Project Lombok

- It's a Java library, which helps to reduce **boilerplate code** **using annotations.**
- During compilation, it process the annotation and inject code into our Java classes.

Lombok is compatible with Java starting from **Java 6** and supports all later versions.

pom.xml

```xml
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.30</version>
    <scope>provided</scope>
</dependency>
```

Now, I am trying to use Lombok

```
import lombok.val;
public class LombokTest {

    public static void main (String args[]) {
        val temp = "hello";
        System.out.println(temp);
    }
}
```

But IntelliJ is showing compilation error and if I try to run the same code its working totally fine

```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java ...
hello

Process finished with exit code 0
```

How so?

- Its because, Lombok will add the code during compile time. Below is the .class file

LombokTest.class

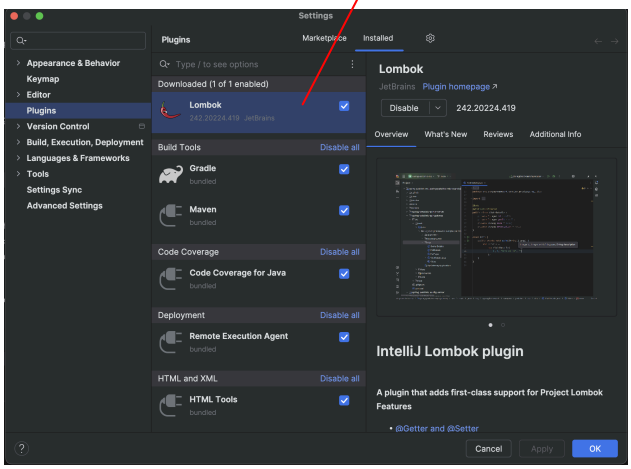
Decompiled .class file, bytecode version: 61.0 (Java 17)

```
/.../

package com.conceptandcoding.LombokTestPackage;

public class LombokTest {
    public LombokTest() {
    }

    public static void main(String[] args) {
        String temp = "hello";
        System.out.println("hello");
    }
}
```

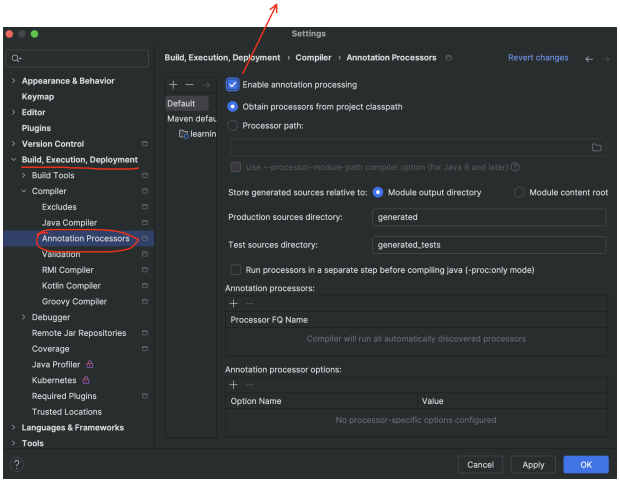
- But our IDE do not see the code which Lombok generates. So its shows the red error, even though compilation works fine.

So, we have to add the Lombok plugin to our IDE and enable annotation processing.
So that our IDE simulates what Lombok generates.

Add this plugin



Enable the annotation processing



Now, IDE do not show any error

```
import lombok.val;
public class LombokTest {

    public static void main (String args[]) {
        val temp = "hello";
        System.out.println(temp);
    }
}
```

Let's see 10 frequently used Lombok features:

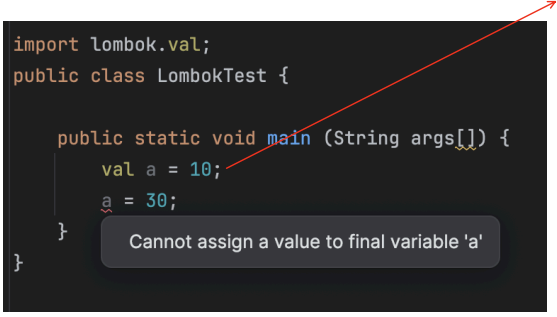
1. val and var

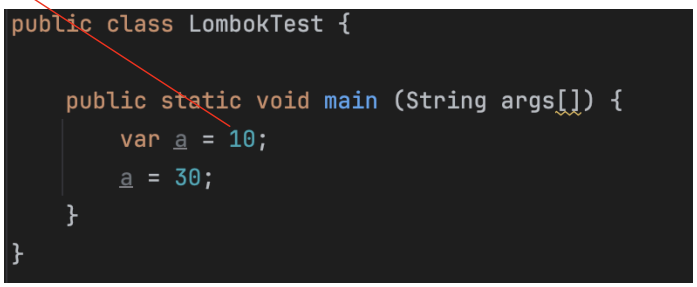
- Instead of actually writing the type, we can use these as the type of **local variable** declaration.
- Type will be inferred from the **initializer expression**.

Note: only works for local variable (not for fields or parameter)

val : marks local variable immutable (variable made final)
var : local variable not marked as final.

Initialization is required, as it inferred the type from that itself.





2. @NonNull

- Generates a null check statement. And throws NPE if value is null.
- Can be used on parameters of a method or constructor.

.java

```
import lombok.NonNull;
public class NonNullExample {

    public void demoMethod(@NonNull String name) {
      System.out.println(name);
    }
}
```

.class

```
import lombok.NonNull;

public class NonNullExample {
    public NonNullExample() {
    }

    public void demoMethod(@NonNull String name) {
        if (name == null) {
            throw new NullPointerException("name is marked non-null but is null");
        } else {
            System.out.println(name);
        }
    }
}
```

3. Getters and Setters

- We can annotate any field with @Getter or @Setter to let Lombok generate the default getter and setter methods.

.java

```java
import lombok.Getter;
import lombok.Setter;

public class TestPojo {

    @Getter @Setter
    String name;

    @Getter @Setter
    boolean committeeMember;
}
```

.class

```java
public class TestPojo {
    String name;
    boolean committeeMember;

    public TestPojo() {
    }

    public String getName() {
        return this.name;
    }

    public void setName(final String name) {
        this.name = name;
    }

    public boolean isCommitteeMember() {
        return this.committeeMember;
    }

    public void setCommitteeMember(final boolean committeeMember) {
        this.committeeMember = committeeMember;
    }
}
```

- By default, generated getters/setters method are Public, but we can also control it.

.java

```java
import lombok.AccessLevel;
import lombok.Getter;
import lombok.Setter;

public class TestPojo {

    @Getter(AccessLevel.PRIVATE) @Setter(AccessLevel.PROTECTED)
    String name;

    @Getter @Setter
    boolean committeeMember;
}
```

.class

```java
public class TestPojo {
    String name;
    boolean committeeMember;

    public TestPojo() {
    }

    private String getName() {
        return this.name;
    }

    protected void setName(final String name) {
        this.name = name;
    }

    public boolean isCommitteeMember() {
        return this.committeeMember;
    }

    public void setCommitteeMember(final boolean committeeMember) {
        this.committeeMember = committeeMember;
    }
}
```

- We can also use this Annotations at class level too, then:

- @Getter annotation will be applied to all non-static fields.
- And @Setter method annotation is applied to all non-static and non-final fields.

.java

```java
import lombok.Getter;
import lombok.Setter;

@Getter @Setter
public class TestPojo {


    String name;
    boolean committeeMember;
    static int maxTime = 100;
}
```

.class

```java
public class TestPojo {
    String name;
    boolean committeeMember;
    static int maxTime = 100;

    public TestPojo() {
    }

    public String getName() {
        return this.name;
    }

    public boolean isCommitteeMember() {
        return this.committeeMember;
    }

    public void setName(final String name) {
        this.name = name;
    }

    public void setCommitteeMember(final boolean committeeMember) {
        this.committeeMember = committeeMember;
    }
}
```

- While using annotation at class level, if we want to skip the default generation for any field, we can do that by using AccessLevel.NONE

.java

```java
import lombok.AccessLevel;
import lombok.Getter;
import lombok.Setter;

@Getter @Setter
public class TestPojo {

    @Setter(AccessLevel.NONE)
    String name;

    boolean committeeMember;
}
```

.class

```java
public class TestPojo {
    String name;
    boolean committeeMember;

    public TestPojo() {
    }

    public String getName() {
        return this.name;
    }

    public boolean isCommitteeMember() {
        return this.committeeMember;
    }

    public void setCommitteeMember(final boolean committeeMember) {
        this.committeeMember = committeeMember;
    }
}
```

4. @ToString

- Used to generate "toString()" method.
- Class name followed by parentheses containing fields (non-static) separated by commas.

.java **.class**

```java
@ToString
public class TestPojo {

    String name;

    boolean committeeMember;
}
```



```java
public class TestPojo {
    String name;
    boolean committeeMember;

    public TestPojo() {
    }

    public String toString() {
        return "TestPojo(name=" + this.name + ", committeeMember=" + this.committeeMember + ")";
    }
}
```

Excluding one specific field

.java

```java
@ToString
public class TestPojo {

    String name;

    @ToString.Exclude
    boolean committeeMember;
}
```


.class

```java
public class TestPojo {
    String name;
    boolean committeeMember;

    public TestPojo() {
    }

    public String toString() {
        return "TestPojo(name=" + this.name + ")";
    }
}
```

Skipping field name

.java

```java
@ToString(includeFieldNames = false)
public class TestPojo {

    String name;
    boolean committeeMember;
}
```


.class

```java
public class TestPojo {
    String name;
    boolean committeeMember;

    public TestPojo() {
    }

    public String toString() {
        return "TestPojo(" + this.name + ", " + this.committeeMember + ")";
    }
}
```

Explicitly selecting the fields

.java

```java
@ToString(onlyExplicitlyIncluded = true)
public class TestPojo {

    String name;

    @ToString.Include
    boolean committeeMember;
}
```


.class

```java
public class TestPojo {
    String name;
    boolean committeeMember;

    public TestPojo() {
    }

    public String toString() {
        return "TestPojo(committeeMember=" + this.committeeMember + ")";
    }
}
```

5. @NoArgsConstructor, @RequiredArgsConstructor, @AllArgsConstructor

@NoArgsConstructor: generates no-arg constructor
@AllArgsConstructor: generates constructor with all fields
@RequiredArgsConstructor: generates constructor with only final and @NonNull fields

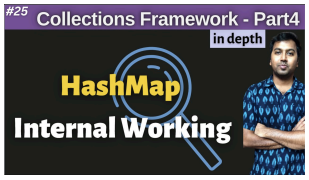
.java → **.class**

```java
@NoArgsConstructor
@AllArgsConstructor
@RequiredArgsConstructor
public class TestPojo {

    String name;
    boolean committeeMember;
    @NonNull Integer age;
}
```

```java
public class TestPojo {
    String name;
    boolean committeeMember;
    @NonNull Integer age;

    public TestPojo() {
    }

    public TestPojo(final String name, final boolean committeeMember, final @NonNull Integer age) {
        if (age == null) {
            throw new NullPointerException("age is marked non-null but is null");
        } else {
            this.name = name;
            this.committeeMember = committeeMember;
            this.age = age;
        }
    }

    public TestPojo(final @NonNull Integer age) {
        if (age == null) {
            throw new NullPointerException("age is marked non-null but is null");
        } else {
            this.age = age;
        }
    }
}
```

6. @EqualsAndHashCode

As we have already seen the Equals and HashCode method contract in →



We can use Lombok to generate this code for us.
By default, It uses all non-static and non-transient fields

.java → **.class**

```java
public class TestPojo {
    String name;
```

```java
@EqualsAndHashCode
public class TestPojo {

    String name;

    @EqualsAndHashCode.Exclude
    boolean committeeMember;

    static int maxTerm = 10;
}
```

```java
    boolean committeeMember;
    static int maxTerm = 10;

    public TestPojo() {
    }

    public boolean equals(final Object o) {
        if (o == this) {
            return true;
        } else if (!(o instanceof TestPojo)) {
            return false;
        } else {
            TestPojo other = (TestPojo)o;
            if (!other.canEqual( other: this)) {
                return false;
            } else {
                Object this$name = this.name;
                Object other$name = other.name;
                if (this$name == null) {
                    if (other$name != null) {
                        return false;
                    }
                } else if (!this$name.equals(other$name)) {
                    return false;
                }

                return true;
            }
        }
    }

    protected boolean canEqual(final Object other) {
        return other instanceof TestPojo;
    }

    public int hashCode() {
        int PRIME = true;
        int result = 1;
        Object $name = this.name;
        result = result * 59 + ($name == null ? 43 : $name.hashCode());
        return result;
    }
}
```

7. @Data

Shortcut for
- @ToString
- @EqualsAndHashCode
- @Getter on all fields
- @Setter on all non-final fields
- @RequiredArgsConstructor

.java

```java
@Data
public class TestPojo {

    String name;
    final Integer age;
    @NonNull String address;

}
```

.class

```java
public class TestPojo {
    String name;
    final Integer age;
    @NonNull String address;

    public TestPojo(final Integer age, final @NonNull String address) {
        if (address == null) {
            throw new NullPointerException("address is marked non-null but is null");
        } else {
            this.age = age;
            this.address = address;
        }
    }

    public String getName() {
        return this.name;
    }

    public Integer getAge() {
        return this.age;
    }

    public @NonNull String getAddress() {
        return this.address;
```

```
    }

    public void setName(final String name) {
        this.name = name;
    }

    public void setAddress(final @NonNull String address) {
        if (address == null) {
            throw new NullPointerException("address is marked non-null but is null");
        } else {
            this.address = address;
        }
    }
```

```
    public boolean equals(final Object o) {
        if (o == this) {
            return true;
        } else if (!(o instanceof TestPojo)) {
            return false;
        } else {
            TestPojo other = (TestPojo)o;
            if (!other.canEqual( other: this)) {
                return false;
            } else {
                label47: {
                    Object this$age = this.getAge();
                    Object other$age = other.getAge();
                    if (this$age == null) {
                        if (other$age == null) {
                            break label47;
                        }
                    } else if (this$age.equals(other$age)) {
                        break label47;
                    }

                    return false;
                }

                Object this$name = this.getName();
                Object other$name = other.getName();
                if (this$name == null) {
                    if (other$name != null) {
                        return false;
                    }
                } else if (!this$name.equals(other$name)) {
                    return false;
                }

                Object this$address = this.getAddress();
                Object other$address = other.getAddress();
                if (this$address == null) {
                    if (other$address != null) {
                        return false;
                    }
                } else if (!this$address.equals(other$address)) {
                    return false;
                }

                return true;
            }
        }
    }

    protected boolean canEqual(final Object other) {
        return other instanceof TestPojo;
    }
```

```
    public int hashCode() {
        int PRIME = true;
        int result = 1;
        Object $age = this.getAge();
        result = result * 59 + ($age == null ? 43 : $age.hashCode());
        Object $name = this.getName();
        result = result * 59 + ($name == null ? 43 : $name.hashCode());
        Object $address = this.getAddress();
        result = result * 59 + ($address == null ? 43 : $address.hashCode());
        return result;
    }

    public String toString() {
        String var10000 = this.getName();
        return "TestPojo(name=" + var10000 + ", age=" + this.getAge() + ", address=" + this.getAddress() + ")";
    }
}
```

8. @Value

Immutable version of @Data

- All fields are made "private" and "final".
- Setters are not generated.
- Class itself made final.
- Like @Data, toString, Equal and HashCode method generated.
- @Getter on all fields
- @RequiredArgsConstructor (since all fields are final, constructor with all fields will get generated, so its equivalent to @AllArgsConstructor)

.java

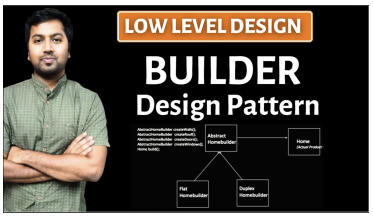
```java
@Value
public class TestPojo {

    String name;
    final Integer age;
    @NonNull String address;

}
```

.class

```java
public final class TestPojo {
    private final String name;
    private final Integer age;
    private final @NonNull String address;

    public TestPojo(final String name, final Integer age, final @NonNull String address) {
        if (address == null) {
            throw new NullPointerException("address is marked non-null but is null");
        } else {
            this.name = name;
            this.age = age;
            this.address = address;
        }
    }

    public String getName() {
        return this.name;
    }

    public Integer getAge() {
        return this.age;
    }

    public @NonNull String getAddress() {
        return this.address;
    }

    public boolean equals(final Object o) {
        if (o == this) {
            return true;
        } else if (!(o instanceof TestPojo)) {
            return false;
        } else {
            TestPojo other;
            label44: {
                other = (TestPojo)o;
                Object this$age = this.getAge();
                Object other$age = other.getAge();
                if (this$age == null) {
                    if (other$age == null) {
                        break label44;
                    }
                } else if (this$age.equals(other$age)) {
                    break label44;
                }

                return false;
            }

            Object this$name = this.getName();
            Object other$name = other.getName();
            if (this$name == null) {
                if (other$name != null) {
                    return false;
                }
            } else if (!this$name.equals(other$name)) {
                return false;
            }

            Object this$address = this.getAddress();
            Object other$address = other.getAddress();
            if (this$address == null) {
                if (other$address != null) {
                    return false;
                }
            } else if (!this$address.equals(other$address)) {
                return false;
            }

            return true;
        }
    }

    public int hashCode() {
        int PRIME = true;
        int result = 1;
        Object $age = this.getAge();
        result = result * 59 + ($age == null ? 43 : $age.hashCode());
        Object $name = this.getName();
        result = result * 59 + ($name == null ? 43 : $name.hashCode());
        Object $address = this.getAddress();
        result = result * 59 + ($address == null ? 43 : $address.hashCode());
        return result;
    }

    public String toString() {
        String var10000 = this.getName();
        return "TestPojo(name=" + var10000 + ", age=" + this.getAge() + ", address=" + this.getAddress() + ")";
    }
}
```

9. @Builder

As we have already discussed Builder Pattern in LLD ~~playlist~~



.java

```java
@Builder
public class TestPojo {

    String name;
    Integer age;
}
```

.class

```java
public class TestPojo {
    String name;
    Integer age;

    TestPojo(final String name, final Integer age) {
        this.name = name;
        this.age = age;
    }

    public static TestPojoBuilder builder() {
        return new TestPojoBuilder();
    }

    public static class TestPojoBuilder {
        private String name;
        private Integer age;

        TestPojoBuilder() {
        }

        public TestPojoBuilder name(final String name) {
            this.name = name;
            return this;
        }

        public TestPojoBuilder age(final Integer age) {
            this.age = age;
            return this;
        }

        public TestPojo build() {
            return new TestPojo(this.name, this.age);
        }

        public String toString() {
            return "TestPojo.TestPojoBuilder(name=" + this.name + ", age=" + this.age + ")";
        }
    }
}
```

```java
public class LombokTest {

    public static void main (String args[]) {

        TestPojo object = TestPojo.builder().age(20).name("xyz").build();
    }
}
```

10. @Cleanup

It ensures that given resource is automatically cleaned up before execution path exits the current scope.

.java

```java
public class TestPojo {

    public void readFile(String path) throws IOException {
        @Cleanup InputStream in = new FileInputStream(path);
        byte[] data = in.readAllBytes();
        System.out.println(new String(data));
    }
}
```

.class

```java
public class TestPojo {
    public TestPojo() {
    }

    public void readFile(String path) throws IOException {
        InputStream in = new FileInputStream(path);

        try {
            byte[] data = ((InputStream)in).readAllBytes();
            System.out.println(new String(data));
        } finally {
            if (Collections.singletonList(in).get(0) != null) {
                ((InputStream)in).close();
            }

        }

    }
}
```