Date : 15/12/2020
Spring Boot 9AM
Mr. RAGHU

@ConfigurationProperties(prefix="anyword")

*)Working with 1D Collections : List/Set and even Array

If we define a variable of type List/Set or Array then in properties file
we have to pass data using below syntax:

```
     prefix.variableName[index]=value
```

*) Index numbers must start from zero, should be given in order,
   else application will not started.

---------code-------------
#1. Create Spring Starter project

Name : SpringBoot2RunnerConfigPropsCollectionEx
Package: in.nareshit.raghu

#2. application.properties
```
# prefix.variable[index]=value
my.app.data[0]=A
my.app.data[1]=B
my.app.data[2]=A
my.app.data[3]=B
```

#3. Runner class
```
package in.nareshit.raghu.runner;

import java.util.Set;

import org.springframework.boot.CommandLineRunner;
import
org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

//ctrl+shift+O

@Component
@ConfigurationProperties(prefix = "my.app")
public class CollectionDataRunner
        implements CommandLineRunner
{

        //private List<String> data;
        private Set<String> data;
        //private String[] data;

        @Override
        public void run(String... args) throws Exception {
                System.out.println(data.getClass().getName());
                System.out.println(this);
```

```
	}

	public Set<String> getData() {
		return data;
	}

	public void setData(Set<String> data) {
		this.data = data;
	}

	@Override
	public String toString() {
		return "CollectionDataRunner [data=" + data + "]";
	}


}
---------------------------------------------
*)Note: Spring or Spring boot recomands us to use interfaces,
  at runtime Impl classes are auto-selected by Spring Container

For  List --> ArrayList,
For  Set  --> LinkedHashSet
For  Map  --> LinkedHashMap

------------------------------------------
*) application.properties

# prefix.variable.mapKey=mapVal
my.app.subjects.ENG=85
my.app.subjects.MAT=95
my.app.subjects.SCI=90

*) Runner class
package in.nareshit.raghu.runner;

import java.util.Map;

import org.springframework.boot.CommandLineRunner;
import
org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

//ctrl+shift+O

@Component
@ConfigurationProperties(prefix = "my.app")
public class CollectionDataRunner
	implements CommandLineRunner
{


	private Map<String,Integer> subjects;

	@Override
	public void run(String... args) throws Exception {
		System.out.println(subjects.getClass().getName());
```

```
                System.out.println(this);
        }

        public Map<String, Integer> getSubjects() {
                return subjects;
        }

        public void setSubjects(Map<String, Integer> subjects) {
                this.subjects = subjects;
        }

        @Override
        public String toString() {
                return "CollectionDataRunner [subjects=" + subjects +
"]";
        }

}
```
===

Java 8:-

Functional Interface : An inteface that contains only one abstract method

*) Adding @FunctionalInterface annotation is optional,
   that indicates to java compiler, -'please check given one is
   Functional Interface or not?'.

--Examples--
#1
```
interface A{  }
```

Ans: NO (Zero abstract methods)

#2
```
interface A {
  void test();
}
```

Ans: Valid Functional Interface

#3
```
interface A {
  void test();
}
interface B extends A{ }
interface C extends B{
   void print();
}
```

ANS: What are valid FI? A,B.
   C is having 2 abstract methods total.

** Including parent interface, count should be one abstract mehod.

```
#4. Valid Functional Interface can have even Object(java.lang)
    class methods syntax  as abstract methods.
    This is optional, that indicatesto sub class please override
    above methods externally.

--Valid one--
@FunctionalInterface
interface Sample {
        void show();
        //indication to sub class to implement this (optional)
        boolean equals(Object ob);
        String toString();
        int hashCode();
}
----------------------------------------------------------------
Lambda Expression: This can be implemented only for Functional
Interface

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

=> This Lamabda Expression is equals to = Writing impl code + creating
object
=> DataTypes are optional inside method params
=> Symbol () is optional , if only one param exist, not for zero.
=> Symbol {} are optional, if only one statement exist.

------Examples--------------------------
interface Message {
  void show();
}

Lambda Exp:
Message m = () -> { sysout("WELCOME TO ALL"); }
--
Message m = () -> sysout("WELCOME TO ALL");   //braces optional(1
stmt)

-Ex#2-------------------
interface  Math {
   int add(int a, int b);
}

Lambda Exp: logic= > return a+b;

Math  mo = (int a, int b) -> { return a+b; }
--
Math  mo = (a,b) -> { return a+b; }   //DataTypes are optional
--
Math  mo = (a,b) -> a+b;   // Do not write return keyword if no braces

Math  mo = (a,b) -> return a+b; //invaild
```

```
--Examplecode#1-----
package in.nareshit.raghu;

interface Math {
	int add(int a,int b);
}

public class Test {

	public static void main(String[] args) {
		Math m = (a,b) ->  a+b;  //impl class + object

		int result = m.add(10, 20); // method call
		System.out.println(result);
	}
}
```
----

*) In realtime, we never define our own functional interfaces.
 All combinations are given by Java only inside package:
    java.util.function

https://docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html

*) We should just compare method params and return type for our loigc suitable one. Do not compare any time interface name or method name.

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

Login to Gmail : Invitation email from admin
type : https://classroom.google.com/h

FQAs: