Date : 08/09/2020

Spring Boot 7AM

Mr. RAGHU

-------------------------------------------------

**Chapter#1 Spring Boot Core**

https://www.youtube.com/watch?v=EA43S5R8LSc&list=PLVlQHNRLflP9XSWeY4x4FLwnL3UOIxnTr

a. Runners

b. Properties | YAML

c. @Value | \*\* @ConfigurationProperties

d. Profiles

e. Scheduling

f. Spring Core Annotation

@ComponentScan

@PropertySource

@Configuration

@Bean

...etc

g.\*\*\* main() class work flow

-------------------------------------------------------------------

a. Runners : To execute any logic only one time after starting application.

=> These are implemented by programmer, auto-executed(called) by Starter class[Main class]

=> These will accept input from main class | Properties file.

=> \*\*\* Runners are generally used at non-web application, just for test our code.

ex: Test Jdbc Application, Test Email Application..etc

=> \*\* JobScueduling (used by Spring Boot it self) is runner.

-Types of Runners--

i. CommandLineRunner (I) [Spring boot 1.0]

ii. ApplicationRunner (I) [Spring boot 1.3]

-----------------------------------------------------------------------

=> ctrl+shift+T - Open Type (Pre-defined cod files)

=> ctrl+O - overview ( variables, methods...)

=> ctrl+L - goto Line

-----------------------------------------------------------------------

\*) Both are functional Interfaces [an interface contains only one abstract method]

having method run(\_) that gets executed by Spring Starter onetime on application startup.

---First Application---------------

#1. Create one Spring Starter Project

> File > new > Spring Starter Project

> Enter details

Name : SpringBoot2FirstApplication

Package/GroupId : in.nareshit.raghu

Version : 14

> Next > Next > Finish

#2. Create one class under 'in.nareshit.raghu' package (or its sub package)

package in.nareshit.raghu.runner;

//ctr+shift+O

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

@Component

public class MessageRunner implements CommandLineRunner {

@Override

public void run(String... args) throws Exception {

System.out.println("FROM BOOT FIRST APPLICATION!");

}

}

#3. Goto Main class

> Right click on code > Run As > Spring Boot application.

----------------------------------------------------------------

Functional Interface: - JDK 1.8

=> Only one abstract method (including its parent)+ zero or more default/static methods

=> Applying @FunctionalInterface is optional, if we add then Java compiler

will cross verify all Functional Interface rules.

-Are given Examples valid Functional Interfaces are not?--

#1

interface Sample {}

A) Invaild [Must have one abstract method, zero found]

#2

interface Sample {

void execute();

}

A) Valid.

#3

interface Sample {

void execute();

String toString();

boolean equals(Object ob);

int hashCode();

}

A) Valid. We can add methods which are existed in java.lang.Object(C).

That indicates to sub class, implement those.

#4.

interface Sample {

void test();

}

interface Export extends Sample { }

A) Both are valid. Sample has one abstract method and Export has one abstract method

which is derived from its parent interface.

#5.

interface Chioce {} ==> zero abstract methods

interface Sample extends Choice {

void test();

}

interface Model extends Sample { ==> 2 abstract methods

void find();

}

A) Sample is Valid. Both Chioce and Model are invaild.

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

\*)Lambda Expressions:- JDK 1.8

Lambda Expressions = it is a class(without any name) + object

=> Lambda Expressions can be provided only for functional interfaces.

=> Lambda Expressions Syntax

Interface ob = (method params) -> { method body };

--Example--

#1. interface Sample {

void show(String name);

}

Lambda Expression:

Interface ob = (method params) -> { method body };

Sample s = (String name) -> {

sysout("Hello" + name );

};

=> Method Param types are optional while writing lambda expression.

Sample s = (name) -> {

sysout("Hello" + name );

};

=> If expression has only one parameter then symbol () optional and

has only one statement in body then {} also optional.

Sample s = name -> sysout("Hello" + name);

-----------------------------------------------

#2 interface Data {

void print(String code);

}

A)

Data ob = code -> sysout("WELCOME TO " + code);

#3. interface CommandLineRunner {

void run(String... args);

}

A) CommandLineRunner cob = (args) -> { sysout("FROM LAMBDA SYNTAX CODE"); };

CommandLineRunner cob = args -> sysout("FROM LAMBDA SYNTAX CODE");

CommandLineRunner cob = abc -> sysout("FROM LAMBDA SYNTAX CODE");

//all are vaild.

===2nd Application-START=========================================================

#1. Create one Spring Starter Project

> File > new > Spring Starter Project

> Enter details

Name : SpringBoot2AppTwo

Package/GroupId : in.nareshit.raghu

Version : 14

> Next > Next > Finish

#2. Create AppConfig class

package in.nareshit.raghu.config;

//ctrl+shift+O

import org.springframework.boot.CommandLineRunner;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class AppConfig {

// 1 method = 1 object

@Bean

public CommandLineRunner clra() {

//lambda expression syntax

/\*

CommandLineRunner cob = (args) -> {

System.out.println("FROM LAMBDA EXP!");

};

return cob;

\*/

return inputs -> System.out.println("FROM LAMBDA EXP!");

}

}

#3. Run main class/Starter class.

===2nd Application-END=========================================================