W

Limitations of implementing strategy DP Using Core Java

1) we need to implement factory pattern manually having logic to create target class,depndent class objs and assigning dependent class obj to target class obj (manually dependency management)

2) No support of Internal Cache maintaining

objs

3) we need to remember implement all the rules of strategy DP manually.. This improves burden the Programmer since he needs to take care of multiple things

To overcome these problems implement strategy DP Using spring framework as show below

Strategy DP using Spring

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

//ICourier.java (Common Interface) package com.nt.sbeans;

public interface ICourier {

public String deliver(int oid);

IOCProj04-StrategyDP-Spring Core JRE System Library [JavaSE-17]

#src

com.nt.client

> Strategy DPTest.java

#com.nt.config

> AppConfig.java

›

com.nt.sbeans

BlueDart.java

› DTDC.java

> Flipkart.java

>

ICourier.java

Referenced Libraries

}

> spring-expression-6.1.4.jar - C:\Users\Nataraz\Downl

>

spring-jcl-6.1.4.jar - C:\Users\Nataraz\Downloads

>

spring-aop-6.1.4.jar - C:\Users\Nataraz\Downloads

> spring-core-6.1.4.jar - C:\Users\Nataraz\Downloads

>

spring-beans-6.1.4.jar - C:\Users\Nataraz\Downloads

> spring-context-6.1.4.jar - C:\Users\Nataraz\Download

```
>
spring-context-support-6.1.4.jar - C:\Users\Nataraz\D
//BlueDart.java (Dependency class1) package com.nt.sbeans;
import org.springframework.context.annotation.Lazy; import org.springframework.stereotype.Component;
@Component("bDart")
@Lazy(true)
public final class BlueDart implements ICourier {
public BlueDart() {
System.out.println("BlueDart:: O-param constructor");
}
@Override
public String deliver(int oid) {
return "BlueDart courier is ready to deliver "+oid+" order number products";
}
}
//DTDC.java (depedent class2)
package com.nt.sbeans;
import org.springframework.context.annotation.Lazy; import org.springframework.stereotype.Component;
@Component("dtdc")
//@Component("courier")
//@Primary
@Lazy(true)
public final class DTDC implements ICourier {
public DTDC() {
System.out.println("DTDC:: 0-param constructor");
}
@Override
public String deliver(int oid) {
}
return "DTDC courier is ready to deliver "+oid+" order number products";
}
}
//AppConfig.java (Confguration class)
package com.nt.config;
import org.springframework.context.annotation.ComponentScan; import
org.springframework.context.annotation.Configuration;
@Configuration
```

```
@ComponentScan (base Packages = "com.nt.sbeans")
public class AppConfig {
}
//StrategyDPTest.java (Client App)
package com.nt.client;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import com.nt.config.AppConfig;
import com.nt.sbeans.Flipkart;
public class StrategyDPTest {
public static void main(String[] args) {
//create the IOC container
AnnotationConfigApplicationContext ctx=
new AnnotationConfigApplicationContext(AppConfig.class);
//get target spring bean class obj ref
Flipkart fpkt=ctx.getBean("fpkt",Flipkart.class);
//invoke the b.method
String resultMsg=fpkt.shopping(new String[] {"shirt","trouser"},
System.out.println(resultMsg);
//close the container
new double[] {90000.0,50000.0});
}
}
ctx.close();
//Flipkart.java (target class)
package com.nt.sbeans;
import java.util.Arrays;
import java.util.Random;
import org.springframework.beans.factory.annotation.Autowired; import
org.springframework.beans.factory.annotation.Qualifier; import org.springframework.stereotype.Component;
@Component("fpkt")
public final class Flipkart {
//HAS- property @Autowired @Qualifier("bDart")
private ICourier courier;
public Flipkart() {
System.out.println("Flipkart:: 0-param constructor");
}
//b.method
```

public String shopping(String items[], double prices[]) {

System.out.println("Flipkart.shopping()");

//calculate bill amount

double billAmt=0.0;

for(double p:prices) {

}

billAmt-billAmt+p;

//generate order id randomly

int oid=new Random().nextInt(100000);

// deliver the order using couier

String msg=courier.deliver(oid);

return Arrays.toString(items)+"are shopped having bill amount::"+billAmt+" --->"+msg;

}

Q) while injecting one of the multiple dependent spring beans of the same type to the HAS-A property of the target spring bean class what is the meaning of ambiguity problem (NoUniqueBeanException) and how to solve that problem?

Ans) if IOC container detects more than one dependent spring bean to Inject to @Autowired enabled HAS-A property of target spring bean class then it raises the ambiguity problem by throwing "NoUniqueBeanException"

Definitation

Caused by: org.springframework.beans.factory.NoUniqueBeanDefinitionException: No qualifying bean of type 'com.nt.sbeans. ICourier' available: expected single matching bean but found 2: bDart,dtdc

This Problem can be solved in 3 ways

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

a) using @Primary (byType mode of Autowiring)

b) Using @Qualifer(-) (best) (by name mode of Autowiring)

c) By Matching HAS-A property name

of the target spring bean with dependent spring bean id

=>if the dependent spring bean is decided based its class name(type) to inject to target spring bean's HAS -A property then it called ByType mode Injection/AutoWiring

(byNAme mode of Autowiring)

=>if the dependent spring bean is decided based its bean id (name) to inject to target spring bean's HAS - A property then it is called ByName mode of Injection/AutoWiring

a) using @Primary

In one of the Dependent class, place @Primary annotation

@Component("dtdc")

@Primary

public final class DTDC implements ICourier {

...

...

}

@Primary can be also used along with @Bean methods of the @Configuration class

note:: placing @Primary on multiple same type dependent spring beans will also give Ambiguity Problem

b) Using @Qualifer(-) (best)

On the @Autowired enable HAS-A property of target spring bean class place @Qualifier(-) annotation having

one of the dependent spring bean id

@Component("fpkt")

public final class Flipkart {

//HAS- property

@Autowired

@Qualifier("bDart")

private ICourier courier;

}

c) By Matching HAS-A property name

of the target spring bean with dependent spring bean id

@Component("courier")

matching with HAS-A property name of target spring bean class

public final class DTDC implements ICourier {

}

...

Why the @Qualifier(-) solution is the best solution while solving the ambiguity Problem?

Ans1) we can get the dependent spring bean id passed in the @Qualifier(-) annotation from the properties file with out touching the java source code the project i.e 100% Loose coupling can be achieved. (will be discussed in fututure)

Ans2) if we apply all the 3 solutions to resolve /solve the ambiguity Problem pointing to 3 different dependent spring beans then the spring bean located by the @Qualifier(-) will be taken as the final one to inject

Advantages of implementing strategy DP using Spring framework?

a) No need of implementing Factory Pattern separately.. becoz the IOC container itself

acts as the factory Pattern towards spring beans instantiation and injecting the

the dependent spring bean to target spring bean

b) Gives the built-in internal cache in the IOC container to store and manage the

spring bean class obj refs.. having feature called reusability

c) Allows to enable or disable eager instantiation of the spring beans

(we can disable pre-instantiation with the support of @Lazy(true) annotation)

d) Reduces the burden on the programmers.. by taking care of multiple

activities internally

and etc...

What is the difference b/w Eager/Pre/early Instantiation and Lazy/Late Intstantiation?

Ans) if the spring bean class object is created the moment IOC container is created irrespective of whether the spring bean will be used or not in the next steps is called pre/early/eager- instantiation of spring beans.. if no scope is specified in the spring bean the default scope is "singleton".. By default all singleton scope spring beans participates in eager/early/pre instantiation and more over their objects will be placed in the internal cache of the IOC container for reusability

if the IOC container is creating the spring bean class obj only when the need is there like when ctx.getBean(-) is called

or when injection process is started then that is called lazy/late instantiation..

note:: we can disable pre-instantiation on the singleton scope spring bean using @Lazy(true) annotation by keeping the annotation along with @Component or @Bean

note:: Other than "singleton" scope spring beans enabled with Lazy instantiation by default

=>prototype, request, session, application, websocket

note:: Spring bean scopes is a separate chapter to learn (future discussion)

=> The default scope for any spring bean is "singleton"

Limitations of implementing strategy DP Using Core Java

========

==========

1) we need to implement factory pattern manually having logic to create target class,depndent class objs

and assigning dependent class obj to target class obj

2) No support of Internal Cache maintaining spring bean class objs

3) we need to remember implement all the rules of strategy DP manually.. This improves burden

the Programmer since he needs to take care of multiple things

To overcome these problems implement strategy DP Using spring framework as show below

In the Project /App Development, the IDE provide first level easiness .. To add another level more easiness take the support of build tools like maven,gradle and etc..

`=> we can use these build tools in two modes`

a) CLI mode (from Command prompt)

b) IDE mode (By linking with IDE)

Some Basics about maven keeping our code ready for execution is called build process.maven,gradle are the build tools to simplify the

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

project build process

=> maven is a build tool .. that helps end to end App/Project development and release process

=> maven gives archetypes having Project Templates

(gives folder structure for different types of projects) =>maven-archetype-quickstart ---- for standalone apps

=>maven-archetype-webapp ---- for webapplications

=> maven gives support dependency management

management

maven is build tool cum

Project management tool

For developing different types of the Projects

Build Path

(Can download jar files to the Project from diffrent repositories (storage places)

note:: able download both main and dependent jar files from the different repositories (supports transitive dependency)

=> Can pack the App/Project into war file or jar file

=> Can run junit testcases to generate the test reports

Project archetype= Project Template

Remote/Private Repositories Third party Repositories (specfic to each company) eg:: IBM repository, Oracle Repository

Central Repository (from the intenet) (https://repo.maven.apache.org/maven2)

Local repository (With in the System) (C:\Users\Nataraz\.m2\repository)

jar file :: java archieve (represents standalone Apps)

war file :: web application archieve

(represents the websites)

War ::: web application archieve (a flavor of jar file)

Maven brings jar files(dependencies) to the Project build path from the repositories in the following order

(a) Local Repository (if not available here then) (b) Maven Central Repository (if not available here) (c) Specified one or another Remote Repository

note::jar file(s) collected from Central or Remote Repository will be placed in Local Repository before placing them in Project's build path

(junit is used as the unit testing tool by programmer

to test his own piece of code)

=>test case is a test plan where expected results

will be matched with actual results in all permitations and combinations)

=> maven gives built-in decompiler to see the source code

=> In Maven jar file / plugin / project is identified with 3 details

=>group Id (company name of jar file /plugin/project) =>artifact Id (name of the jar file /plugin/project)

=> version (version of the jar file /plugin/project)

=>In the version, if the word "SNAPSHOT" is there we can say Project/jar/plugin

is under development

=>In the version, if the word "RELEASE" is there we can say Project/jar/plugin

is realeased note:: In Latest Versions of eclipse IDE.. maven comes as the built-in tool...

note::: Maven, Gradle both are Build tools (maven is popular)

note:: we give all inputs to maven through pom.xml file

plugin is patch s/w (addtional software) that provides extra functionalaties to existing software ..

eg:: amazon firestik makes normal TV as the smart TV

eg:: plugins added to browsers (GSon plugin, TestNg plugin and etc..)

POM :: Project Object IModel

=> In maven project build process we generally deal with archetypes, dependencies(jar files), Projects and plugins

procedure to develop Strategy DP application by using spring framework in Eclipse IDE Using Maven tool

step1) create maven Project in Eclipse IDE by choosing "maven-archetype-quickstart" as the archetype

File menu -----> new ------> maven Project -----> next --->

search for achetype (maven-archetype-quickstart) ---->

Catalog: All Catalogs

Filter: maven-archetype-qu

Group Id

com.github.ywchang

com.haoxuer.maven.archetype org.apache.maven.archetypes

Artifact Id

Version

maven-archetype-quickstart maven-archetype-quickstart

1.1

1.01

maven-archetype-quickstart

1.4

New Maven Project

New Maven project

Specify Archetype parameters

Group Id:

д

nit (company name)

ArtifactId: IOCProj3-StrategyDP03-Spring

Version:

Package:

0.0.1-SNAPSHOT

com.nt.test

Properties available from archetype:

(Project)

(version of the Project)

(default pkg name)

name of the

POM:: Project Object Managememt

The standard folders in maven standalone project (maven-archetype-quickstart)

src/main/java ---> To create packages related to project source code src/test/java ---> To create packages related to project unit testing code src/main/resources ---> To keep supporting files related to Project source code like properties files, yml files and etc..

src/test/resources ---> To keep supporting files related to Project unit test code like properties files, yml files and etc..

The standard folders in maven web app project (maven-archetype-quickstart) src/main/java ---> To create packages related to project source code src/test/java ---> To create packages related to project unit testing code src/main/resources ---> To keep supporting files related to Project source code like properties files, yml files and etc..

src/test/resources ---> To keep supporting files related to Project unit test code like properties files, yml files and etc.. (static web comps) src/main/webapp ----> To place the web application related view comps (html files,jsp files, audio files, vedio files and etc...)

Name

Value

step2) Understand the generated directory structure

>

Project Folder

↑

IOCProj03-StrategyDP03-Spring

src/main/java

src/test/java

> JRE System Library [JavaSE-1

folder maintaining inputs

> Maven Dependencies

src

target

M pom.xml

folder to hold outputs

To give instructions

to maven tool

to

next->finish

maintains

Jdk supplied JRE

To place packages related

to source code development

To place packages related to

unit testing code devleopment (related to junit)

shows all the jar files (dependencies) added

through maven

POM :: Project Object Model (pom)

step 3) Chanage java version 17 (ur choice) in pom.xml and perform maven update to reflect.

in pom.xml

17

```
<maven.compiler.source>7</maven.compiler.source>
<maven.compiler.target></maven.compiler.target>
```

17

maven-archetype-quickstart is the maven archetype/project template for standalone apps

=>Perform maven update --> right click on the Project ---> maven ---> update --->ok

step4) add dependencies (jar files) to the Project through pom.xml file

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-context-support -->
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-context-support</artifactId>
<version>6.1.5</version>
</dependency>
```

go to mvnrepository.com ---> search for spring context support-->

select springContext support--->

select latest verison

(6.16)

--->

copy the xml tags ---> paste in

Maven supports transitive dependency i.e maven downloads given main jar files, their dependent jar files and the dependents of the dependent jar files, this is called transitive depndency

pom.xml under <dependencies> tags -->save the project

and observe ---> maven dependencies section

popular maven terminologies

a) maven archetype = Project Template

b) maven artifact fact = jar name/plugin name/project name

c) dependency = jar file/library

d) pom.xml = file to give inputs to maven

✓ Maven Dependencies

spring-context-support-6.1.6.jar - C:\Use

spring-beans-6.1.6.jar - C:\Users\Nataraz spring-context-6.1.6.jar - C:\Users\Natari

Transitive spring-aop-6.1.6.jar - C:\Users\Nataraz

dependency spring-expression-6.1.6.jar - C:\Users\Na

micrometer-observation-1.12.5.jar - C:\U

micrometer-commons-1.12.5.jar - C:\Us

spring-core-6.1.6.jar - C:\Users\Nataraz\.

spring-jcl-6.1.6.jar - C:\Users\Nataraz\.m

>junit-4.11.jar - C:\Users\Nataraz\.m2\rep

>hamcrest-core-1.3.jar - C:\Users\Nataraz

@Lazy(true) disables the eager/early/pre instantiation of the

spring beans i.e their objects will not be created to moment container is started/created

Using maven if u add main jar file .. then it automatically downloads

main jar file and its dependent jar files by seaching in

Local Repository (current computer) ----> central Repository.

(internet)

step5) create the following pkgs in src/main/java folder

#src/main/java

com.nt.cfgs

com.nt.sbeans

com.nt.test

step6) add spring beans to com.nt.sbeans pkg

✓com.nt.sbeans

> BlueDart.java

› Courier.java

› DTDC.java

> Flipkart.java

step8) Develop the client App in com.nt.test pkg

✓com.nt.test

› StrategyDP Test.java

step 9) Run the Client App

Right click in StrategyDPTest.java ---> run as ---> run as Java App

IOCProj03-StrategyDP03-Spring

#src/main/java

com.nt.sbeans

>

BlueDart.java

> Courier.java

(g) Searches for

> DTDC.java

> Flipkart.java

com.nt.test

> Strategy DPTest.java

> #src/test/java

the @Component

Spring bean s

> JRE System Library [JavaSE-17]

✓Maven Dependencies

>

>

Configuation

gets 3 spring beans

junit-4.11.jar - C:\Users\NATARAJ\.m2\repository\junit\juni

hamcrest-core-1.3.jar - C:\Users\NATARAJ\.m2\repository\

>spring-context-support-5.3.23.jar - C:\Users\NATARAJ\.m2\

>spring-beans-5.3.23.jar - C:\Users\NATARAJ\.m2\repository >spring-context-5.3.23.jar - C:\Users\NATARAJ\.m2\repositor >spring-aop-5.3.23.jar - C:\Users\NATARAJ\.m2\repository\o >spring-expression-5.3.23.jar - C:\Users\NATARAJ\.m2\repos >spring-core-5.3.23.jar - C:\Users\NATARAJ\.m2\repository\c >spring-jcl-5.3.23.jar - C:\Users\NATARAJ\.m2\repository\orc

> src

>

target

M pom.xml

What is transitive dependency in Maven?

Ans) when we add main jar file to the maven

Flow of execution

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

```
//Courier.java (Common Interface)
package com.nt.sbeans;
public interface Courier {
}
public String deliver(int oid);
//DTDC.java (dependent class1)
package com.nt.sbeans;
```

(h) Searches for are there any @Bean methods

in @Configuration class (Somehow not there)

(i) IOC container performs the eager instanitation of

singleton scope spring beans (if no scope is given the default scope is singleton scope)

import org.springframework.stereotype.Component;

@Component("dtdc")

public final class DTDC implements Courier {

project, the maven downloads depedent jar files

and their depdent jar files of hierarchy ..This process

is called transtive dependency

}

@Override

public String deliver(int oid) {

IOC container

#e

Flipkart obj

OQ

DTDC obj Blue Dart obj

appConfig obj

(#1)

(#i) (#i)

return oid+" order items are kept for delivery by DTDC";

(Injection takes place)

(#m)

(j) @ComponentScan annotation activates the code related

@Autowired annotation and that codes searches for @Autowired annotation location in the spring beans finds in Flipkart class

}

je

//BlueDart.java (dependent class2)

package com.nt.sbeans;

import org.springframework.stereotype.Component;

@Component("bDart")

public final class BlueDart implements Courier {

@Override

(v)

Maven gives built-in decompiler, just press F3 by keeping the cursor on the Soruce code.

To see the source code of any random class, take the support of

Ctrl+shift+T option

public String deliver(int oid) {

return oid+" order items are kept for delivery by BlueDart"; (w)

(1) IOC container searches for the Dependent spring bean

of type Courier(I).. but it finds two dependent spring bean class objects (BlueDart, DTDC).. But select one

S

dependent spring bean (that is BlueDart) based on Qualifer(-) annotation bean id

}

}

//Flipkart.java (target class)

package com.nt.sbeans;

import java.util.Arrays;

import java.util.Random;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Qualifier;

import org.springframework.stereotype.Component;

@Component("fpkt")

public final class Flipkart {

//HAS-A property

@Autowired (#j)

@Qualifier("bDart") (#1) private Courier courier;

(k) The code of @Autowired collects the type

(m) The chose dependent spring bean BlueDart class obj

will be injected to the target spring bean class obj (Flipkart)

(n) IOC container keeps the spring bean class objs in the internal

cache of the IOC container

Internal Cache of the IOC container

(q?) Flipkat class obj ref

fpkt

dtdc

of the HAS-A property that Courier(I) and name of the

DTDC class obj ref

// b.method

property (courier)

appConfig AppConfig obj ref

bDart

BlueDart obj ref

public String shopping(String items[], double prices[]) {

}

//calculate bill amount (t)

double billAmount=0.0;

for(double p: prices) {

billAmount=billAmount+p;

}

// generate the order id (random number as the order)

int oid=new Random().nextInt(1000);

// deliver the products using courier

String msg=courier.deliver(oid);

(x)

(U)

return Arrays.toString(items)+" items with billAmount:::"+billAmount+" - "+msg;

windows+shift+s :: To get screen short

(y)

AppConfig.java

package com.nt.config;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

@Configuration

(e) Loads the Configuration class and cretes the object

@ComponentScan(base Packages = "com.nt.sbeans") (f) collects the package name

public class AppConfig {

from @ComponentScan

as the spring bean

}

Client App

package com.nt.client;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.nt.comps.Flipkart;

import com.nt.config.AppConfig;

public class StrategyDPTest {

(a) -> Run the application

(0) AnnotationConfigApplicationContext ctx= (c) -> Container creation

(b)

public static void main(String[] args) {

// create the IOC container

//get Target spring bean class obj

(r)

//invoke the B.methods (p)

new AnnotationConfigApplicationContext(AppConfig.class);

Flipkart fpkt=ctx.getBean("fpkt", Flipkart.class);

(d) Takes the class as the @Configuration class

(z) String resultMsg=fpkt.shopping(new String[] {"shirt","trouser"}, new float[] {60000.0f,70000.Of });

System.out.println(resultMsg); (5)

(a1)

ct.close(); // (b1) --> closes the container by

}

(c1) end of the application

vanishing the objects

advantages of developing strategyDP Application using spring

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

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

a) No need of creating seperate Factory becoz the IOC container itself acts as Factory

b) IOC container gives internal cache to maintain spring bean objs to give reusability

c) IOC container takes care of spring bean life cycle management and dependency management

(This reduces burden on the Programmers)

and etc...

What is NoUniqueBeanDefinitation Exception /Ambiguity Problem and How to solve that Problem?

Ans) if multiple dependents are there to get Injected to target class spring Bean HAS-A property then

the IOC container goes to dilema state and rasises the NoUniqueBeanDefinitation FoundExeception

problem code

//target class

@Component("fpkt")

public final class Flipkart {

//HAS-A property

@Autowired //field Injection

@Qualifier("dtdc")

private Courier courier;

@Component("dtdc")

//@Primary

public final class DTDC implements Courier {

...

....

}

@Component("bDart")

public final class BlueDart implements Courier {

}

....

...

}

raises the exception

Caused by: org.springframework.beans.factory.NoUniqueBeanDefinitionException: No qualifying bean of type 'com.nt.comps.Courier' available:

expected single matching bean but found 2: bDart,dtdc

This problem can be solved in 3 approches

a) Using @Primary on the one of dependent spring bean

b) Using @Qualifier(-) having dependent spring bean id (best)

c) By taking one Dependent spring bean id matching HAS-A property name of

the

target class

a) Using @Primary on the one of dependent spring bean

//target class

@Component("fpkt")

public final class Flipkart {

//HAS-A property

@Autowired //field Injection

private Courier courier;

@Component("dtdc") @Primary

public final class DTDC implements Courier {

...

....

}

@Component("bDart")

public final class BlueDart implements Courier {

}

}

www.

b) Using @Qualifier(-) having dependent spring bean id (best)

//target class

@Component("fpkt")

```
public final class Flipkart {
//HAS-A property
@Autowired //field Injection
@Qualifier("dtdc")
private Courier courier;
@Component("dtdc")
public final class DTDC implements Courier {
....
}
@Component("bDart")
public final class BlueDart implements Courier {
...
...
}
}
....
```

is best

@Qualiifer(-) becoz the required bean if for @Qualifer(-) can be passed

with out of disturbing the the source code target and dependent sporing beans

c) By taking one Dependent spring bean id matching HAS-A property name of the target class

```
//target class
@Component("fpkt")
public final class Flipkart {
//HAS-A property
@Autowired //field Injection
private Courier courier;
@Component("dtdc")
public final class DTDC implements Courier {
}
@Component(" courier")
public final class BlueDart implements Courier {
}
}
```

@Primary is class level and method level annotation

@Qualifer is field level, param level annotation

IOCProj06-Strategy DP-Spring ✓src/main/java

#com.nt.client

> Strategy DPTest.java

#com.nt.config

> AppConfig.java

com.nt.sbeans

> DesielEngine.java

> IEngine.java

>

PetrolEngine.java

<

Vehicle.java

>

src/test/java

> JRE System Library [JavaSE-17]

✓

Maven

Dependencies

Another App on strategy DP

>spring-context-support-6.1.2.jar - C:\Users\Nataraz\.r

>spring-beans-6.1.2.jar - C:\Users\Nataraz\.m2\reposit

>spring-context-6.1.2.jar >spring-aop-6.1.2.jar - C:\Users\Nataraz\.m2\repositor

- C:\Users\Nataraz\.m2\repos

>

>

spring-expression-6.1.2.jar - C:\Users\Nataraz\.m2\re

micrometer-observation-1.12.1.jar

micrometer-commons-1.12.1.jar

- C:\Users\Nataraz

- C:\Users\Nataraz\.

>spring-core-6.1.2.jar - C:\Users\Nataraz\.m2\reposito

>spring-jcl-6.1.2.jar - C:\Users\Nataraz\.m2\repository\

> junit-4.11.jar - C:\Users\Nataraz\.m2\repository\junit\

hamcrest-core-1.3.jar - C:\Users\Nataraz\.m2\reposit

>

src

>

target

```
Mpom.xml
//IEngine.java (common interface)
package com.nt.sbeans;
public interface IEngine {
public void startEngine();
public void stopEngine();
}
//PetrolEngine.java
package com.nt.sbeans;
3
import org.springframework.stereotype.Component;
;
› @Component("pEngine")
'public class Petrol Engine implements IEngine {
Je
)
@Override public void startEngine() {
System.out.println("PetrolEngine:: Petrol Engine started");
je
@Override
public void stopEngine() {
System.out.println("PetrolEngine:: Petrol Engine stopped");
3
}
)
}
//DieselEngine.java
package com.nt.sbeans;
import org.springframework.stereotype.Component;
@Component("dEngine")
public class DesielEngine implements IEngine {
@Override
public void startEngine() {
System.out.println("DesielEngine:: Diesel Engine started");
}
@Override
public void stopEngine() {
}
}
System.out.println("DesielEngine:: Desiel Engine stopped");
//AppConfig.java
package com.nt.config;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
@Configuration @ComponentScan (basePackages = "com.nt.sbeans")
public class AppConfig {
}
//Client App
package com.nt.client;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import com.nt.config.AppConfig;
import com.nt.sbeans.Vehicle;
public class StrategyDPTest {
public static void main(String[] args) {
//create IOC Container
AnnotationConfigApplicationContext ctx=new AnnotationConfigApplicationContext(AppConfig.class);
//get Target spring bean class obj Vehicle vehicle=ctx.getBean("vehicle",Vehicle.class);
//invoke the b.method
vehicle.jounery("hyd", "goa");
//close the container
ctx.close();
}
}
```

Assignment (using strategy DP)

==========

target spring bean:: com.nt.sbean.Studnet

dependent spring beans :: ICourse Material (1)

JAvaMetrial(c)

otNetMaterial(c)

python Meterial (c)