=> The IOC containers of spring /spring boot frameworks are given for two basic operations
a) spring bean life cycle management
b) Dependency Management

1. Dependency Lookup (DL)
[Here the target spring bean searches and gets dependent spring bean class obj from various resources]

2. Dependency Injection (DI)
[Here the underlying server /container/frameworks/ JRE and etc.. dynamically assigns /injects dependent spring bean class obj to target spring bean class obj]

Dependency Lookup

=> Here the target spring bean /class dynamically searches and gets Dependent spring bean /class obj from various resources

=> Here the target spring bean /class pulls dependent spring bean /class obj

eg1 (non-technical) : Student (target) searching and getting course material (dependent) from the institute on demand

eg2: The way Java App searches and gets Jdbc obj reference from the JNDI Registry (JNDI : Java Naming And Directory Interface)

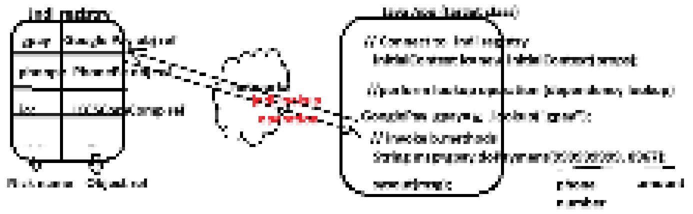
=> The object that is created inside the method is specific to that method (Local Scope/Method Scope)
=> The object that is created inside the class as the non-static property is specific to the object of a class (Object Scope)
=> The object that is created inside the class as static property is specific to entire class (class scope)
=> To provide more and more global visibility and accessibility for any object we need its reference in the Jndi Registry s/w

eg: Cos Registry , RMI Registry , Weblogic Jndi Registry , Tomcat Jndi Registry , GlassFish Jndi registry and etc..

note: Every web server (like Tomcat) and every App server (like weblogic) gives built-in Jndi registry

=> The important java objects of distributed Apps will be kept in Jndi registry to make them visible and accessible for local and remote clients

=> The objs like google pay, phone pe , SMS Service , ICC Score Service and etc.. will be kept in Jndi registry to provide global Visibility and accessibility for them



Jndi registry
gpay
GooglePay object
phonepe
PhonePe object
icc
ICCScoreComp obj
Nick name
Object ref
Java App (target class)
// Connect to Jndi registry
InitialContext ic=new InitialContext(props);
// perform lookup operation (dependency lookup)
GooglePay gpay=(...) ic.lookup("gpay");
// invoke b.methods
String msg=gpay.doPayment(9xxxxxxxxx, 6767);
network
Jndi lookup operation
phone number
amount

JDBC API (Technology)
Java App ------------------------> DB s/w

(Java App uses JDBC API to interact with DB s/w to manipulate DB tables Data nothing perform CURD Operations (insert, update, delete and select operations))

JNDI API (Technology)
Java App ------------------------> Jndi Registry

(Java app uses Jndi API to interact with Jndi registry to perform Jndi operations only)

The Jndi Operations are

a) Bind Operation :: Keeping the Object ref in Jndi registry having nick name
b) Unbind Operation :: Removing the Object ref from the Jndi registry
c) Rebind Operation :: Replacing the existing object with new object
d) Lookup operation :: Searching and getting Object ref from the Jndi registry

pros and cons of Dependency Lookup

pros

=> The target class/spring bean will search and get only the required Dependent spring bean /java class obj

cons

=> The target spring bean/java class should spend time and should write logic to search and get Dependent spring bean class objs (This increases the burden on the Programmer)

Dependency Injection

=> Here the underlying the server /Container /framework dynamically assigns Dependent class/ spring bean obj to target class/spring bean class obj.

=> Here the underlying env (like server/container /...) dynamically pushes Dependent spring bean /java class obj to target spring bean /java class object

eg1 :: The moment student (target) joins the course the he is assigned with Course material (dependent) automatically and dynamically

=> The IOC containers of spring /spring boot frameworks are given for two basic operations a) Spring Bean Life cycle management

b) Dependency Management

1.Dependency

Lookup (DL)

2. Dependency Injection (DI)

[Here the target spring bean

searches and gets dependent

spring bean class obj from various resources]

[Here the underlying server /container/framework/ JRE and etc.. dynamically assigns /injects dependent spring bean class obj to target spring bean class obj]

Dependency Lookup

===================

=> Here the target spring bean /class dynamically searches and gets Dependent spring bean/class obj from various resources

=> HEre the target spring bean/class pulls dependent spring bean/class obj

eg1 (non-technical) :: Student (Target) searching and getting course material (dependent)

from the institute on demand

eg1:: The way java App searches and gets imp obj references from

the JNDI Registry (JNDI :: Java Naming And Directory Interface)

=> The object that is created inside the method is specific to that method (Local Scope/Method Scope)

=> The object that is created inside the class as the non-static property

is specific to the object of a class (Object Scope)

=> The object the is created inside the class as static property is specific to entire

class (class scope)

=> To provide more and more global visibility and accessibility for any object we need its reference in the Jndi Registry s/w

eg: Cos Registry, RMI Registry, Weblogic Jndi Registry, Tomcat Jndi Registry, GlassFish Jndi registry and etc...

note:: Every web server (like Tomcat) and every App server (like weblogic) gives built-in Jndi registry

=> The important java objects of distributed Apps will be kept in Jndi registry to make them visible and accessible for local and remote clients

=> The objs like google pay, phone pe, BSE Service, ICC Scope Service and etc.. will be kept in Undi registry to provide global Visibility and accessibility for them

Java App (target class)

// Connect to Jndi registry

InitialContext ic=new InitialContext(props);

//perform lookup operation (dependency lookup)

Jndi registry

gpay Google Pay obj ref phonpe PhonePe objref

network

icc

ICCSCoreComp ref

indricekup operation

// invoke b.methods

Nick name Object ref

String msg=gpay.doPayment (999999999, 8967); sysout(msg);

phone number

amount

Google Pay gpay=ic .lookup("gpay");

Java App

JDBC API (Technology)

=======> DB s/w

(Java App uses JDBC API to interact with DB s/w to manipulate DB tables Data nothing perform CURD Operations (insert,update,delete and select operations)

Java App

JNDI API (Technology)

=======> Jndi Registry

(Java app uses Jndi API to interact with Jndi registry to perform Jndi operations on it) The Jndi Operations are

a) Bind Operation :: Keeping the Object ref in Jndi registry having nick name b) UnBind Operation :: Removing the Object ref from the Jndi registry

c) ReBind Operation :: Replacing the existing object with new object

d) Lookup operation :: Searching and getting Object ref from the Jndi registry

pros and cons of Dependency Lookup

====

pros

=> The target class/spring bean will search and get only the required Dependent spring bean/java class obj

cons

=>The target spring bean/java class should spend time and should write logics to search and

get Dependent spring bean class objs (This improves the burden on the Programmer)

Dependency Injection

======

==

=> Here the underlying the server /container/Framework dynamically assigns Dependent class/ spring bean obj to target class/spring bean class obj.

=> HEre the underlying env. (like server/container /...) dynamically pushes Dependent spring bean /java class obj to target spring bean /java class object

eg1:: The moment student(target) joins the course the he is assigned with Course material (dependent) automatically and dynamically

eg2:: The way JVM assigns default values (0,0.0,null,false as dependent values) to java class obj(target) the it creates the object is called Dependency Injection

eg3:: The way Servlet Container assigns SErvletconfig object(dependent class obj) to Servlet comp class obj (target object)is called Dependency Injection

pros(advantages)

==============

=> The target spring bean /java class can use dependent objs directly as they are available readily injected by underlying server/container/.....

=>The target spring bean /java class need not to spend time and need not to write logics to search and get dependents

cons

====

=> The underlying container /server /framework may inject both necessary unnecessary dependent objs to target spring bean