Tommo no session[02/04/2023]
Next Session :: April 1st week morning=> [tue,wed,thu] (6.30AM to 9.00AM)
Maven[70%],log4j,Gradle[30%,groovy based syntax],Lombok=> recorded videos will be provided within 2weeks time.

List of Annotations
==========
1. @Component
2. @ComponentScan
3. @Service
4. @Repository
5. @Controller
6. @Bean
7. @Qualifier
8. @Primary
9. @Lazy
10. @Scope
11. @PostConstruct
12. @PreDestroy
13. @Autowired
14. @Value
15. @Configuration
16. @Import
17. @ImportResource
18. @PropertySource

There are 2 different ways to perform injection to spring bean properties
a. @Value => It can be used to inject each value to spring bean properties
b. @ConfigurationProperties => It can be used to perform bulk injection.

eg:
input.properties
==========
org.info.companyName = ineuron
org.info.companyLoc = bengaluru
org.info.companyType = IT

using @Value
==========
```
@Component("company)
@PropertySource(location = "in/commons/properties/input.properties")
public class Company{

      @Value("${org.info.companyName}")
      private String name;

      @Value("${org.info.companyLoc}")
      private String adress;

      @Value("${org.info.copmanyType}")
      private String type;
}
```

using @ConfigurationProperties
==========
```
@Component("company)
```

```
@PropertySource("application.properties")
@ConfigurationProperites(prefix= "org.info")
public class Company{
      private String companyName;
      private String companyLoc;
      private String companyType;

      setXXXX(),toString()
}
```

What is the difference b/w @Value and @CofigurationProperties?

@Value

=> It is given by Spring framework,so it can be used in Spring and SpringBoot applications.

=> Support single value injection to Spring bean property.

=> It performs field level injection(setters not required)

=> Common prefix of all keys are not required in application.properties/application.yml file

=> Keys in properties file and property names need not match.

=> If specified key is not present then it would result in "IllegalArgumentException".

@ConfigurationProperties

=> It is given by SpringBoot framework,so it can be used only SpringBoot applications.

=> Support bulk operation

=> It perform setter level injection internally, so setters are mandatory

=> Common prefix of all keys are required in application.properties/application.yml file.

=> keys in properties file and property names should match

=> If the matching key is not found then it would neglect the injection.

Note: While working with @ConfigurationProperties, it is always suggested to add configuarationProcessor inside pom.xml file

```
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-configuration-processor</artifactId>
      <optional>true</optional>
</dependency>
```

Note:

If we try to inject different values to spring bean property using both Fieldlevel(@value) and @ConfigurationProperties annotations, which one will be injection?

Answer:: Since @ConfigurationProperties uses setter injection,so the values injected at field level(@Value) will be overriden with Setter level

application.properties

```
=====================
my.app.name=ineuron
my.app.location=bengaluru
my.app.type=IT
org.ineuron.name=PhysicsWallah
org.ineuron.location=Delhi
org.ineuron.type=EdTech
```

```
@Component(value="comp1")
@ConfigurationProperties(prefix = "org.ineuron")
public class Company1 {
      private String name;
      private String type;
      private String location;
            setter/toString()
}

@Component(value = "comp2")
@ConfigurationProperties(prefix = "my.app")
public class Company2 {
      private String name;
      private String type;
      private String location;
      setter/toString()
}

output
Company [name=PhysicsWallah, type=EdTech, location=Delhi]
Company [name=ineuron, type=IT, location=bengaluru]

Injecting values to different types like Arrays,List,Set,Map,HAS-A Property of
SpringBean using Properties/.yml file
================================================================================
==================================
=> The allowed special characters in properties file is ".","-","[]".
=> To work with Array,List,Set we need to use
prefix.<propertyName>[index]=value  //index should be sequential
=> To work with Map<K,V> we need to use prefix.<propertyName>.<key>=<value>.
                              refer::BootProj04-
BeanInjectionWithCollectionProperties


application.properties
======================
emp.info.id=10
emp.info.name=sachin

#HAS-A property injection
emp.info.company.name=MI
emp.info.company.location=Bandra
emp.info.company.size=35


#Array object injection
emp.info.skill-set[0]=java
emp.info.skill-set[1]=jee
emp.info.skill-set[2]=ORM
emp.info.skill-set[3]=SpringBoot

#List object injection
emp.info.project-names[0]=IND
emp.info.project-names[1]=World1X
emp.info.project-names[2]=Mumbai
emp.info.project-names[3]=Asia1X
```

```
#Set Object injection
emp.info.mobile-numbers[0]=9997778886
emp.info.mobile-numbers[1]=6667778889
emp.info.mobile-numbers[2]=5556667776
emp.info.mobile-numbers[3]=5556667776

#Map object injection
emp.info.id-details.adharNo=7645345
emp.info.id-details.panNo=232345
emp.info.id-details.voterId=2323454

@Component(value = "employee")
@ConfigurationProperties(prefix = "emp.info")
public class Employee {
      private String name;
      private long id;
      private Company company;
      private String[] skillSet;
      private List<String> projectNames;
      private Set<Long> mobileNumbers;
      private Map<String, Object> idDetails;

}

@Component("company")
public class Company {
      private String name;
      private String location;
      private int size;

}

Output
======
Employee[ name=sachin, id=10,
             company=Company [name=MI, location=Bandra, size=35],
             skillSet=[java, jee, ORM, SpringBoot],
             projectNames=[IND, World1X, Mumbai, Asia1X],
             mobileNumbers=[9997778886, 6667778889, 5556667776],
             idDetails={adharNo=7645345, panNo=232345, voterId=2323454}
          ]

YML/YAML Injection
==================
=> It stands for Yet Another MarkUp Language.
=> The extension of the file is .yml or .yaml
=> The biggest limitation of properties file is nodes/level will be repeated in
mulitple keys,especially while
   working with common prefix concepts like collection,HAS-A property to support
bulk injection using @ConfigurationProperties.
=> SpringFramework doesnot support yml file/where as SpringBoot support yml
injection
=> SpringBoot framework internally use snakeyml<ver>.jar for processing the yml
file.

application.properties
======================
emp.info.id=10
emp.info.name=sachin
```

```
emp.info.loc=MI


application.yml
===============
emp:
   info:
      id: 10
      name: sachin
        loc : MI


Rules while writing yml file
============================
 => same nodes/level in the key should not be duplicated
 => replace "." of each node/level with ":" and write new node in next line  with
proper indentation(minimum single space is required)
 => replace "=" symbol with ":" before placing value having minimum single space.
 => To replace Array,List,Set elements use "-".
 => Take Map collection keys and HAS-A property subkeys as the new nodes/levels.
 => use #symbol for Commenting.

application.properties
======================
emp.info.id=10
emp.info.name=sachin

#HAS-A property injection
emp.info.company.name=MI
emp.info.company.location=Bandra
emp.info.company.size=35


#Array object injection
emp.info.skill-set[0]=java
emp.info.skill-set[1]=jee
emp.info.skill-set[2]=ORM
emp.info.skill-set[3]=SpringBoot

#List object injection
emp.info.project-names[0]=IND
emp.info.project-names[1]=World1X
emp.info.project-names[2]=Mumbai
emp.info.project-names[3]=Asia1X

#Set Object injection
emp.info.mobile-numbers[0]=9997778886
emp.info.mobile-numbers[1]=6667778889
emp.info.mobile-numbers[2]=5556667776
emp.info.mobile-numbers[3]=5556667776

#Map object injection
emp.info.id-details.adharNo=7645345
emp.info.id-details.panNo=232345
emp.info.id-details.voterId=2323454


application.yml
===============
```

```
emp:
  info:
    id: 7
    name: dhoni
    company:
      name: iNeuron
      location: Bengaluru
      size: 35
    mobile-numbers:
    - 2223334445
    - 7776665554
    - 5556665554
    skill-set:
    - java
    - jee
    - orm
    - SpringBoot
    project-names:
    - WorldX1
    - IND
    - AsiaX1
    - CSK
    id-details:
      adharNo: 12345
      panNo: 1343556
      voterId: XUCSA12
```

eg#2.

application.properties

```
spring.datasource.url=jdbc:mysql:///octbatch
spring.datasource.username=root
spring.datasource.password=root123
```

application.yml

```
spring:
   datasource:
      url: jdbc:mysql:///enterprisejavabatch
      username: root
      password: root123
```

=> Once we have properties file in eclipse, we can convert into yml using sts supplied plugin.
=> The nodes/level in the keys of properties file/.yml file are not case sensitive.

What is the difference b/w properties file and .yml file?

Properties file

=> no rules and guideliness to develop properties file,just Key=Value
=> it can be used only in java
=> No way related to json format
=> can be used in both Spring and SpringBoot project
=> nodes/level in the keys can be duplicated.
=> it is not a hierarchial data
=> Custom properties file can be injected to bean using @PropertySource
=> While working with profiles in springboot we can't place multiple profiles in single properties file.

=> Spring/SpringBoot directly loads and reads the content of properties file.
=> use properties file when no of keys are minimal and nodes/level in the key are not duplicated.

YML file
========
=> specification/rule and guideliness given by www.yml.org
=> can be used in .java,.ruby,.python etc
=> Super set of JSON
=> Supported only by SpringBoot
=> nodes/level in the keys can't be duplicated.
=> Its a hierarchial data
=> Custom files will be configured using @PropertySource and specifying PropertySource class is required.
=> we can place multiple profiles in single yml file having seperation with "--".
=> every yml file will be converted to property files before loading.
=> use yml file when no of keys are more and nodes/level in the key are repeating.

Realtime DI using application.yml to injection HikariDataSource object in DAO layer
==
refer:: BootProj06-RealTimeDIUsingYML

application.yml
===============
```
spring:
  datasource:
    password: root123
    url: jdbc:mysql:///enterprisejavabatch
    username: root
```

```
package in.ineuron.comp;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import javax.sql.DataSource;

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

import in.ineuron.dto.Employee;

@Repository
public class EmployeeDaoImpl implements IEmployeeDAO {

      private static final String SQL_SELECT_QUERY = "select eid,ename,eage,eaddress from employee";

      @Autowired
      private DataSource dataSource;

      @Override
```

```
	public List<Employee> findAllEmployees() throws Exception {
		System.out.println("DataSource Connection is :: " +
dataSource.getClass().getName());

		List<Employee> empList = new ArrayList<Employee>();
		try (Connection connection = dataSource.getConnection();
				PreparedStatement pstmt =
connection.prepareStatement(SQL_SELECT_QUERY);
				ResultSet resultSet = pstmt.executeQuery()) {

			while (resultSet.next()) {
				Employee employee = new Employee();
				employee.setEid(resultSet.getInt(1));
				employee.setEname(resultSet.getString(2));
				employee.setEage(resultSet.getInt(3));
				employee.setEaddress(resultSet.getString(4));

				empList.add(employee);
			}

		} catch (SQLException se) {
			se.printStackTrace();
			throw se;
		} catch (Exception e) {
			e.printStackTrace();
			throw e;
		}
		return empList;
	}
}

pom.xml
=======
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
<dependency>
	<groupId>com.mysql</groupId>
	<artifactId>mysql-connector-j</artifactId>
	<scope>runtime</scope>
</dependency>
```