Spring
popular framework for building java applications
simpler and lightweight alt of J2EE

J2EE : Multi - tier
CLIENT LAYER ◄ ► SERVER SIDE ◄ - - ► SERVERBUSINESS LOGIC ◄ - - ► DATABASE

EJB Framework : Container Module EJB V1 and V2 (complex)
J2EE1.2/1.3/1.4

#Multiple Deployment desc
#Multiple Interface

Bean Class

Component Interface

Home Interface

EJB Client

2003 : Rod Johnson
Project using J2EE without EJB
he introduced his own Bean Container (POJO)
2004 Spring 1.0

J2EE
1999 (J2EE 1.2)

2006 (EJB 3.0) : Java EE 5
2009 Java EE 6
2013 Java EE 7
2017 java EE 8

Spring Sep 2017 (V 5);

Spring

Lightweight development java POJOs

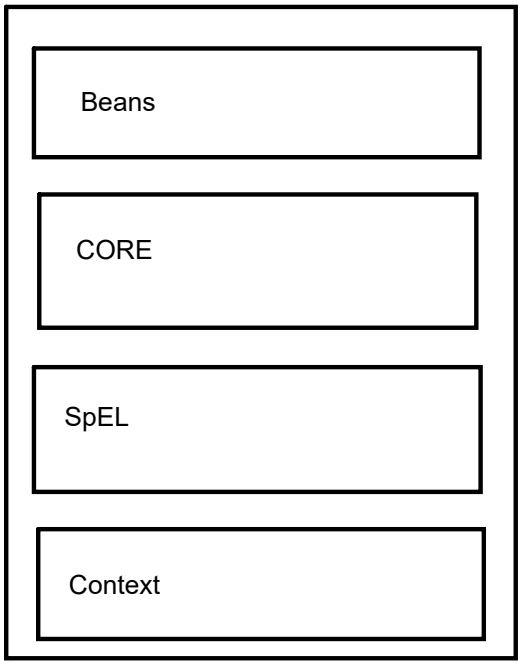
Dependency Injection for loose coupling

Declarative Programming using AOP(Aspect Oriented Programming)

#Minimize the java boilerplate code

Different layers (classification of SPring resources)

Layer of Containers



Responsible for low level activities for there component

hold the components

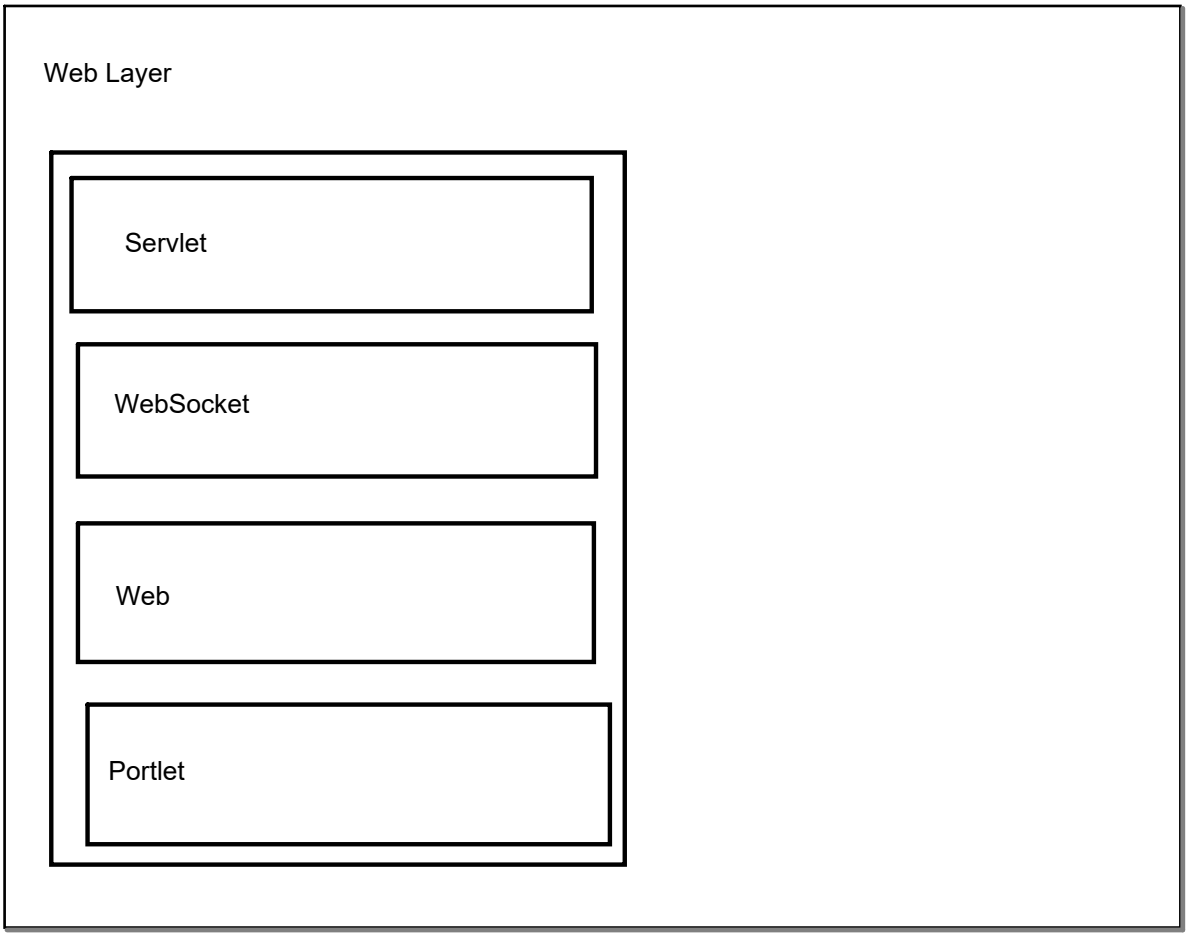
#initiating/destroying

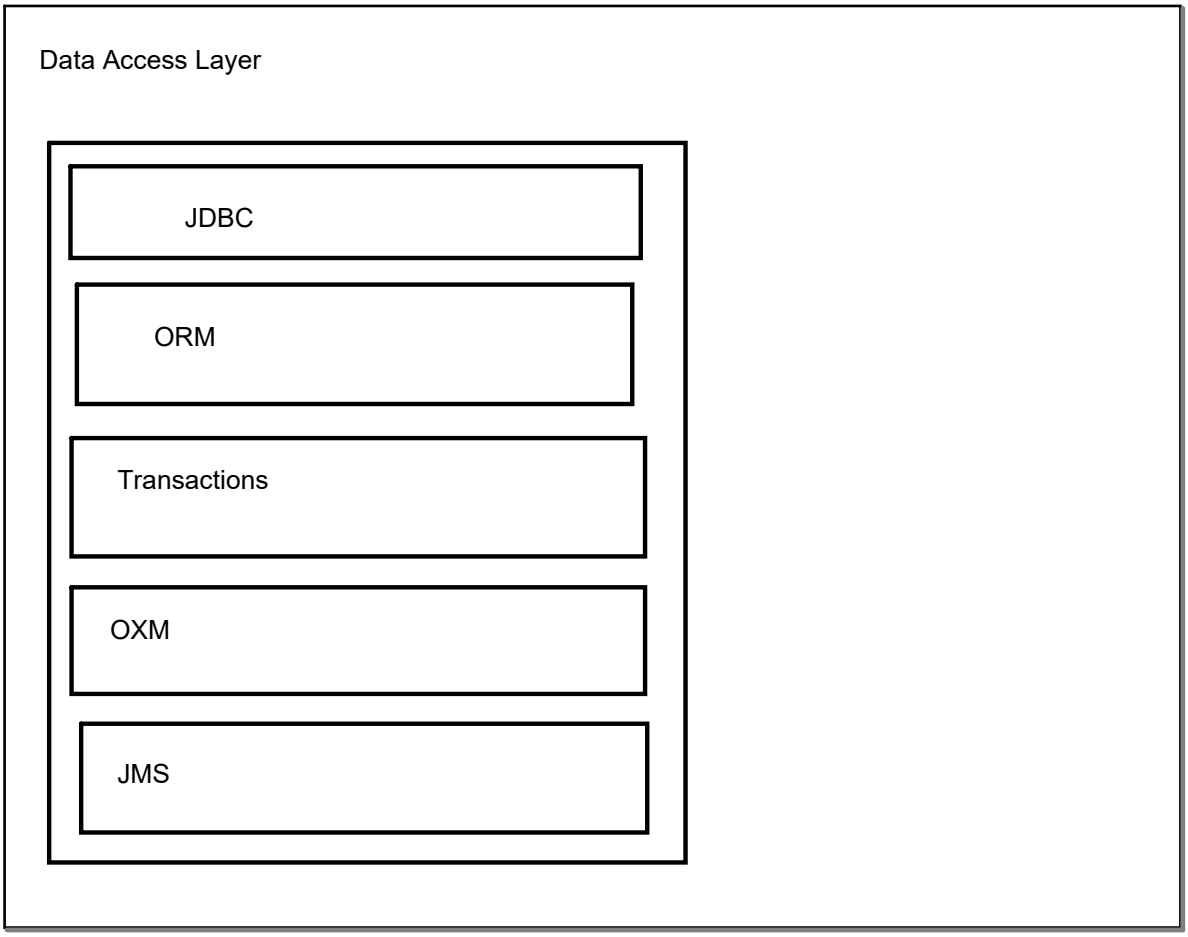
#memory management

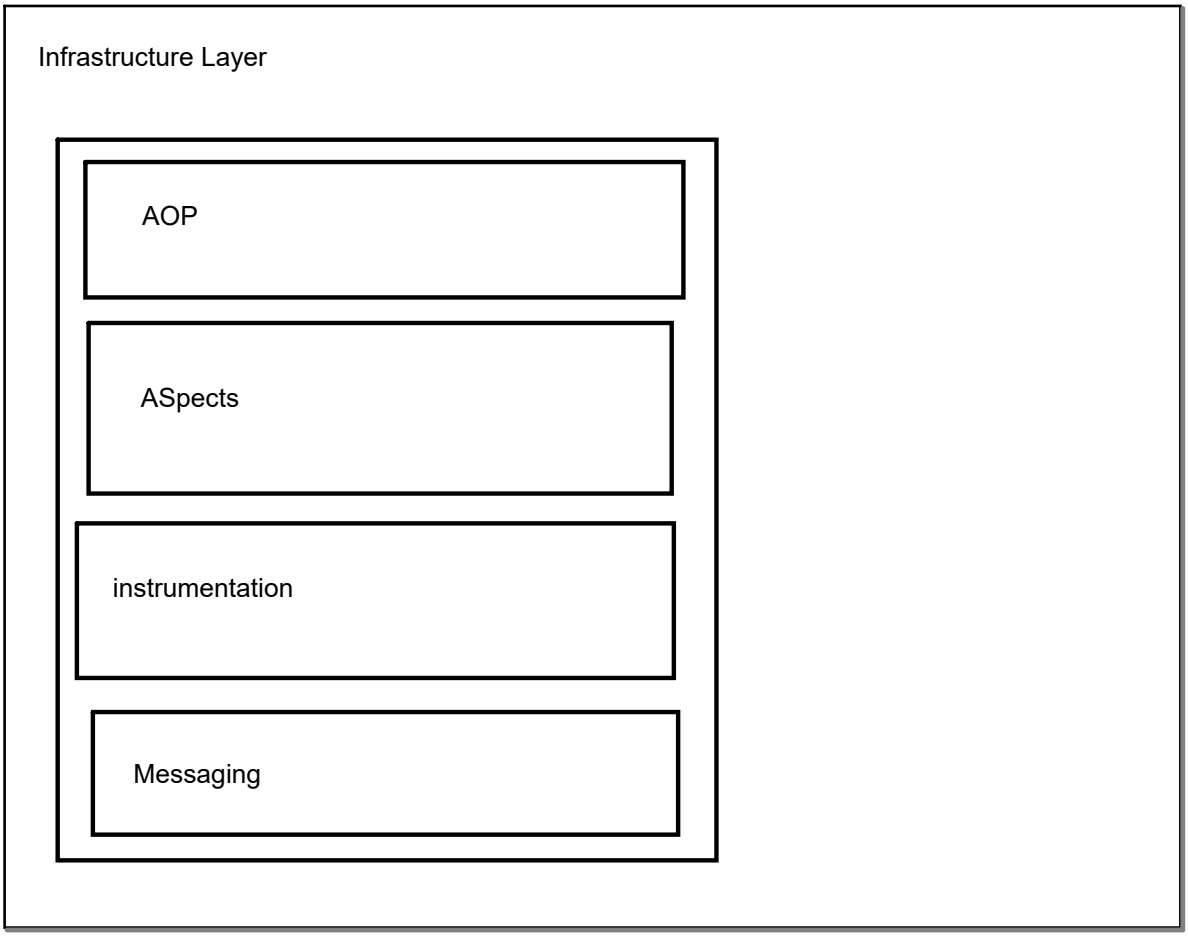
#cache

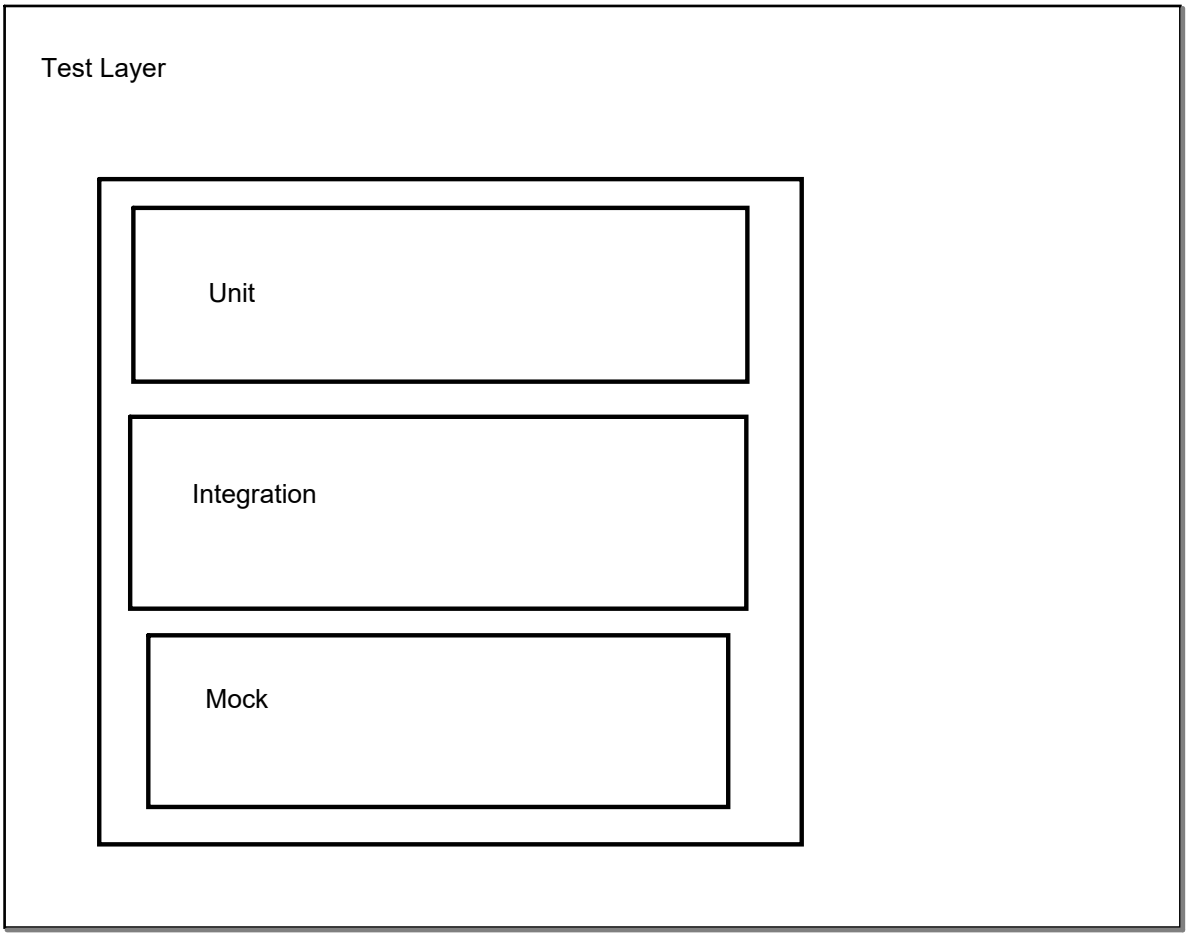
#threads

#object managements


Web Layer
Servlet
WebSocket
Web
Portlet


Data Access Layer
JDBC
ORM
Transactions
OXM
JMS


Infrastructure Layer
AOP
ASpects
instrumentation
Messaging


Test Layer
Unit
Integration
Mock

Spring Framework 5

#min req java 8 higher

#deprecated legacy integration for : Tiles,Velocity,Portlet,Guava

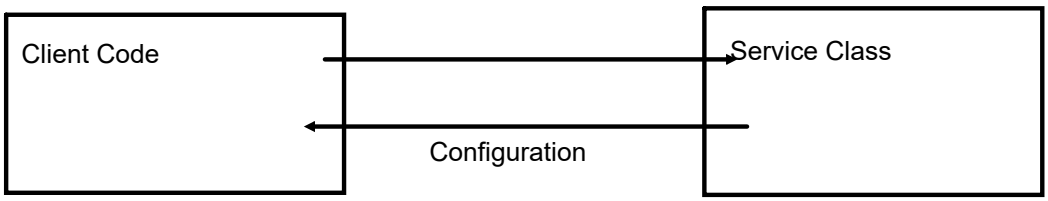
#Spring MVC : new Servlet API4.0

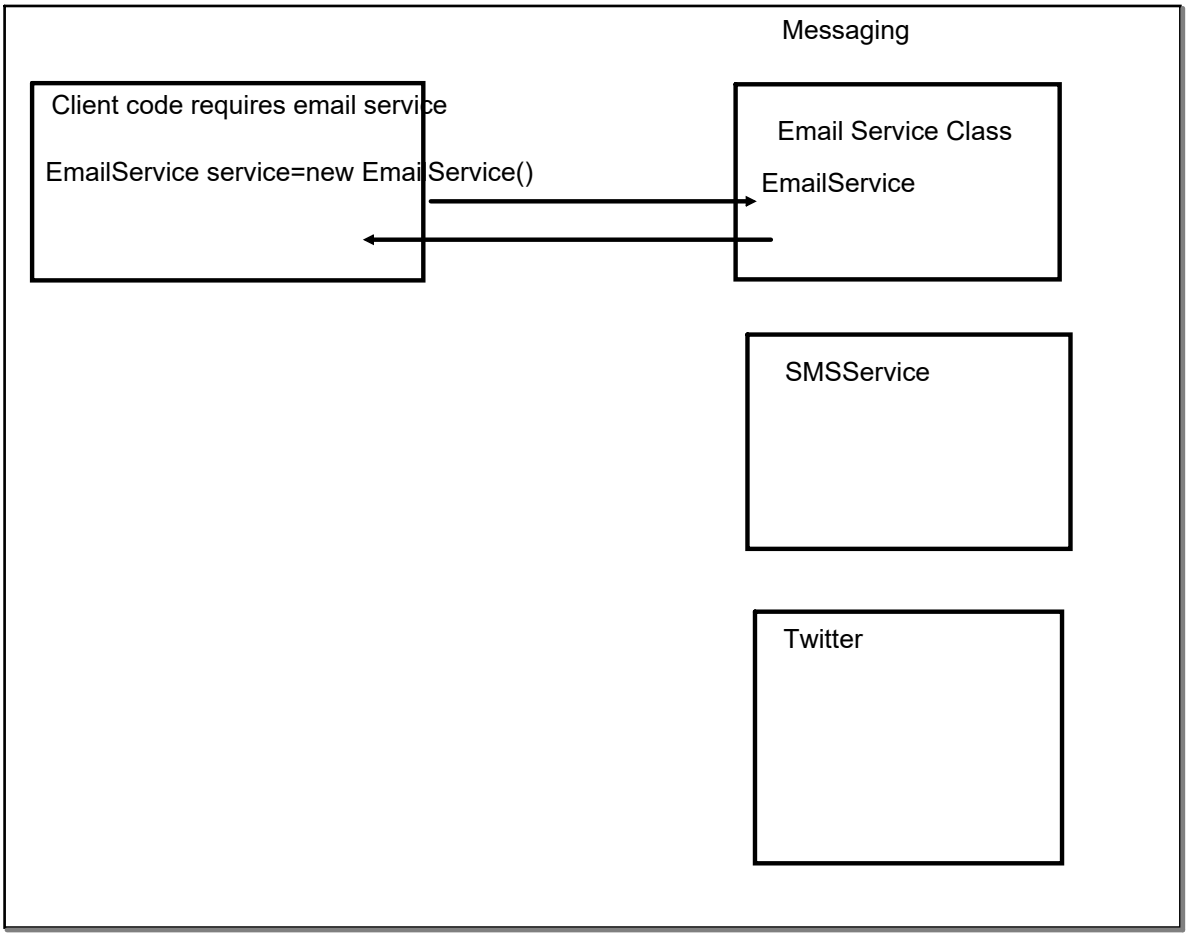
#new reactive Programming framework : WebFlux

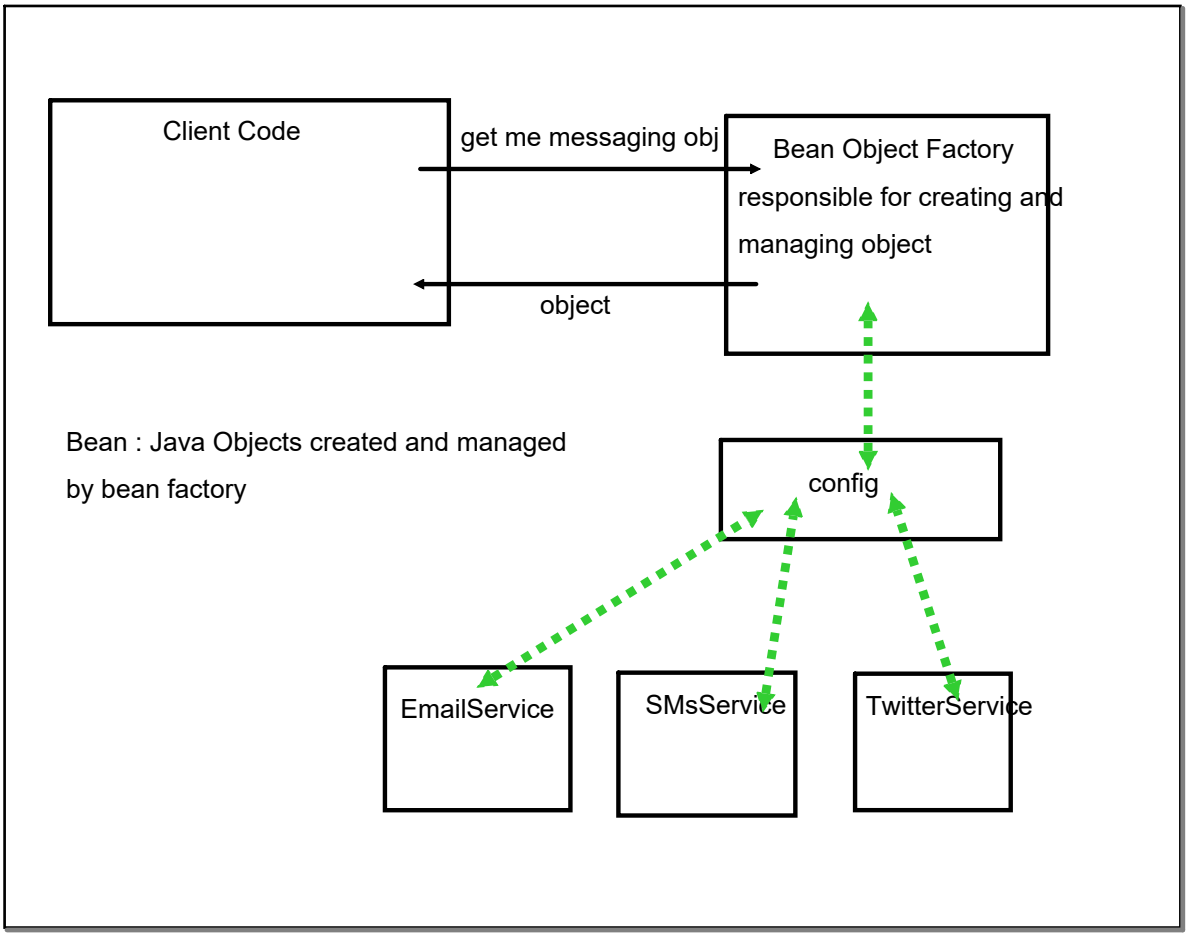
Inversion Of Control (IoC)

The approach of outsourcing the construction and management of objects

EJB: Bean Container




Messaging
Client code requires email service
EmailService service=new EmailService()
Email Service Class
EmailService
SMSService
Twitter


Client Code
get me messaging obj
Bean Object Factory
responsible for creating and
managing object
object
Bean : Java Objects created and managed
by bean factory
config
EmailService
SMsService
TwitterService

Spring Container/Bean Factory/Bean Container

#Primary functionality

=> Inversion of control

=> Dependency Injection

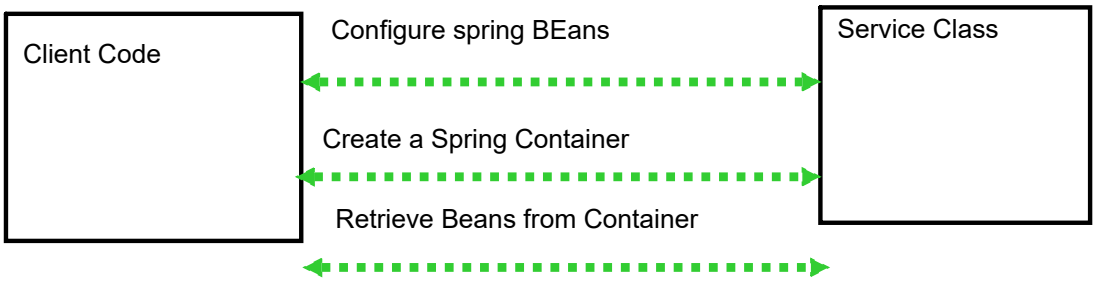
Configuring the Spring Container

=>XML config file(legacy)

=>Java Annotation (modern)

=>Pure Java (modern)

DEvelopment Process



package naming : cognizant.com

com.cognizant.

Spring Container is generically known as ApplicationContext

Specialized Implementation

=>ClassPathXmlApplicationContext

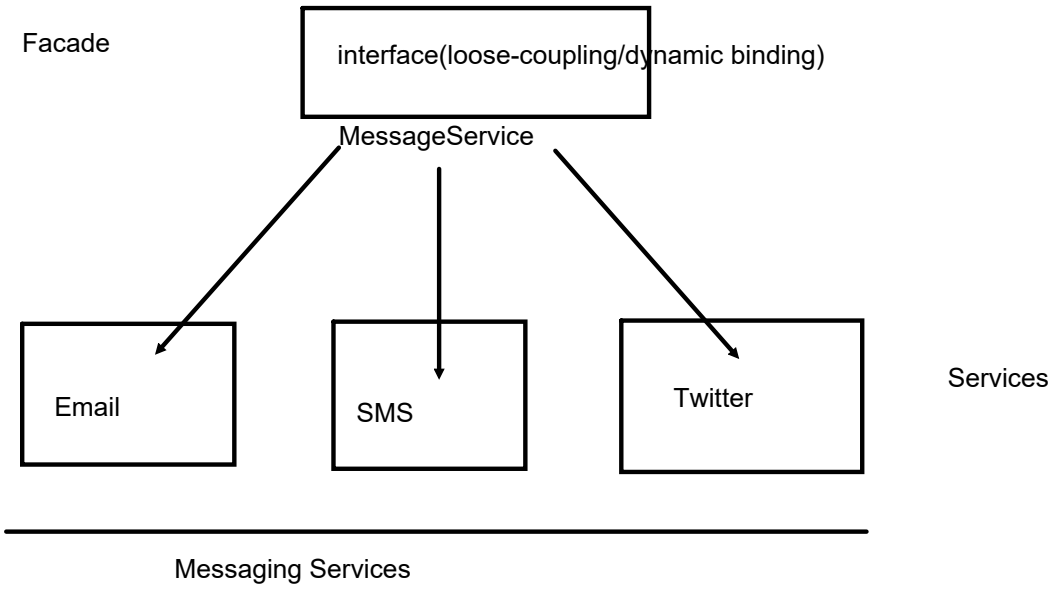
=>AnnotationConfigApplictionContext

=>GenericWebApplicationContext

```
<bean id="msgService" class="com.training.ioc.services.EmailService"/>
```

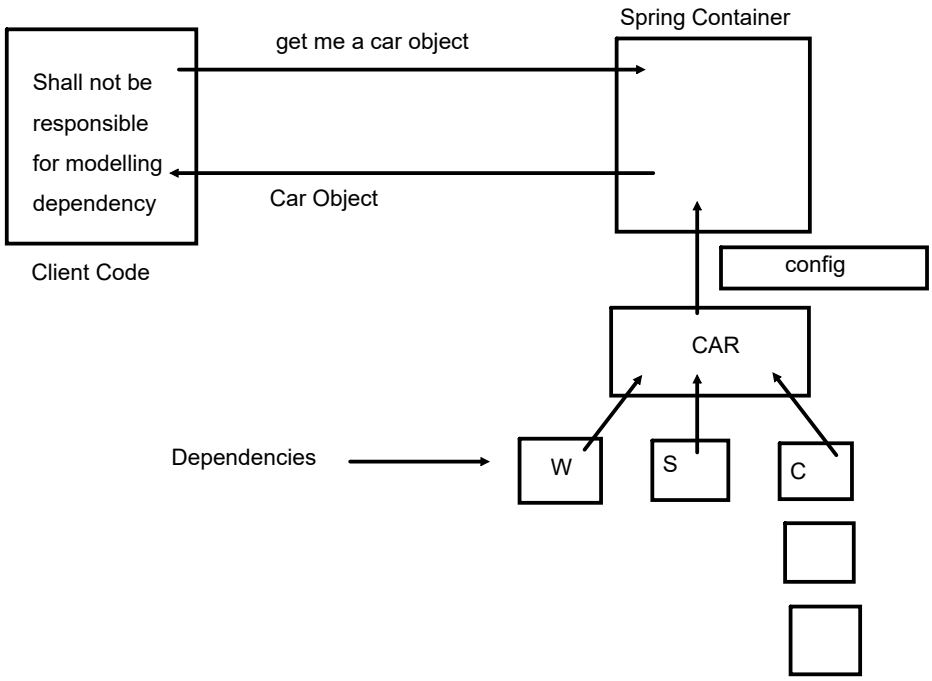
bean name

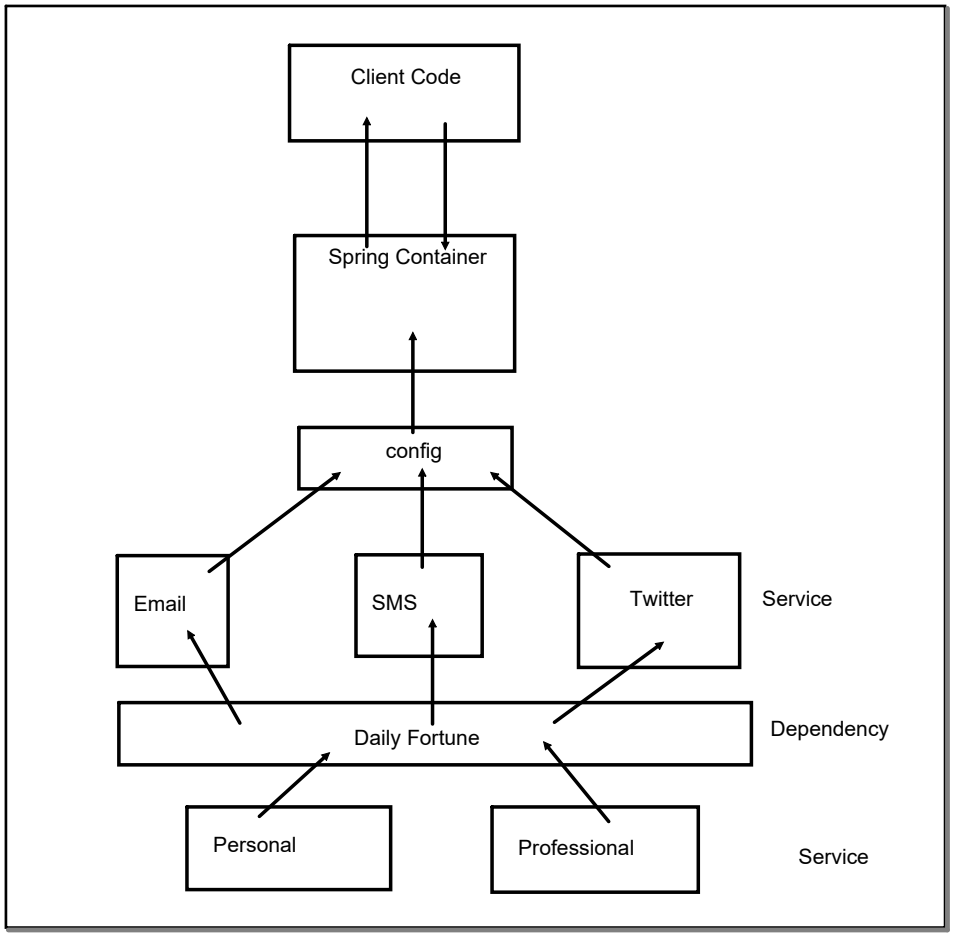
Java coding style : facade coding style (Totally decouple the client code from services)



Dependency Injection:

Delegating calls to another object, the reponsibility of providing its dependency




Client Code
Spring Container
config
Email
SMS
Twitter
Service
Daily Fortune
Dependency
Personal
Professional
Service

Dependency Injection (DI) in XML based config

#Constructor Injection

#Setter Injection

Constructor Injection

==>create dependency resources

==>create a appropriate constructor in message classes

==>config the DI in Spring config file

```
<bean id="fortune" class="com.training.ioc.helpers.PersonalFortune"/>

    <!-- Initiate Message Service -->
    <bean id="msgService" class="com.training.ioc.services.SmsService">
        <constructor-arg ref="fortune"/>
    </bean>
```

Setter based Injection

#instead of constructor, create appropriate setter method

#config the DI in spring config file

```
<bean id="fortuneService" class="com.training.ioc.helpers.ProfessionalFortune"/>
```

```
<bean id="msgService" class="com.training.ioc.services.EmailService">
    <!-- Inject dependency -->
    <property name="fortune" ref="fortuneService"/>
</bean>
```

Name of field whose setter method is created

Inject a literal value (setter based dependency)

Literal values can be outsourced to other files (property file)

#Create a property file (key-value pair)

#refer / load property file in spring config file

read values from property file

```
<context:property-placeholder location="classpath:message.properties"/>
```

root of src folder

Spring Container:

Initiate/Construct the bean(container managed POJOs)

Manage the bean

Scope of the beans

#How long does the bean live?

#How many instances are created?

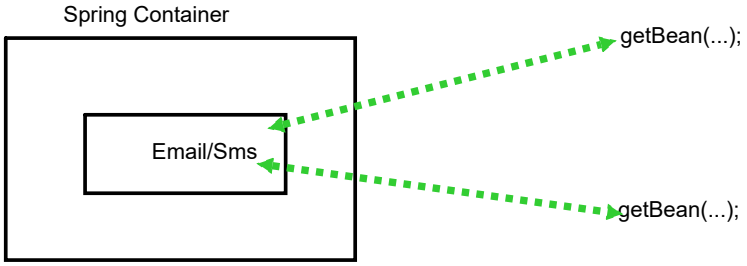
#How a bean is shared?

Default scope of bean : Singleton

#Spring Container creates only one instance of the bean

#Cached in memory

#All request (any client) for the bean : will return a SHARED ref to the same bean



When does bean object intantiates:

1. When spring container is created
2. When a call to getBean() is done

Singleton(default)

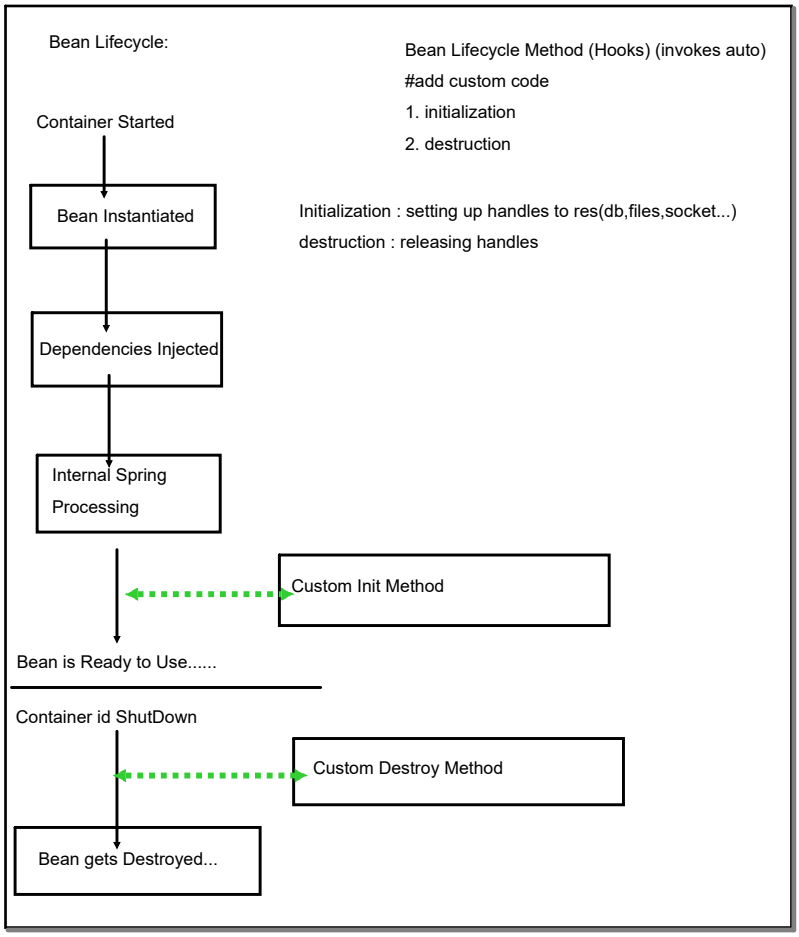
prototype : Creates a new bean instance for each container

Web Env:

request : Scoped to an HTTP web request

session : Scoped to an HTTP web Session

global-session : Scoped to a global web Session(Application Context)


Bean Lifecycle:
Bean Lifecycle Method (Hooks) (invokes auto)
#add custom code
1. initialization
2. destruction
Container Started
Bean Instantiated
Initialization : setting up handles to res(db,files,socket...)
destruction : releasing handles
Dependencies Injected
Internal Spring Processing
Custom Init Method
Bean is Ready to Use.....
Container id ShutDown
Custom Destroy Method
Bean gets Destroyed...

Custom Lifecycle hooks:

#Define the methods in bean class

#Configure method names in spring config file

Prototype of custom methods:

Access modifiers : can have any access modifier (private,public , protected,<default>)

Return type: can have any return type, can return values: will not be able to capture values

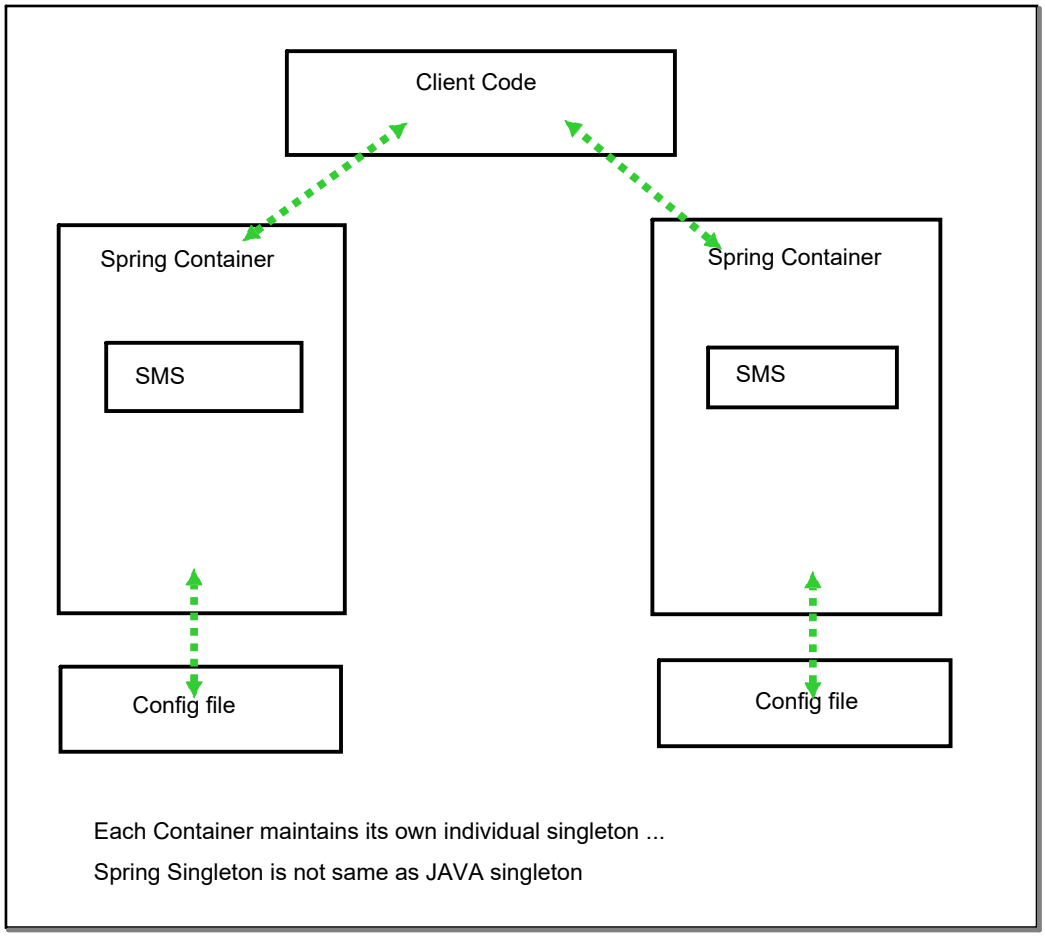
"void" is common

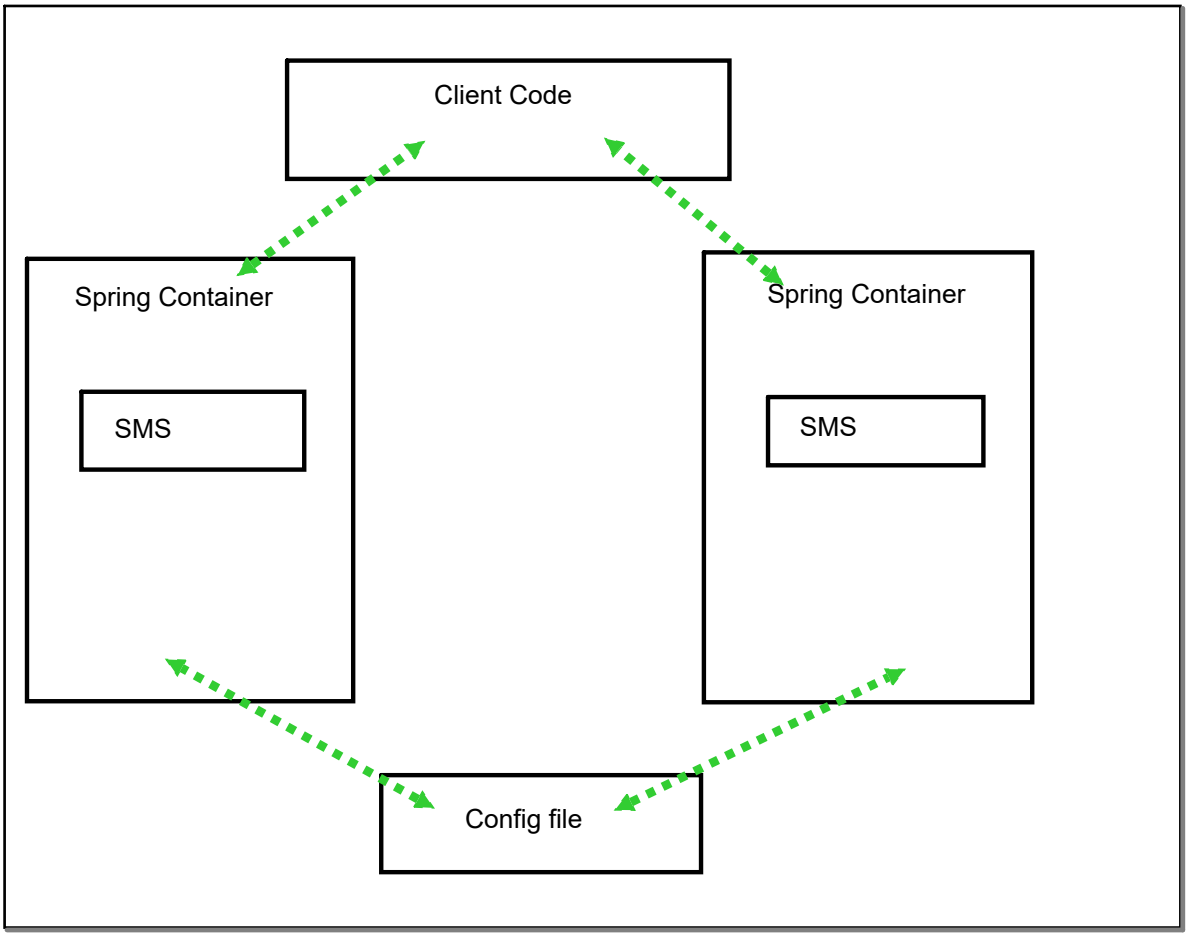
Method-name: can have any method name

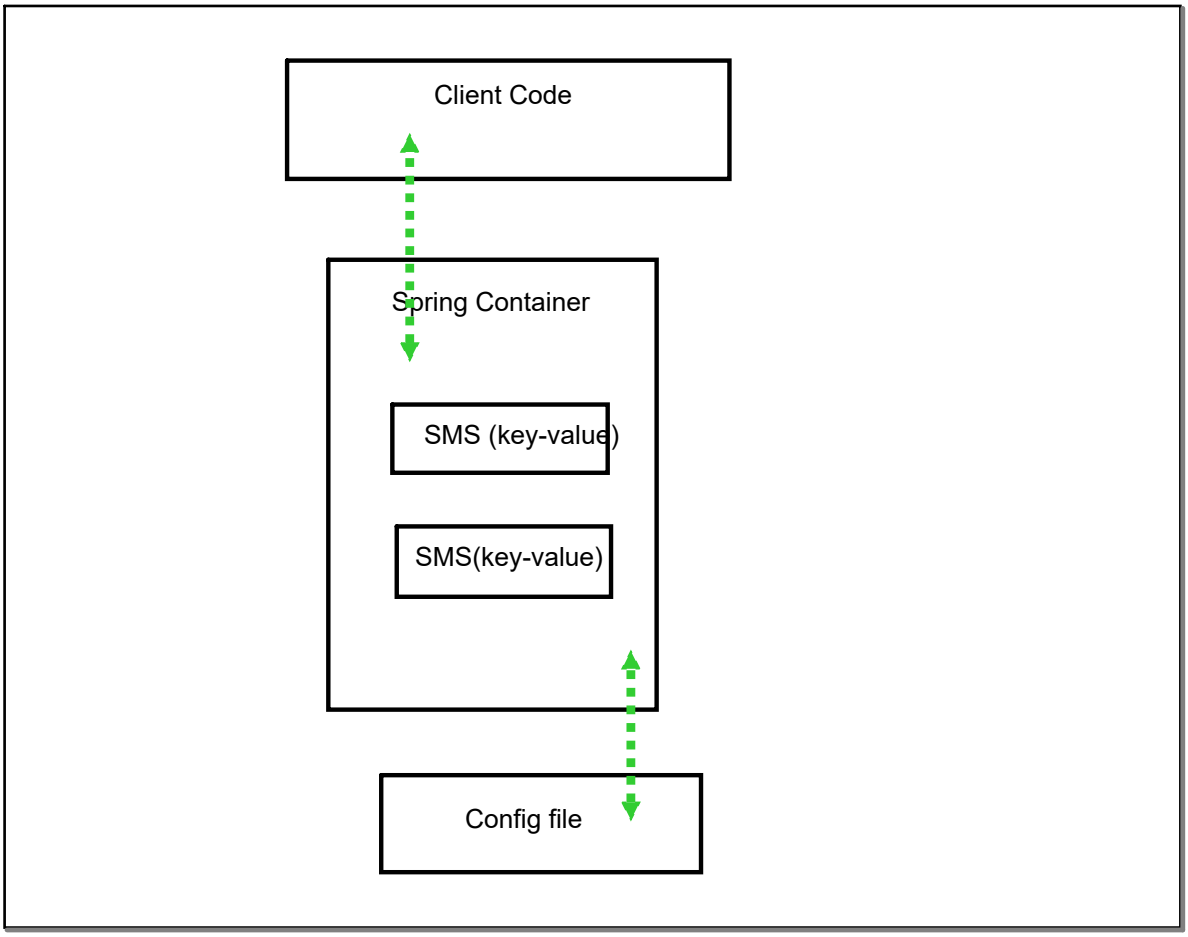
Arguments : cannot accept arg : no-arg method

==> Prototype scope : Spring does not manage the complete lifecycle:

container intantiates,config,assembles,internal processing and hands over bean to client code. Client code is responsible for any clean up operations


Client Code
Spring Container
SMS
Spring Container
SMS
Config file
Config file
Each Container maintains its own individual singleton ...
Spring Singleton is not same as JAVA singleton


Client Code
Spring Container
SMS
Spring Container
SMS
Config file


Client Code
Spring Container
SMS (key-value)
SMS(key-value)
Config file

Annotation :

#Special labels/markers added to java elements
#meta-data
#runtime/compilation time

XML config : verbose
Annotations minimizes XML config

Spring framework behavior:
=>will scan JAVA classes for special annotations
=>Auto register beans in Spring Container

Process

#Spring config file : enabling component scanning (specify path to scan for)

#Add @Component Annotation

Retrieve bean from Spring Container

#Spring also supports default bean Id :

class name : first letter lower case

EmailService : emailService

Spring Dependency Injection with Annotations

For DI spring uses AutoWiring

Spring will look for a class that matches the property

#matches by type : class or interface

Spring will inject it auto :

Autowiring Injection :

#Constructor Injection

Setter Injection

Field Injection

#have an appropriate constructor

#config the DI with @Autowired ann

#scan an intantiates the beans of classes decorated with @Component (all beans registered in Container)

#tracks all Autowired instance and injects the appropriate bean based on type

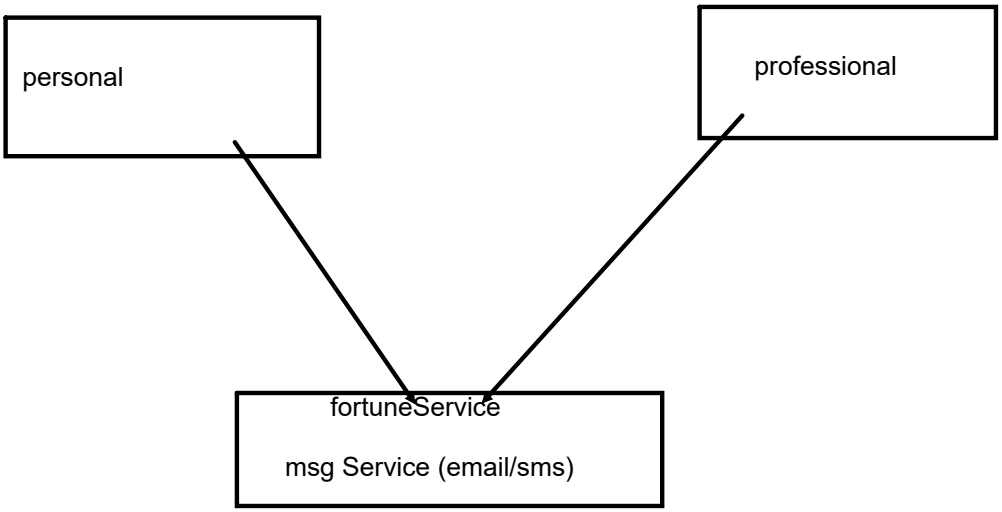
Setter based injection:

Create an appropriate setter method

Decorate with @Autowired

Field based injection

#Inject dependency by setting field values directly (even for private flds)



need to specify, which implementation of fortune service to inject...

@Qualifier(<bean id>)

Constructor injection : @Qualifier with parameter

using @Autowire in constructor is no longer necessary , as long as only one constructor is there...

#should be using @Autowired : auto doc, to resolve any future conflict

#Setter based injection...

==>no need to have standard naming convention for setter methods

==>can be any name....

#Injecting literal values...

#config spring config file for path of properties file

#add @Value over fields to inject literal values

Defining scope:

#@Scope

Lifecycle methods :

#define the methods

decorate with annotation

java based config:

#Java class and decorate with @Configuration

#to enable the component scanning : @ComponentScan

#use different implementation of Application context

==>exposing java beans using java code

#define method to expose bean

#java code to inject bean dependency

==>inject literal values

#Load prop in config class

#ref values from property file