Spring Framework

#lightweight and simpler

2004 : Rod Johnson

EJB ~ Bean Container : Spring

J2EE : EJB (V1/V2)

Complex

Multiple Deployment Descriptors

Multiple Interface

Poor performance on production

java 5 : Java EE 5

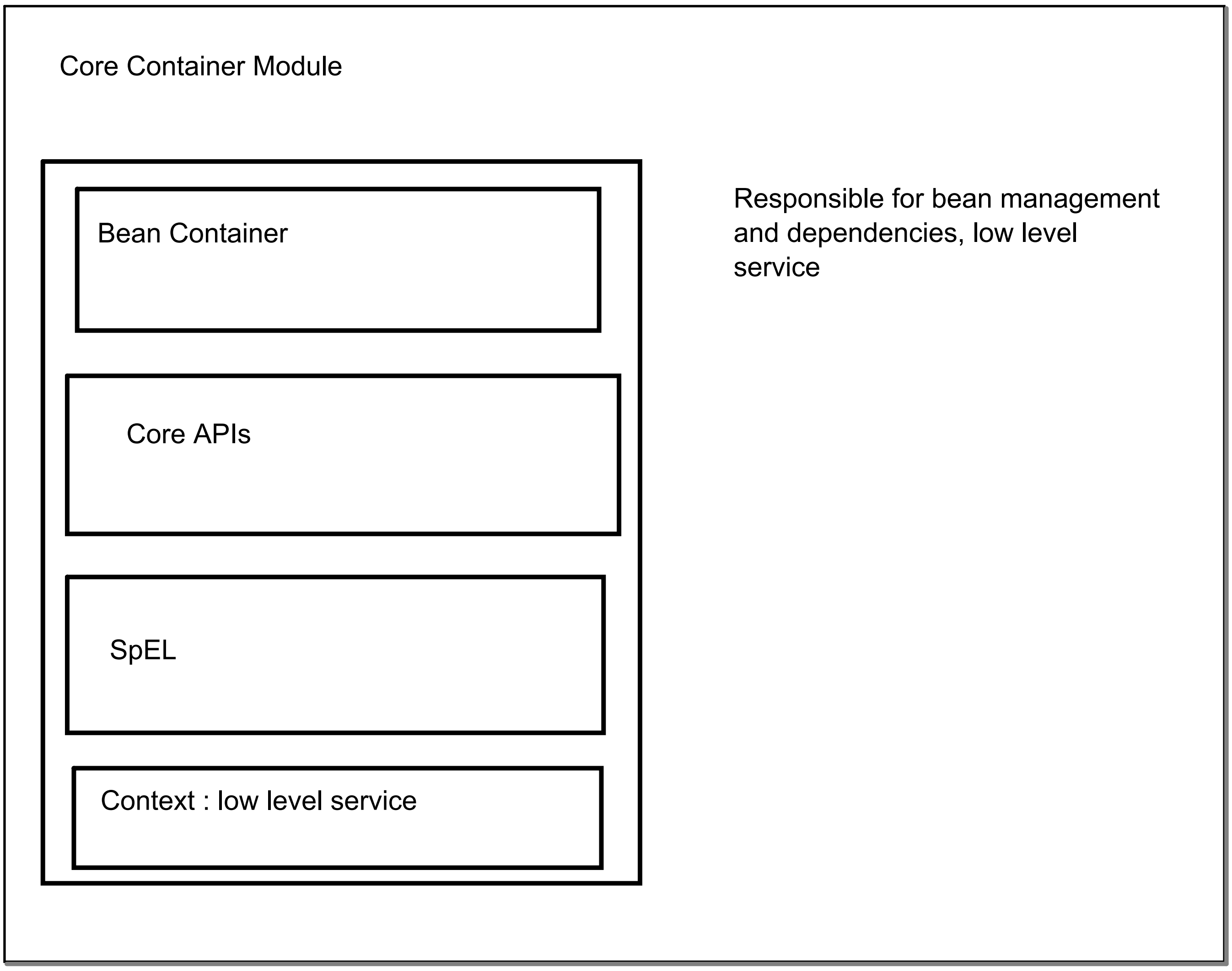
Modular development

Java POJO : min java boilerplate code

Dependency Injection to promote loose coupling

Declarative Programming : AOP

Spring Modular architecture


Core Container Module
Bean Container
Core APIs
SpEL
Context : low level service
Responsible for bean management and dependencies, low level service

Infrastructure Module

AOP APIs

ASPECT based inject

Instrumentation mappings

Messaging

Declarative : AOP

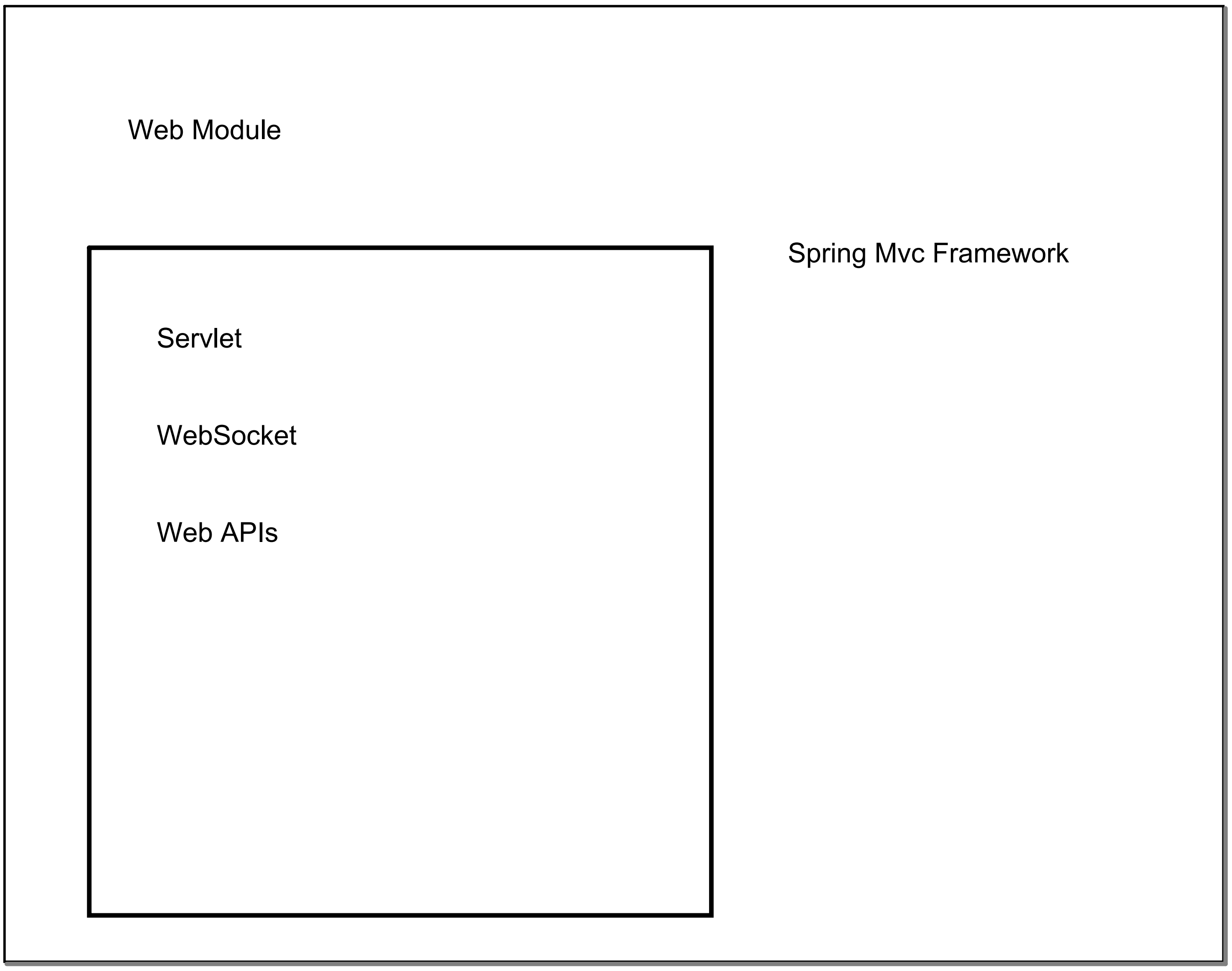
Data Access Module

JDBC : Spring way

ORM

Transaction

OXM/JMS


Web Module
Spring Mvc Framework
Servlet
WebSocket
Web APIs

Test Module

- Unit
- Integration
- Mock Objects

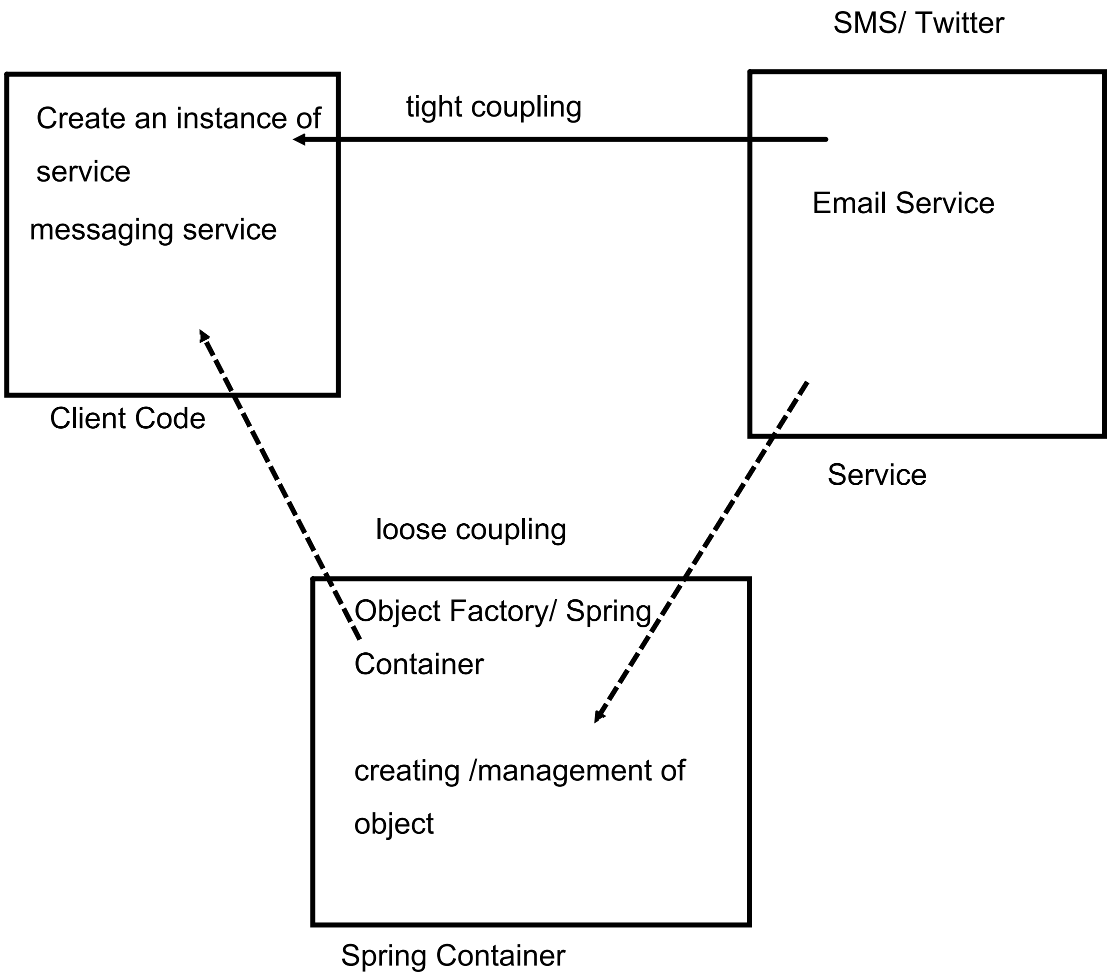
Support of Test-Driven-development (TDD)

Any additional resource set up required : Spring Projects
Additional Modules built on top of Core Framework
==>Spring cloud,Data,Batch,Security,Boot

Development/Management Tools (Maven, Gradle)
explicit framework integration

Inversion of Control (IoC)

#Approach of Outsourcing the creation and management of objects

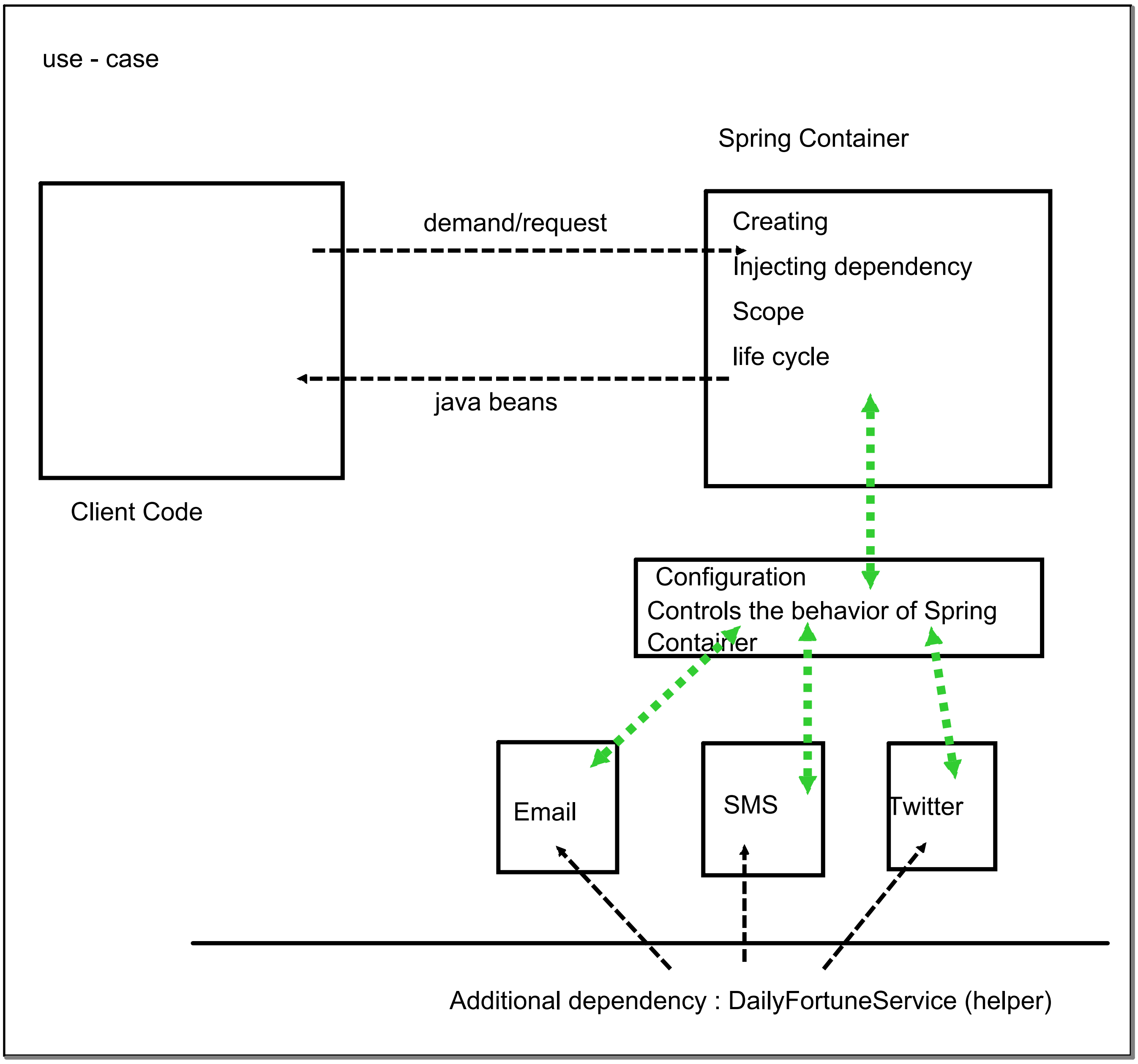


1.. Client code no more require to manage the low level activities related to object (DI)

2. loose coupling (changing the object scenario explicitly)

App must be configurable

Java beans : those POJO created and managed by spring container


use - case
Spring Container
demand/request
Creating
Injecting dependency
Scope
life cycle
java beans
Client Code
Configuration
Controls the behavior of Spring Container
Email
SMS
Twitter
Additional dependency : DailyFortuneService (helper)

Configuration

- # XML config (legacy)
- # Java Annotation
- # Java Code (no use of XML)

www.spring.io

Java Design pattern (Interface/Facade design) : loose coupling

Process:

1. Add Configuration for Spring Container
2. Create a Spring Container based on that config
3. Retrieve the bean from container

XML Config:

1. core spring dependency available in XML file (special XML tags)

Client Code: need to create bean container

Spring Framework (base-interface: Spring Container) : ApplicationContext

#specialized implementation

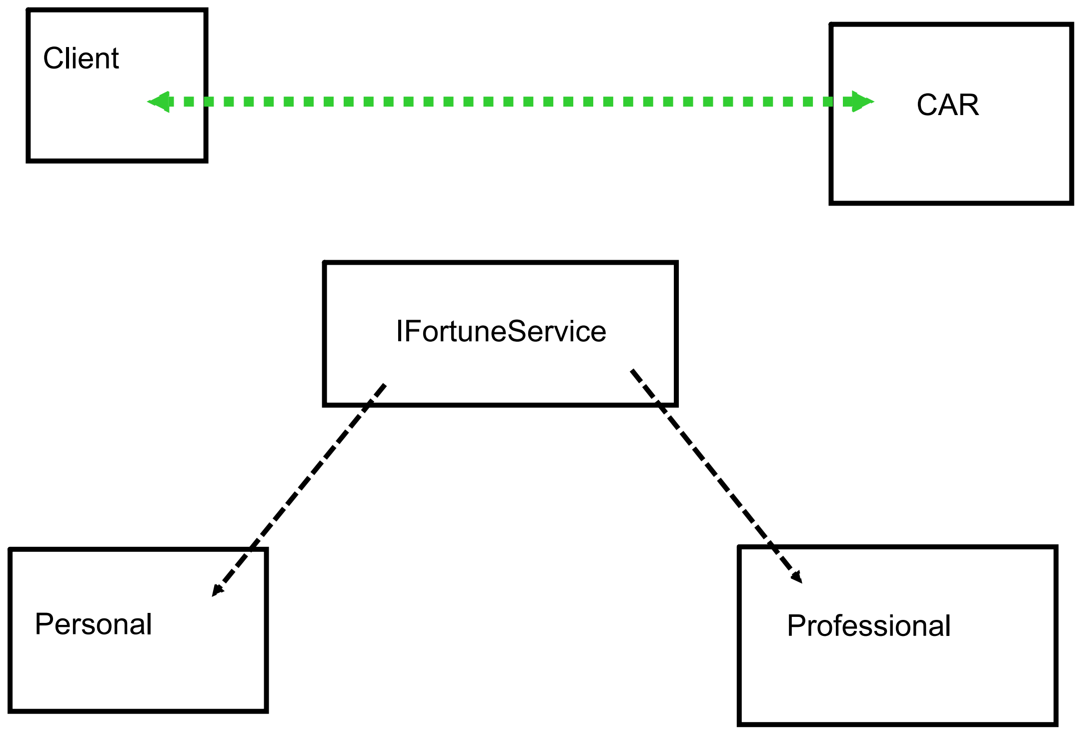
#ClassPathXmlApplicationContext (XML file)

#AnnotationConfigApplicationContext (Java impl)

#GenericWebApplicationContext....(web app)

Dependency Injection :

delegating the responsibility of providing dependency to config section



XML Config:

1. Constructor based
2. Setter based

Injecting values from properties file (repo)

Process:

Create a properties files (key-value)

Load properties file in Spring config

Refer the values from property file

Spring Container :

creating the bean (DI)

manage the beans

- #scope
- #life cycle

Scope :

singleton:

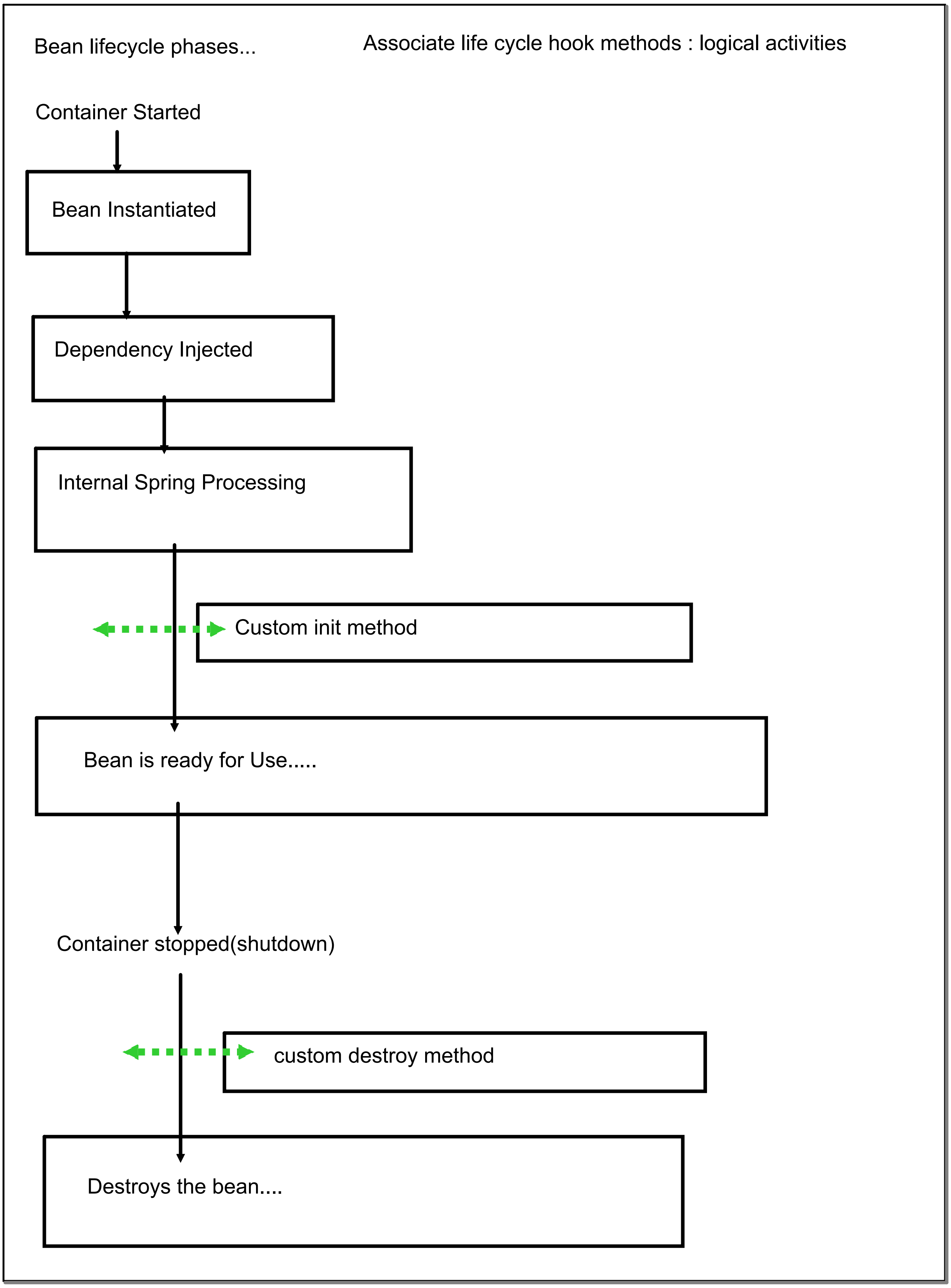
- single instance
- cached in memory
- return shared ref when demanded...

singleton: single instance

prototype : create a new instance on each request

web container

- request : single request - response cycle
- session : complete session (user specific)
- global-session : global http web - session


Bean lifecycle phases...
Associate life cycle hook methods : logical activities
Container Started
Bean Instantiated
Dependency Injected
Internal Spring Processing
Custom init method
Bean is ready for Use.....
Container stopped(shutdown)
custom destroy method
Destroys the bean....

Every Spring container has its own singleton env

#Spring singleton is not same as Java Singleton!!!

#Create the custom method

#add them in config file

Custom method:

1. access modifiers (any modifiers)
2. return type : generally void (can have any return)
3. name : any name
4. argument: cannot accept any arg

###Spring does not manage the complete lifecycle of prototype scoped bean

Annotation based IoC/DI

#special labels/markers

#meta-data

#can be processed runtime/compile time

Spring does:

- will scan Java Classes for special annotations
- auto register the bean in container

XML file : enable the component scanning

IoC :

Special Annotation : @Component

#default bean Id: class name (beg with small letters)

Annotation DI

Auto-Wiring

Spring will look for class that matches the property you need (type : class name/ interface)

Autowiring :3 types

1. Constructor
2. Setter
3. Field based

#by default if only instance of dependency is there... (no need to use bean id)

#multiple instances of same type : need to use bean id

#Constructor (if there is only single) : no need to use @Autowired

#literal values : add path of properties file in xml

Scope : life cycle method (annotations)

Pure Java Config:

1. Replace xml file with a java class
2. let the spring know to register class as config class (@Configuration : inherited from @Component)
3. @ComponentScan : to specify the component scanning path
4. @PropertySource : to specify the properties
5. Client code need to tell spring to fetch config detail from which java class