method.  Gasp!**

In contrast to the other scopes, Spring does not manage the complete lifecycle of a prototype bean: the container instantiates, configures, and otherwise assembles a prototype object, and hands it to the client, with no further record of that prototype instance.

Thus, although initialization lifecycle callback methods are called on all objects regardless of scope, in the case of prototypes, configured destruction lifecycle callbacks are not called. The client code must clean up prototype-scoped objects and release expensive resources that the prototype bean(s) are holding.

**Configuring Spring with Java annotations**

**Java /annotaions**

1 special labels/ markers added to java class

2 provide meta -data about class

3 process at compile / run time for special processing

**Why Spring config with annotations**

1 xml config can be verbose

2 config spring beans with annotations

3 annotations minimize xml config

**Scanning for Component class**

Spring will scan java class for special annotations

Automatically register the beans in the spring container

**Development Process**

1 Enable component scanning in spring config file

<context:component-scan base-package=*"com.annotaionbased.sample"*/>

2 Add @Component to your class

@Component

**public** **class** TennisCoach

3 Retrive bean fro srping Container

Spring **Component** annotation is used to denote a class as **Component**. It means that Spring framework will autodetect these classes for dependency injection when annotation-based configuration and classpath scanning is used.

**@componet and default bean id**

Spring also support default bean id

Default bean id : the class name, make the first ltr lower case

Classname = TennisCoach

Default Bean Name== tennisCoach

**Spring Dependency injection with Annotaions and autowiring**

**Spring AutoWiring**

1 For dependecy injection spring use autowiring

2 spring will look for a class that matches the property

Matches by type :class / interface

3 Spring will inject automatilcally hence it is autowired

**Autowiring InjectionType**

1 Constructor

2 Setter

3 field

**Deveploemrnt process of Constructor Injection**

1 Define the dependency interface and class

2 create a constructor in your class for injections

3 Config dependency injection with @Autowired annotation

**Deveploemrnt process of setter Injection**

1 create Seter method in ur class for injection

2 Config dependency injection with @Autowired annotation

We can injectect depency by calling any method of our class

Instead of setter we can use any methods

**Field Injection**

Inject depencies by setting our field values on your class directly

(even private fields)

**Accomplishes by using java reflections**

**Deveploemrnt process of Field Injection**

1 Configure dependency injection with autowired annotations

Apply directly to feild

No need of setter method

**Annotations Autowiring and Qualifiers**

When we have multiple implementaions of dependency Inteface it will give conflict which to take then Qulaifer used to tell the particaulr

**@Qualifiers**

**Applied to Constructor,setter,feild injections**

*As of Spring Framework 4.3, an @Autowired annotation on such a constructor is no longer necessary if the target bean only defines one constructor to begin with. However, if several constructors are available, at least one must be annotated to teach the container which one to use.*

**@Qualifier** is a nice feature, but it is tricky when used with Constructors.

The syntax is much different from other examples and not exactly intuitive.  Consider this the "deep end of the pool" when it comes toSpring configuration LOL :-)

**You have to place the @Qualifier annotation inside of the constructor arguments.**

**@PostConstruct ? @PostDestory?**

**Bean Scopes with Annotaions**

Default scope is Singleton

@Scope(“Singleton”) with class name

Bean Life Cycle Method Annotaions

Same as been but inti and Detroy methos are differnt based on Annotaions

Developemnt Procees

1 define your methos for inti and destroy

2 add annotaios : @PostConstruct ,@PreDestroy

**Spriung Configuration with Java Sourrce Code**

**Java Configuration**

Instead of config Spring Container in XML config spring container with java code

NO XML

**3 ways config Spring container**

1 XML Config Fully

2 XML Commponet Scan with use of Annotations

3 Java Config Class

**Deveoplment Process of Java Based Spring Container Config**

1 Create a Java Class and Annotae I as @Configuration

@Configuration

@ComponentScan("com.annotaionbased.sample") @PropertySource("classpath:external.properties")

**public** **class** JavaConfigBased

2 Add Componet Scan (optional)

3 reading spring java congif class

AnnotationConfigApplicationContext context=**new** AnnotationConfigApplicationContext(JavaConfigBased.**class**);

4 retrive bean from spring conainer

**Java Bean Configuration with no annotaions**

**Develpoment Process**

1 Define methos to expose bean

2 inject bean dependnecy

3 reading spring java congif class

4 retrive bean from spring conainer

**Inject vaules using Proerpty files**

**Deveolpme Process**

1 create property file

2 load in spring config

@PropertySource("classpath:external.properties")

3 refernec property file

@Value("${country.player}")

**private** String Country;

**Creating Logger File**

**Overview of the steps**

0. Create a logging properties file

1. Create a configuration class to configure the parent logger and console handler

2. Reference the configuration class in the main app