



# INDEX

## Lombok API

---

- |   |                           |
|---|---------------------------|
| 1. Introduction                                     | <a href="#"><u>01</u></a> |
| 2. Steps to configure Lombok with Eclipse [STS IDE] | <a href="#"><u>05</u></a> |
| 3. Annotations                                      | <a href="#"><u>07</u></a> |

# Lombok API

## Introduction

### Java bean

- ✚ Java bean is a java class that is developed by following some standards
  - a. Class must be public
  - b. Recommended to implement Serializable
  - c. Bean properties (member variables) should be private and non-static
  - d. Every bean property should have 1 set of public setter & getter 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 carry 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)

### Note:

- ✓ if equals() and hashCode() methods 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.
- ✓ Without setters & 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 number 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 Project Lombok, generates the following boilerplate code of java class
  - Constructors
  - getters & setters
  - toString
  - equals
  - hashCode 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 code.
- 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 Eclipse [STS IDE]

**Step 1:** Download Lombok [\[Link\]](#) or you can download from mvnrepository.com also [\[Link\]](#).

**Step 2:** Make sure Eclipse/ STS IDE installed.

**Step 3:** Create project in Eclipse or STS IDE by adding lombok-<ver>.jar file to the build path.

**Note:** If you will use the lombok annotation now then you will get error because lombok is not configured with your IDE.

**Step 4:** To configure launch Lombok API by double clicking on lombok-<ver>.jar then it will scan and detect your IDEs exe files other wise you can select your IDEs path by click on the Specify location... button then choose the IDEs and

click on Install/ Update.



**Step 5:** Now you will get Install successful message then click on the Quit Installer button.



**Step 6:** Restart Eclipse /STS IDE.

**Step 7:** Add one Java bean class to project of Eclipse/ STS IDE by having lombok

API annotation and observe whether code is generated or not?

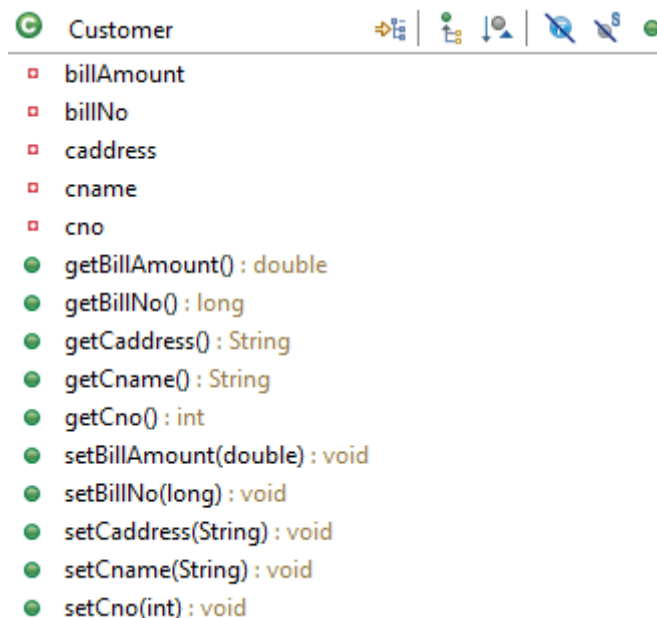
### Customer.java

```
package com.sahu.model;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
public class Customer {
    private int cno;
    private String cname;
    private String caddress;
    private double billAmount;
    private long billNo;
}
```

If you got to Type hierarchy (FN + F4/ F4) you can see the methods



### Note:

- ✓ Lombok API annotations makes java compiler to generate certain code dynamically in the .class file.
- ✓ Java compiler is having ability to add code dynamically to .class file though instructions are not there in .java file like default constructor

generation, making java.lang.Object as default super class and etc.

- ✓ Lombok API 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.

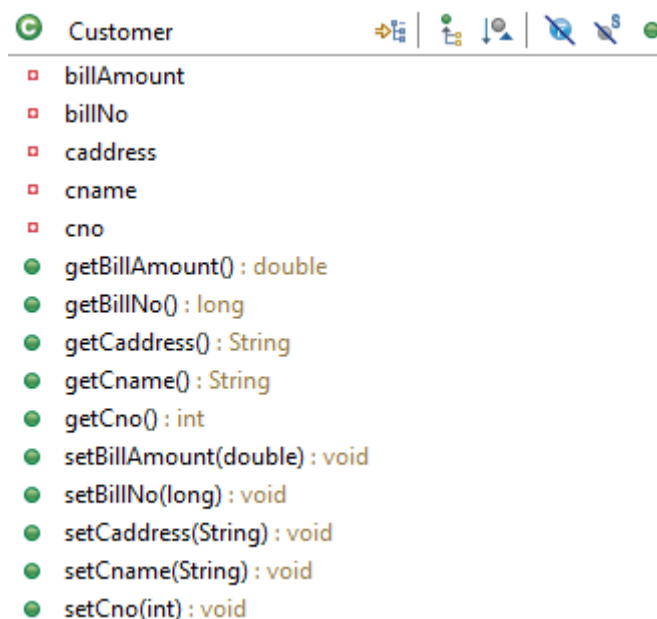
## Annotations

### @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 specific fields/ properties.

### Customer.java

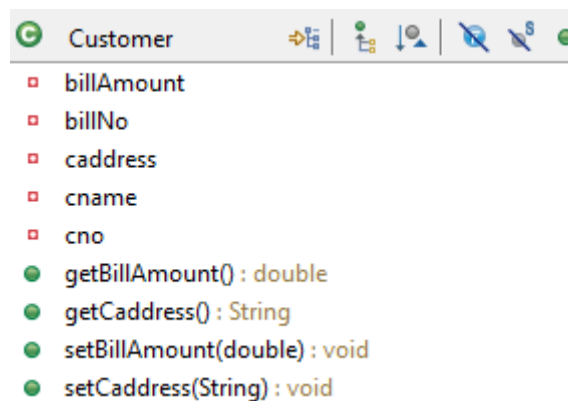
```
@Setter
@Getter
public class Customer {
    private int cno;
    private String cname;
    private String caddress;
    private double billAmount;
    private long billNo;
}
```





### Customer.java

```
public class Customer {  
    private int cno;  
    private String cname;  
    @Setter  
    @Getter  
    private String caddress;  
    @Setter  
    @Getter  
    private double billAmount;  
    private long billNo;  
}
```



### @ToString

- Makes the javac compiler to generate/ override toString() having logic to display object data. It is applicable only top of class.

### Q. What is the use toString()?

Ans.

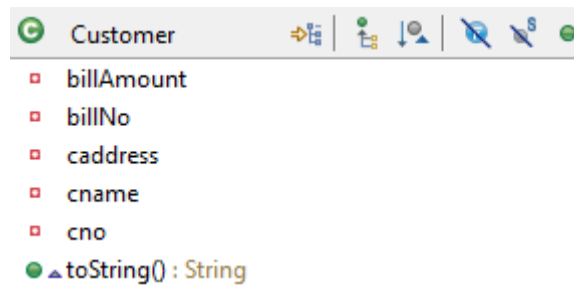
- It is useful to display object data in the form of single String format.
- If we do not override this method in our class then java.lang.Object class toString() method executes and this method gives <fully qualified class name>@<hexa-decimal notation of hashCode>

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

If we call S.o.p() with any reference variable on that toString() will be called automatically.

### Customer.java

```
@ToString
public class Customer {
    private int cno;
    private String cname;
    private String caddress;
    private double billAmount;
    private long billNo;
}
```



**Q. Can we customize the logics generated by Lombok API?**

**Ans.** Not possible

**Q. What happens if we place toString() explicitly in our class along with Lombok API @ToString?**

**Ans.** @ToString of Lombok API will not give instruction to javac compiler to generate toString() because same method is already available in .java file. (warning will come for @ToString annotation like "Not generating toString(): A method with that name already exists").

**Note:** Same is applicable for @Getter and @Setter annotations.

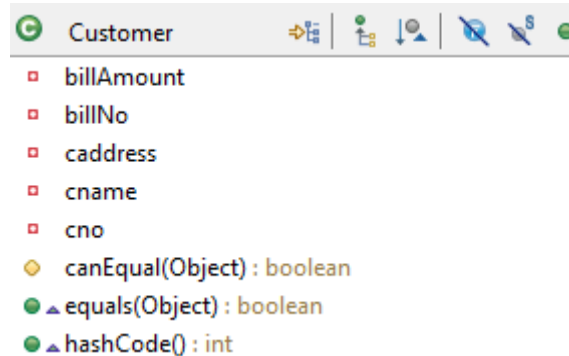
### @EqualsAndHashCode

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

### Customer.java

```
@EqualsAndHashCode
public class Customer {
    private int cno;
    private String cname;
```

```
private String address;  
private double billAmount;  
private long billNo;  
}
```



**Q. What is hashCode()?**

**Ans.** hashCode is unique identity number given by JVM by every object and we can get it by calling hashCode() method.

E.g.,

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

**Q. Can two objects have same hashCode()?**

**Ans.**

- If hashCode() method java.lang.Object class is called then not possible because 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 we get same hashCode for both objects.

○ E.g.,

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

**Q. Can one object have two hashCode?**

**Ans.** 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 algorithm.

E.g.,

```
Customer c1 = new Customer(101,"raja","hyd",677.88);
```

```
Customer c2 = new Customer(101,"raja","hyd",677.88);
System.out.println(c1.hashCode()+" "+c2.hashCode()); //our hashCode

System.out.println(System.identityHashCode(c1)); //JVM hashCode
System.out.println(System.identityHashCode(c2));
```

**Q. What is the use of equals(-) method?**

**Ans.**

- It is given to compare that 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);
}
```

**Q. what is the difference between == and equals() method.**

**Ans.**

- 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.

```
Customer c1 = new Customer(101,"raja","hyd",677.88);
Customer c2 = new Customer(101,"raja","hyd",677.88);
System.out.println(c1.equals(c2));
System.out.println(c1==c2);
```

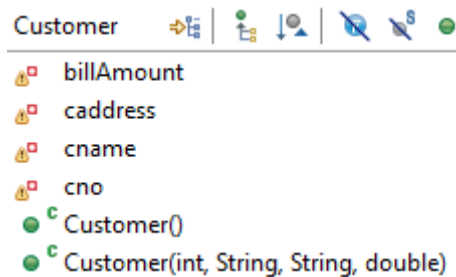
Answers are #1. false & #2. False if equals(-) method is not overridden in Customer class otherwise answers are #1. true & #2. false.

**@NoArgsConstructor, @AllArgsConstructor**

- @NoArgsConstructor Generates 0-param constructor.
- @AllArgsConstructor Generates parameterized having all properties/ fields as params, if class is not having any properties/ fields then it generates 0-param constructor.
- Both are applicable at class level (type).

## Customer.java

```
@AllArgsConstructor
@NoArgsConstructor
public class Customer {
    private int cno;
    private String cname;
    private String caddress;
    private double billAmount;
}
```



### 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

### Q. Can we perform constructor overloading and constructor overriding?

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

### F&Q:

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

Gives error because constructor chaining is leading infinite loop. To break it write in any constructor call super().

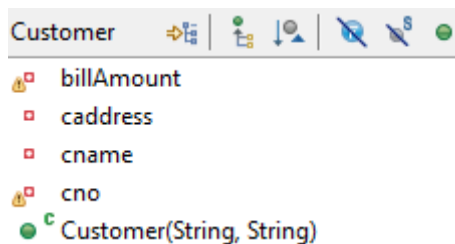
### @RequiredArgsConstructor

- Allows to generate parameterized constructor involving our choice 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.

### Customer.java

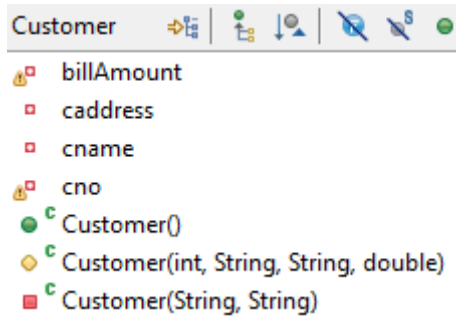
```
@RequiredArgsConstructor
public class Customer {
    private int cno;
    @NonNull
    private String cname;
    @NonNull
    private String caddress;
    private double billAmount;
}
```



- ✚ 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

### Customer.java

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

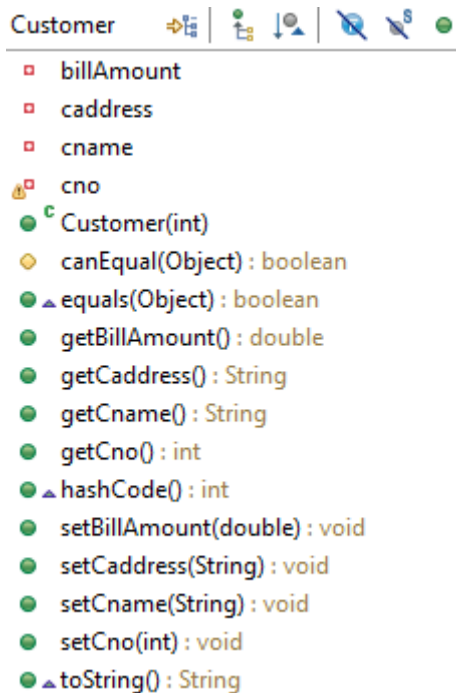


## @Data

- It is combination of @Getter + @Setter + @EqualsAndHashCode + @ToString + @RequiredArgsConstructor

## Customer.java

```
@Data
public class Customer {
    @NotNull
    private int cno;
    private String cname;
    private String caddress;
    private double billAmount;
}
```

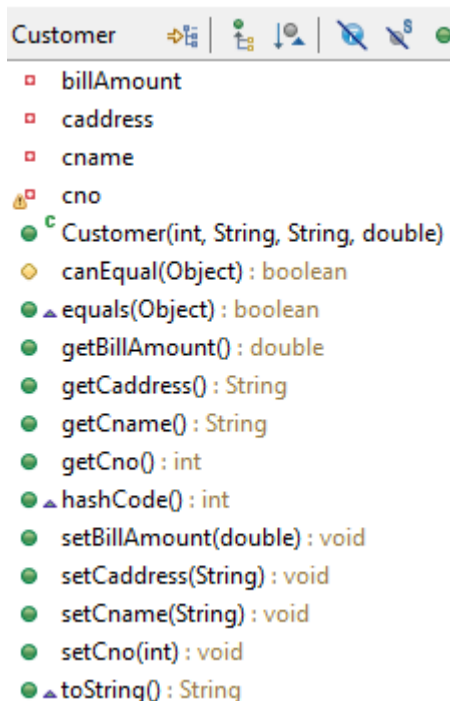


**Note:** Using Lombok API

- ✓ We cannot generate constructor with var args.
- ✓ We cannot generate multiple our choice parameterized constructors at time like with 1 param, with 2 params, with 3 params at a time.
- @RequiredArgsConstructor of @Data works only when @AllArgsConstructor, @NoArgsConstructor is placed on the top of the class if you still need the effect of @RequiredArgsConstructor then place it explicitly.

### Customer.java

```
@Data
@AllArgsConstructor
public class Customer {
    @NonNull
    private int cno;
    private String cname;
    private String caddress;
    private double billAmount;
}
```



The screenshot shows the 'Customer' class in an IDE. The class has four private fields: `billAmount` (double), `caddress` (String), `cname` (String), and `cno` (int). The `cno` field is annotated with `@NonNull`. The class has a constructor `Customer(int, String, String, double)` and several methods: `canEqual(Object)` (boolean), `equals(Object)` (boolean), `getBillAmount()` (double), `getCaddress()` (String), `getCname()` (String), `getCno()` (int), `hashCode()` (int), `setBillAmount(double)` (void), `setCaddress(String)` (void), `setCname(String)` (void), `setCno(int)` (void), and `toString()` (String).

----- The END -----