COURSEWARE

Lombok

Contents

Overview

Lombok is a Java library which aims to reduce boilerplate code by automatically generating common class components such as: constructors, getters and setters, `toString()`, `equals()` and `hashCode()` methods.

Setting up your IDE with Lombok

Lombok has to be installed as an extension for your IDE in order to use it easily within your projects.
The reason for this is that Lombok generates methods at runtime so when you're writing the code the IDE can't 'see' any of methods Lombok will generate and will thus flag any use of them as an error - after installing the Lombok extension your IDE will be able to recognise these generated methods before they're actually created and won't complain about them.

To set up Lombok in your IDE, you'll need to get the Lombok jar from Lombok's site.

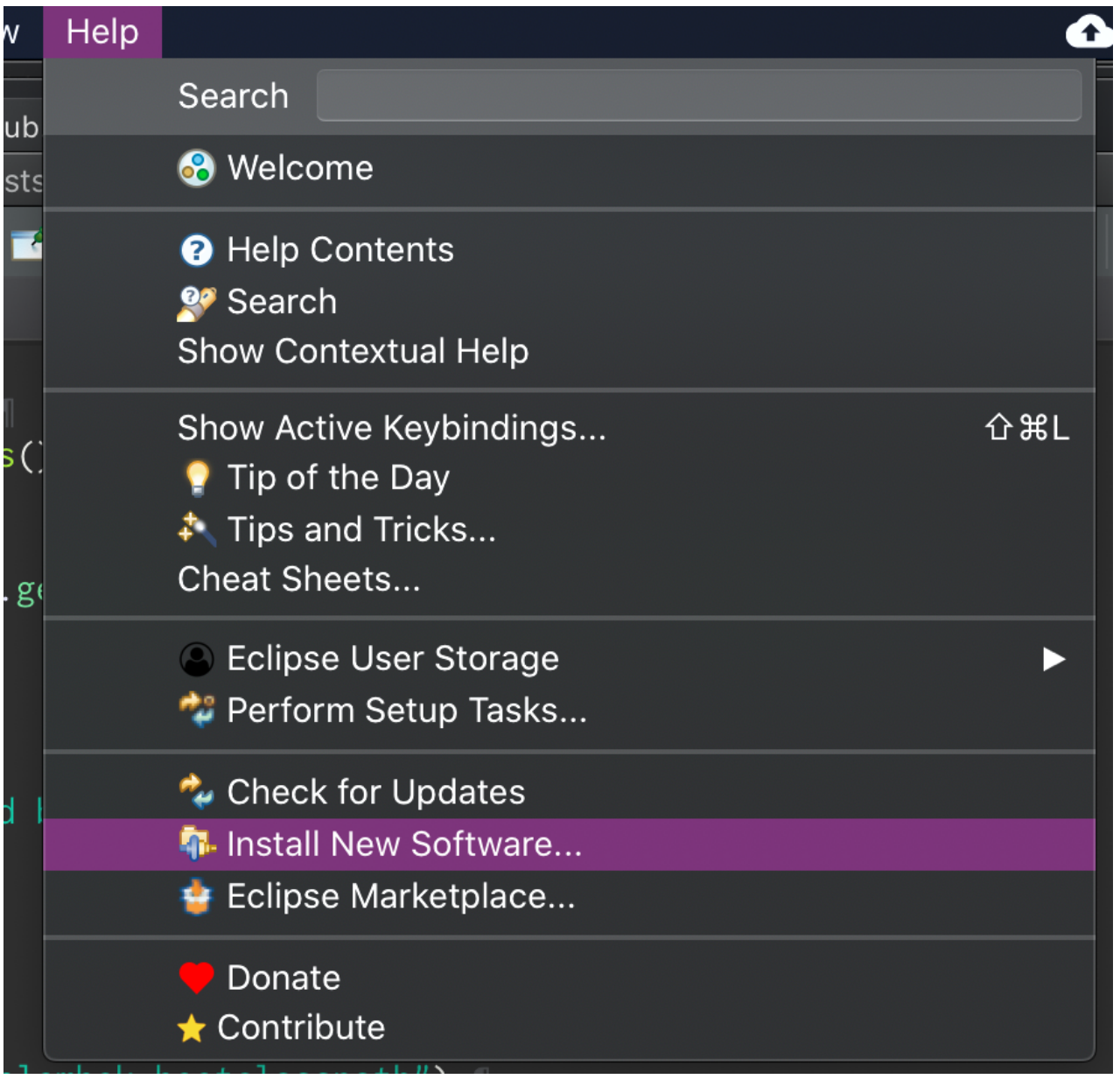
Once the `lombok.jar` file is downloaded, open it up. You should see a similar-looking screen to this:

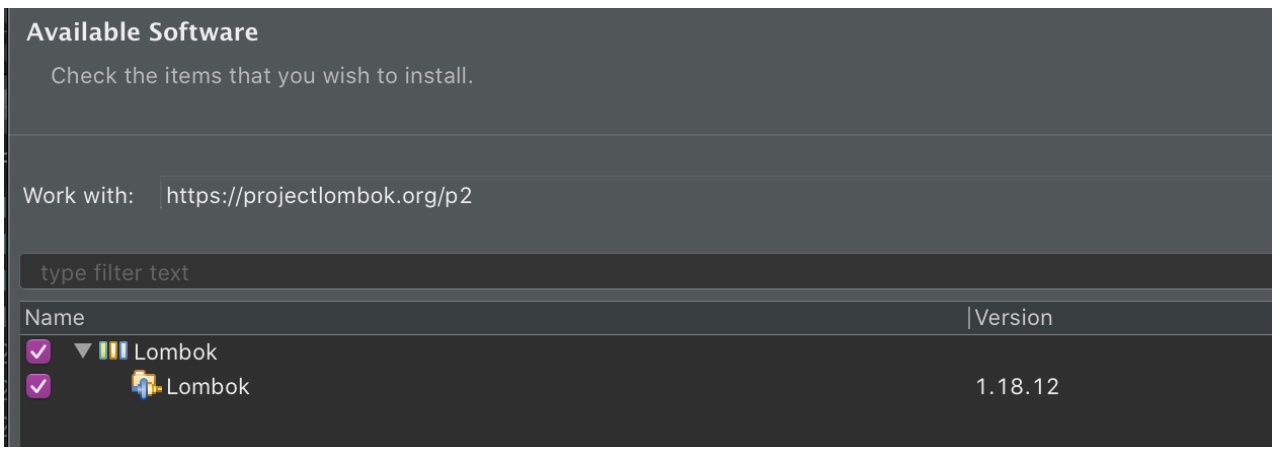
Lombok jar

Lombok should automatically scan for installed IDEs, but if not, click the `Specify location...` button, then point Lombok to your IDE's executable, such as `Eclipse.exe`:

Lombok pointing to Eclipse

If that doesn't work, you can also install Lombok directly within Eclipse:





Adding the dependency

You can add Lombok to a Spring application by adding the following dependency to your `pom.xml`:

```
<!-- LOMBOK -->
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
</dependency>
```

Tutorial

Here, we have an entity with lots of boilerplate code which we can remove:

▶ a very cluttered entity

Lombok has several useful annotations which can entirely eliminate this boilerplate code.

`@NoArgsConstructor`

The `@NoArgsConstructor` annotation will generate a constructor with no parameters:

```
@Entity
@NoArgsConstructor
public class Wood {

    // equivalent to:

    // public Wood() {
    //     super();
    // }

}
```

@AllArgsConstructor

The `@AllArgsConstructor` annotation will generate a constructor with one parameter for each field:

```
@Entity
@AllArgsConstructor
public class Wood {

/*  equivalent to:

    public Wood(Long id, boolean coniferous, String colour, String name, int
age, int weight, boolean artificial, boolean soft) {
        super();
        this.id = id;
        this.coniferous = coniferous;
        this.colour = colour;
        this.name = name;
        this.age = age;
        this.weight = weight;
        this.artificial = artificial;
        this.soft = soft;
    }
*/
}
```

@RequiredArgsConstructor

If your class has certain required fields, such as fields annotated with `@NonNull`, then Lombok will generate a constructor for you with all those required fields filled out.

(*note: you will still need to include specific constructors yourself, such as for creating objects without ID fields.*)

@Getter/@Setter

The `@Getter` and `@Setter` annotations replace your usual getter and setter code for you:

```
@Entity
@Getter
@Setter
public class Wood {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private boolean coniferous;

    private String colour;

    private String name;

    private int age;

    private int weight;

    private boolean artificial;

    private boolean soft;

    // equivalent to:

    // public String getColour() { return colour; }
    // public void setColour(String colour) { this.colour = colour; }
    // etc.

}
```

`@ToString`/`@EqualsAndHashCode`

The `@ToString` annotation will generate a `toString()` method for you. By default, it prints the class name, plus every field name, in order, separated by commas.

The `@EqualsAndHashCode` will do the same for your `equals()` and `hashCode()` implmentations. By default, it does this for all non-static fields:

```
@Entity
@ToString
@EqualsAndHashCode
public class Wood {

/*  equivalent to:

    @Override
    public String toString() {...}

    @Override
    public int hashCode() {...}

    @Override
    public boolean equals(Object obj) {...}
*/
```

`@Data`

The best of the lot is the `@Data` annotation, which combines the following annotations together for you:

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

The final implementation of Lombok for this entity might look like this:

```java
@Entity
@Data
@NoArgsConstructor
public class Wood {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private boolean coniferous;

    private String colour;

    private String name;

    private int age;

    private int weight;

    private boolean artificial;

    private boolean soft;

    public Wood(boolean coniferous, String colour, String name, int age, int
weight, boolean artificial, boolean soft) {
        super();
        this.coniferous = coniferous;
        this.colour = colour;
        this.name = name;
        this.age = age;
        this.weight = weight;
        this.artificial = artificial;
        this.soft = soft;
    }

}
```

This is a much cleaner and easier way of organising your entities (and DTOs, which are covered here).

The documentation for Lombok is a great resource if you need more specific configuration.

Excluding Lombok from test coverage

A frequent issue with boilerplate code is that it isn't excluded by default from test coverage tracker (e.g. Eclipse's inbuilt coverage tracker, SonarQube).

And, by default, even though Lombok automatically generates boilerplate code for you, it does not get excluded from these trackers.

This can be easily remedied, however, by adding the following code into a `lombok.config` file at the root level of your Spring application (i.e. at the same level as the `pom.xml`):

```
config.stopBubbling = true
lombok.addLombokGeneratedAnnotation = true
```

(*note: you may need to delete the `/target/` folder, refresh your project folder, restart your IDE, or re-add JUnit to your `classpath` before it registers correctly in your IDE.*)

Exercises

Implement Lombok into your account project.

Add a configuration file which ignores Lombok-generated code when checking test coverage.