Spring Framework Servlet-API

MVC Architecture : Manual

Architecture is implemented strictly, disciplined way

remove lot of Boiler-plate code

abstract the low level complexity

Focus more on business logic

Most popular frameworks to develop java application

J2EE : Java 2 Enterprise Edition : Framework to develop web app using java

Complex in nature

lots of deployment descriptor

lots of interface, abstract classes needs to be created to expose a single service

productivity reduces, reduces efficiency

Rod Johnson

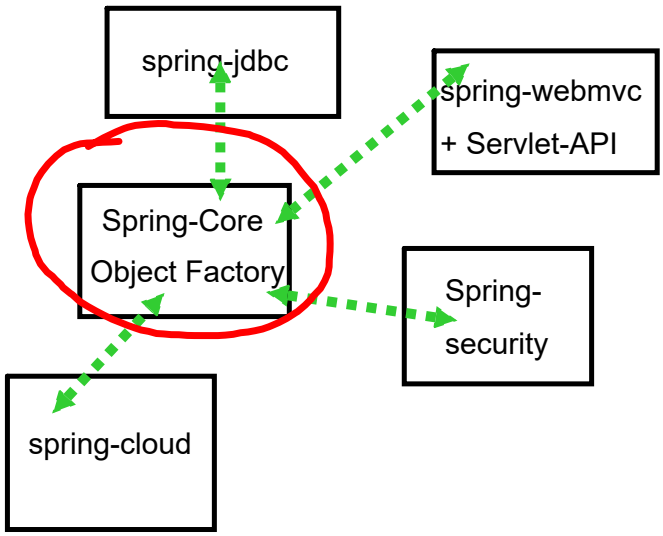
=> Object Factory : Responsible for creating and managing object

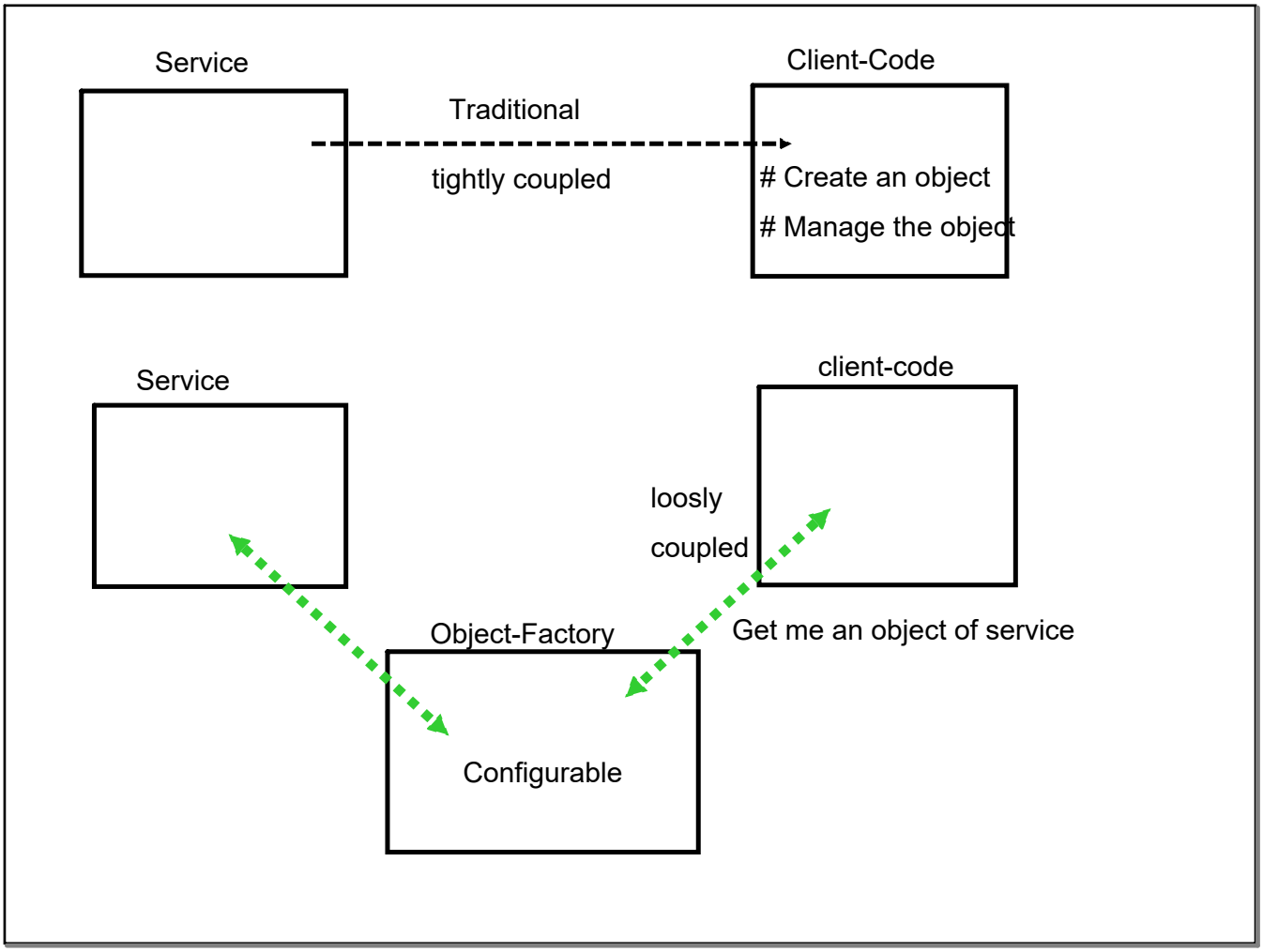
Increased the Productivity

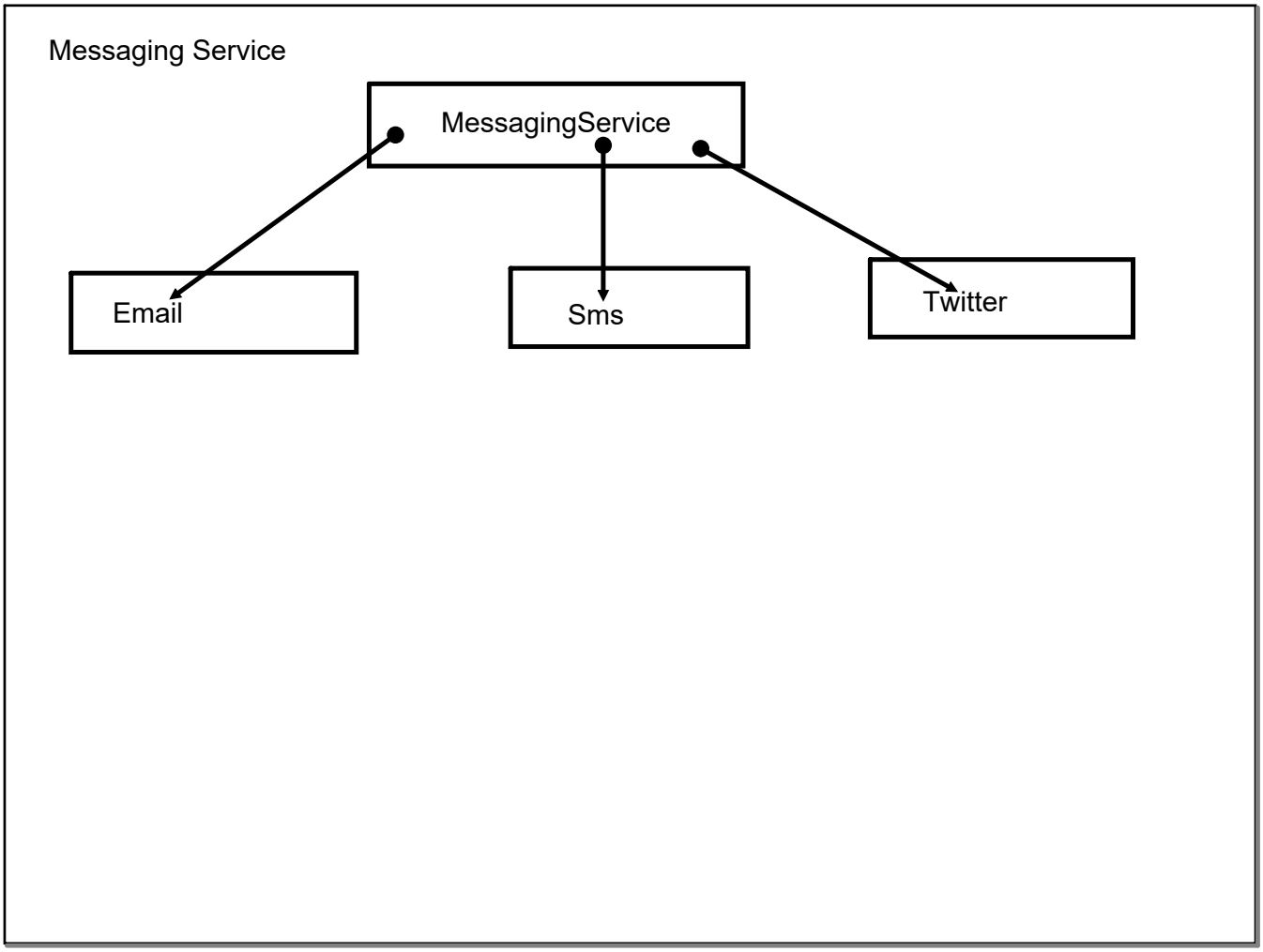
Increased the efficiency

1. Object Factory
2. Highly Modular
3. POJOs

Spring Framework




Service
Client-Code
Traditional
tightly coupled
Create an object
Manage the object
Service
client-code
loosly
coupled
Object-Factory
Get me an object of service
Configurable


Messaging Service
MessagingService
Email
Sms
Twitter

Object Factory | Bean Factory | Application Context

Provided by Spring - Core Module

A Custom Configuration needs to be provided to define the behavior of Object Factory

XML Based Configuration (Legacy)
Annotation Based Configuration (Modern)
Pure Java Based Configuration (Modern)

Std Spring Framework :

bundle of few Modules

=> Core

=> Spring-web-mvc

=> Spring AOP (proxy)

Bean Factory works on two key principals

1. IoC : Inversion of Control
2. DI : Dependency Injection

IoC : Outsourcing the (control of) creation and management of Object

XML Based Config :

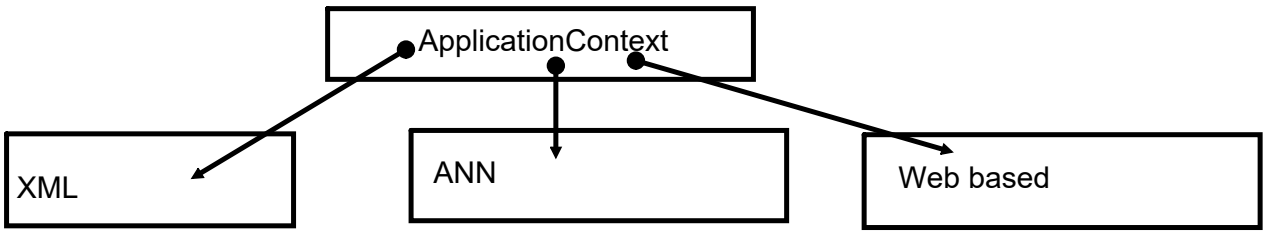
XML file + certain dependencies for support of additional spring tags

BEAN : Container(Object Factory) managed Object

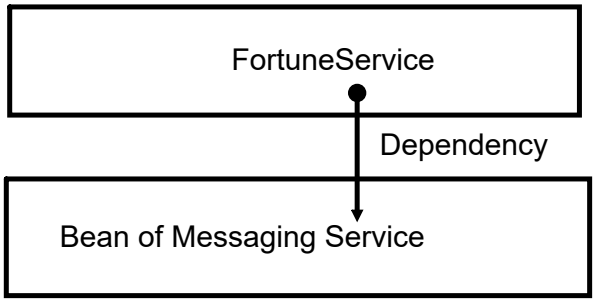
Multiple classes provided for Bean Factory

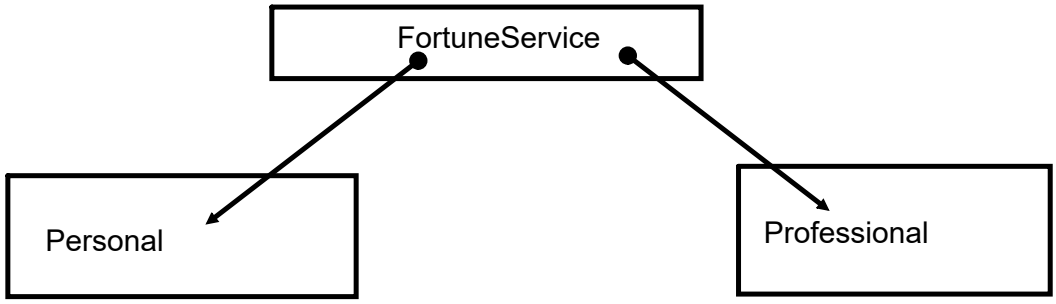
way of config (XML or java)

env for which bean factory (simple java, web app)



Each bean must have a unique id





Need the BEan factory to inject Fortune Service instance into EmailService

Constructor Based DI

Setter BAsed DI

```
<bean id="personal"
class="com.wf.training.spring.factory.service.PersonalFortune"></bean>
    <bean id="professional"
     <!-- Injecting Dependency -->

        <!-- Constructor Based -->
     <bean id="emailservice"
class="com.wf.training.spring.factory.service.EmailService">
        <constructor-arg ref="personal"/>
     </bean>
```

Injecting the literal values :

Delegate them to a text file (properties files)

literal values as key-value pair

need to specify property file in config

Bean Management :

1. Life cycle
2. Scope

=> Scope : Accessibility of bean

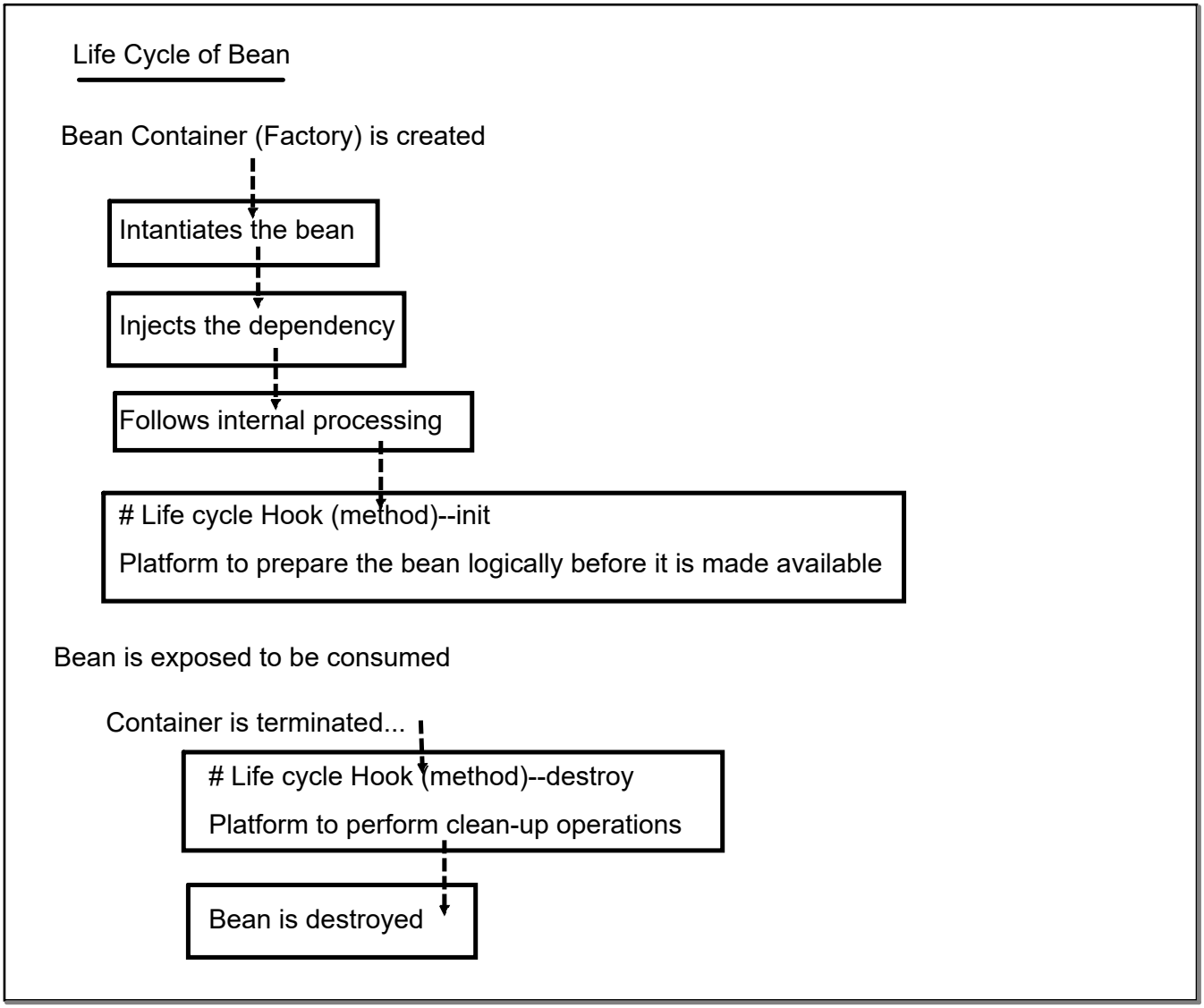
by default scope : Singleton : Single instance will be created

: Prototype : Diff each time

request

session : Web based

application


Life Cycle of Bean
Bean Container (Factory) is created
Intantiates the bean
Injects the dependency
Follows internal processing
Life cycle Hook (method)--init
Platform to prepare the bean logically before it is made available
Bean is exposed to be consumed
Container is terminated...
Life cycle Hook (method)--destroy
Platform to perform clean-up operations
Bean is destroyed

Prototype : BEan container does not maintain life cycle..

Annotation based config

xml file : path reference

Creating the bean

@Component :

Any class decorated with @Component will be initiated by bean factory

By default the class name itself becomes the id , first character being small case...

DI using annotation

1. Constructor
2. Setter
3. Field

@Autowired : search for bean, if found, inject it

Scope : @Scope

Life cycle hook methods : Annotations

Pure Java Based Config :

xml file will be replaced by Java class