Lombok API

Java Bean – java bean is a java class that is developed by following some standards.

I. Class must be public.
II. Recommended to implement Serializable interface.
III. Bean properties (member variables) should be private and non-static.
IV. Every bean property should have 1 set of public getter, setter methods.
V. Should have 0-param constructor directly or indirectly.

Three types of Java beans: -

I. VO class (Value Object class) → to hold inputs or outputs.
II. DTO class (Data transfer Object class) → to carry data from one layer to another layer or from one project to another project.
III. BO class/Entity class/Model class → to hold persistable/persistent data.

Well-designed java class should contain:
==

i. Overloaded constructors
ii. toString() method
iii. equals() method
iv. hashCode() method
v. getters and setters method (optional)

=> if equals() and hashCode() are not there in java class, then keeping objects of that java class becomes very problematic as the elements of collections.
=> if toString() method is not there.. then we cannot print object data in single shot as string using System.out.println() statement.
=> Without setters and getters setting data and reading to/from properties is very complex.

Before Lombok API: -

We should manually make java classes as well-designed classes by adding the above said methods and we should manually increase or decrease getter and setter methods based on no. of properties we are adding and removing.

With Lombok API: -

=> All the above things will be taken care and will be generated internally.
=> Lombok API also called as Lombok project generates following boilerplate code of java.

i. Constructors
ii. Getters and Setters
iii. toString() method
iv. equals() method
v. hashCode() method and etc..

=> it is an open source api.
=> it must be configured with IDEs to make IDEs using Lombok api to generate the common boilerplate.
=> it supplies bunch of annotations for generating this common code.

i. @Setter
ii. @Getter
iii. @AllArgsConstructor
iv. @NoArgsConstructor
v. @RequiredArgsConstructor
vi. @ToString
vii. @EqualsAndHashCode
viii. @Data

Steps to configure Lombok with eclipse/STS IDE: -

Step 1). Download Lombok-<ver>.jar file from mvnrepository.com
Step 2). Make sure eclipse/sts IDE installed.
Step 3). Create project in eclipse or sts IDE by adding Lombok-<ver>.jar file to build path.
Step 4). Launch Lombok app by clicking on Lombok-<ver>.jar file and select eclipse or sts IDE installation folder → click on install/update button → Quit installer.
Step 5). Restart eclipse/sts IDE.
Step 6). Add one java bean class to project of eclipse/sts IDE by Lombok api annotation and observe whether code is generated or not.

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 though instructions are not there in .java file such as default constructor generation and 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 because 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 to class level… compiler generates the setters and getters for all fields/properties.
=> if these are applied to fields level… compiler generates setters and getters for specific fields/properties.

Example1
===========

```
 Customer.java
 =============
 @Setter
 @Getter
 public class Customer {
        private int cno;
        private String cname;
        private  String cadd;
        private  double billAmt;
        private long billNo;

 }
```

code in Customer.class
==================

```
   public class Customer {
          private int cno;
          private String cname;
          private  String cadd;
          private  double billAmt;
          private long billNo;
           public   Customer() { ….} //default constructor

        //5setter methods
          ….
        //5 getter methods
          …
        }
```

Customer.java

```
public class Customer {
        private int cno;
       @Setter
       @Getter
       private String cname;
       @Setter
       @Getter
        private  String cadd;
        private  double billAmt;
        private long billNo;

}
```

Code in Customer.class
=====================

```
public class Customer {
        private int cno;
       private String cname;
        private  String cadd;
        private  double billAmt;
        private long billNo;
       //2 setter methods for  cname,cadd
          …
        //2 getter methods for cname,cadd
         …
        public  Customer(){  }

}
```

@ToString
==========
=> makes the javac compiler to generate/override toString() method having logic to display object data… this is applicable only on top of class.

```
Customer.java
==============
@ToString
public class Customer {
       private int cno;
       private String cname;
       private  String cadd;
       private  double billAmt;

}
```

```
Code in   Customer.class
============
public class Customer {
       private int cno;
       private String cname;
       private  String cadd;
       private  double billAmt;

      @Override
            public String toString() {
                  return "Customer [cno=" + cno + ", cname=" + cname + ", cadd=" + cadd + ", billAmt=" + billAmt + "]";
            }

    }
```

What is the use of toString()?
=> it is useful to display object data in single string format.
=> if do not override this method in our class then java.lang.Object class toString() executes and this method gives <fullyqualified class name>@<hexadecimal notation of hashcode>

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

If call System.out.println() with any reference variable… on that toString() will be called automatically.

Can we customize the logics generated by Lombok api?
Not possible

What happens of we place toString() explicitly in our class along with Lombok api @ToString?
@ToString of Lombok api will not give instruction to javac to generate toString() because same method is already available in .java file… (warning will come for @ToString). → Warning: Not generating toString(): A method with that name already exists

      Note::same is applicable for @Getter and @Setter annotations.

@EqualsAndHashCode
==================
=> Generates equals() method and hashCode() method in the .class file dynamically by giving instruction to java compiler javac.
=> Applicable at class level (type).

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?
=> if hashCode() method of java.lang.Object class is called then not possible because it generates hashcode based on hashing algorithm. As unique number, If we override hashCode() method with our code… generally it generates the hashcode based on state of object, if two objects are having same state.. then we get same hashcode for both objects.

```
	Customer c1 = new Customer(101, “raja”, “hyd”, 677.88f);
	System.out.println(c1.hashCode()+” “+c2.hashCode()); //prints same hashCode
```

Can one object have two hashcodes?

=> if you override hashCode() method then based on state of the object one hashcode will be generated but internally JVM maintains another hashcode based on hashing algo.

```
	Customer c1 = new Customer(101, “raja”, “hyd”, 677.88f);
	Customer c1 = new Customer(101, “raja”, “hyd”, 677.88f);
	System.out.println(c1.hashCode()+” “+c2.hashCode()); //our hashcodes
	System.out.println(System.IdentityHashCode(c1));
	System.out.println(System.IdentityHashCode(c1));	// jvm hashcodes
```

What is the use of equals() method?

Ans.	It is given to compare the state of two objects.
	It internally uses hashCode() support also.
	If equals() is overridden 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 between == and equal() method?

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

```
	Customer c1 = new Customer(101, “raja”, “hyd”, 677.88f);
	Customer c1 = new Customer(101, “raja”, “hyd”, 677.88f);
#1	System.out.println(c1.equals(c2));
#2	System.out.println(c1 == c2);
		#1 → false,
		#2 → false if equals() not overridden in customer class otherwise
		#1 → true,
		#2 → false
```

```
@Setter
@Getter
@ToString
@EqualsAndHashCode
@AllArgsConstructor
public class Customer {
	private int cno;
	private String cname;
	private  String cadd;
	private  double billAmt;
}
```

Generated code in .class file
======================

```
public class Customer {
	private int cno;
	private String cname;
	private  String cadd;
	private  double billAmt;
	//4 param constructor
	// 4 setters
	// 4 getters
	// equals(-)  and canEquals(-)   → same as equals(-) but is protected method
	//toString ()
	//hashCode()
}
```

@NoArgsConstructor → Generates 0-param constructor
@AllArgsConstructor → Generates parameterized constructor having all properties/fields as parameter, if class is not having any properties/fields then it generates 0-param constructor.

→ Both are applicable at class level (type).

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

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

```
@AllArgsConstructor
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(int cno,String cname,String cadd,double billAmt){
       .....
      }
}
```

```
@AllArgsConstructor
public class Customer {

}
```

```
//Generated code
public class Customer {
      public  Customer(){  }
}
```

```
@NoArgsConstructor
@AllArgsConstructor
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(int cno,String cname,String cadd,double billAmt){
       .....
      }
    public  Customer(){ }

}
```

```
@NoArgsConstructor
@AllArgsConstructor
public class Customer {

}
```

```
Generated code
================
@NoArgsConstructor
@AllArgsConstructor
public class Customer {
    public      Customer(){ }
    public      Customer(){ }
}
```

Error will be generated...

Note::Only compiler generated 0-param constructor is called default constructor. If we place 0-param constructor explicitly or if Lombok generates the 0-param constructor then that should not be called default constructor.

Can we perform constructor overloading and constructor overriding?
Constructor overloading is possible but constructor overriding is not possible because in sub class constructor names changes to sub class name.

```
@NoArgsCosntructor
public class  Customer{

   public  Customer(){
    }
}
```

```
Generated code
===============
 public class Customer{

    public Customer(){ }    // duplicate constucTors (error)
    public Customer(){ }
 }
```

```
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()

this, this()
super, super() ,

@RequiredArgsConstructor: -
→ allows to generates parameterized constructor involving our choice of number of properties/fields. The properties that you want to 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)

```
@RequiredArgsConstructor
@AllArgsConstructor
@NoArgsConstructor
public class Customer {
        @NonNull
        private int cno;
        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(){  }
            public  Customer(int cno,String cname,String cadd,double billAmt){
             .....
             }
       public Customer(int cno,String cadd . ·){
            ..
          }
 }
```

We can take constructor as private, protected and public.. to get them through Lombok api we can use "access" attribute of @XxxArgsConstructor 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 a 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()
}
```

Note:: -

@RequiredArgsConstructor of @Data works only when @AllArgsConstructor, @NoArgsConstructor is placed on the top of class.. if you still need the effect of @RequiredArgsConstructor then place these explicitly.

```
@Data
@AllArgsCosntructor
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;
       //toString()
      //4 setters & 4 getters
       //equals(-) ,canEquals(-)
       //hashCode()

       public Customer(int cno,String cname,String cadd,double billAmt){
         }
 }
```

```
@NoArgsConstructor
@AllArgsConstructor
@Data
@RequiredArgsConstructor
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;
       //toString()
      //4 setters & 4 getters
       //equals(-) ,canEquals(-)
       //hashCode()

       public Customer(int cno,String cname,String cadd,double billAmt){
         }
      public Customer(){  }
      public  Customer(int cno){   ..}

     }
```