Lombok API

Java bean

========

=>Java bean is a java class that is developed by following some standards

a) class must be public

b) Recomanded to implement Serializable

c) Bean properties(member variables) should be private and non-static

d) Every Bean property should have 1 set of public getter , setter methods

e) should 0-param constructor directly or indirectly

3 types of Java Beans

==========

VO class (Value Object class) ----> To hold inputs or outputs

DTO class (Data Transfer Object class) --> To carray data from one layer to another layer or from one project to another Project

BO class/Entity class /Model class --------> To hold persistable /Persistent data

Well-Designed java class should contain

a) overloaded constructors

b) toString()

c) equals(-) method

d) hashCode() method

e) getters & setters (optional)

=>if equals() and hashcode() are not there in java class keeping objects of that java class becomes very problematic as the elements of collections.

=> if toString() method is not there.. we can not print object data in single shot as String using S.o.p(-) statement..

=> With out setters && getters setting data and reading to/from Propertiies is very complex...

Wth Lombok API

==========

=> All the above things will be taken care and will be generated internally..

=>Lombok API also called as Project Lombok generates following boilerplate of java calss

a) Constructors

b) getters && setters

c) toString

d)equals()

e) hashCode and etc..

=>It is an open source api...

=>It must be cfg with IDEs to make IDEs using lombok api to genernate the common Boilerplate ..

=> It supplies bunch of annotations for generating this common code..

@Setter

@Getter

@AllArgsConstructor

@NoArgsConstructor

@RequiredArgsConstructor

@ToString

@EqualsAndHashCode

@Data (It is mix of multiple annotations)

and etc..

steps to configure Lombok with Elcipse /STS IDE

==========

step1) Download lombok-<ver>.jar file from mvnrepository.com

step2) make sure Eclipse /STS IDE installed ...

To donwload STS :: https://spring.io/tools

|-->gives the jar file ---> run it (double click) ---> gives

extracted folder ---> use SpringToolSuite4.exe to launch IDE

step3) create PRoject In Eclipse or STS IDE by adding lombok-<ver>.jar file to the build path..

step4) Launch lombok App by clicking on lombok-<ver>.jar file and select Eclipse or STS IDE instalation folder... ==> click on install/Update button ===>Quit Installer..

step5) Restart Eclipse /STS IDE...

Annotation Type Getter

```
@Target({FIELD,TYPE})
@Retention(SOURCE)
public @interface Getter
```

Put on any field to make lombok build a standard getter. gives instruction to compiler and the .class file generated by the compiler has the result of these annotations

note:: lombok api annotations makes java compiler to generate certain code dynamically in the .class file...

note:: Java compiler is having ability to add code dynamically to .class file thoguh instructions are not there in .java like default cosstructor generation, making java.lang.Object as default super class and etc..

note:: Lombok annotations Retention level is Source i.e these annotations will not be recorded to .class files.. but becoz of lombok annotations instructions the code generated by javac compiler like setters, getters , toString() and etc.. will be recorded into .class file.

@Getter ,@Setter

==========

==> if these are applied at class level .. generate setters and getters for all fields/ properties

==> if these are applied at fields level .. generate setters and getters for specif fields/ properties

What is the use toString()?

=> it useful to display object data in the for single String format..

=> if do not override this method in our class then java.lang.Object class toString() executes and this method gives <fullyqualied Class name>@<hexa decimal notation of hashCode>

@ToString annotation can be used only on class level but not field level.

The following is the toString() method of Object class

```
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

if call S.o.p() with any ref varible.. on that toString() will be called automatically...

Q) What happends if we place toString() explicitly in our class along with lombok api @ToString ?

Ans) @ToString of lombok api will not give instruction javac to generate toString() becoz same method is already available in .java file.. (warning will come for @ToString)

If you use the setter/getter method explicitly and also use lombok annotation then lombok annotation will not work, explicit method will work.

@EqualsAndHashCode

=>Generates equals() method and hashCode() method in the .class file dynamically by giving instruction to java compiler (javac)..

What is hashCode?

=>hashCode is unique identity number given by jvm by every object... and we can get it by calling hashCode() method..

```
System.out.println(c1.hashCode()+" "+c2.hashCode());
```

Can two objects have same hashCode?

Ans) if hashCode() method java.lang.Object class is called then not possible.. becoz it generates hashCode based hashing algorithm as unique number... if we override hashCode() method our code .. generally it generates the hashCode based on state of the object.. of two objects are having same state.. then then we get same hashCode for both objects..

Can one object have two hashcodes?

=> if u overide hashCode() method then based on state of the object one hashCode will be generated.. but internally jvm maintains another hashCode based on hashing algorithm

What is the use equals(-) method?

ans) It is given to compare that state of two objects..
It internally uses hashcode() support also ...
if equals() is overriden in our class then it will compare the state of the two objects.. otherwise Object class equals() method executes which always compares references by internally using == operator...

equals() method of java.lang.Object class

```
public boolean equals(Object obj) {
    return (this == obj);
}
```

what is the difference == and equals() method?

=>Both will compare references if equals() method is not overridden the our class.. if overridden equals(-) method compares state of objects .. and == operator checks the references...

@NoArgsConstructor --> Generates 0-param constructor

@AllArgsConstructor --> Generates parameterized Constructor having all properties/fileds as params
if class is not having any properties/fields then it generates 0-param constructor..

=>Both are applicable at class level (type)

note:Only compiler generated 0-param constructor is called default constructor...

if we place 0-param constructor explicitly or if lomok generates the 0-param constrctor then that should not be called default constructor

int a; default value 0 is

int a=0; initial value is 0

we

Q) can perform constructor overloading and constructor overriding?

Ans) constructor overloading is possible ,
but constructor overriding is not possible becoz in sub class constructor names changes to sub class name..

```
public  class Test{
  public Test(int x){  this(x,y)}
  public  Test(int x ,int y){  this (x) }
  public  Test(){ this(x) }
 }
```

Gives error .. becoz cosntructor chaining is leading infinite loop.. To break it in any consturctor call super()

@RequiredArgsCosntructor

=> Allows to generates parameterized constrictor involving our choice no.of properties/fields..
The properties that u do not want involve should be annotated with @NonNull annotation.

=> if no properties are annotated with @NonNull then it will give 0-param constructor

```
@RequiredArgsConstructor
public class Customer {
      @NonNull
      private int cno;
      @NonNull
      private String cname;
      @NonNull
      private  String cadd;
      private  double billAmt;
}
```

Generated code

```
public class Customer {
      private int cno;
      private String cname;
      private  String cadd;
      private  double billAmt;
    public   Customer(int cno,String cname, String cadd){
          ....
    }
}
```

```
@RequiredArgsConstructor
public class Customer {
      private int cno;
      private String cname;
      private  String cadd;
      private  double billAmt;
}
```

```
public class Customer {
      private int cno;
      private String cname;
      private  String cadd;
      private  double billAmt;
    public  Customer(){ }
}
```

```
@RequiredArgsConstructor
@NoArgsCosntructor
public class Customer {
        private int cno;
        private String cname;
        private  String cadd;
        private  double billAmt;
   }
```

Generated code

```
public class Customer {
        private int cno;
        private String cname;
        private  String cadd;
        private  double billAmt;
      public  Customer(){ }
      public  Customer(){ }
}
```

duplicate constructors (error)

we can take constructor as private ,protected and public .. To get them through lombok api we can use "access" attribute of @XxxArgsCosntrctor annotations as shown below

```
@RequiredArgsConstructor(access = AccessLevel.PRIVATE)
@AllArgsConstructor(access = AccessLevel.PROTECTED)
@NoArgsConstructor(access = AccessLevel.PUBLIC)
public class Customer {            enum
      @NonNull
      private int cno;
      @NonNull
      private String cname;
      @NonNull
      private  String cadd;
      private  double billAmt;

}
```

note:: Using Lombok api

->we can not generate constrictor with var args

->we can not generate multiple our choice parameterized constructors at time like with 1 param , with 2 params wtih 3 params at a time..

@Data ====== It is combination of @Getter + @Setter +@EqualsAndHashCode +@ToString + @ RequiredArgsConstructor

```
@Data
public class Customer {
      @NonNull
      private int cno;
      private String cname;
      private  String cadd;
      private  double billAmt;

}
```

Generated code

```
public class Customer {
      private int cno;
      private String cname;
      private  String cadd;
      private  double billAmt;
     public Customer(int cno){ this.cno=cno;}
      //toString()
      //4 setters & 4 getters
      //equals(-) ,canEquals(-)
      //hashCode()
}
```

@Data will generate RequiredArgsConstructor only when AllArgsConstructor or NoArgsConstructor annotations are not there. If AllArgsConstructor or NoArgsconstructor is/are there then you have to place @RequiredArgsConstructor