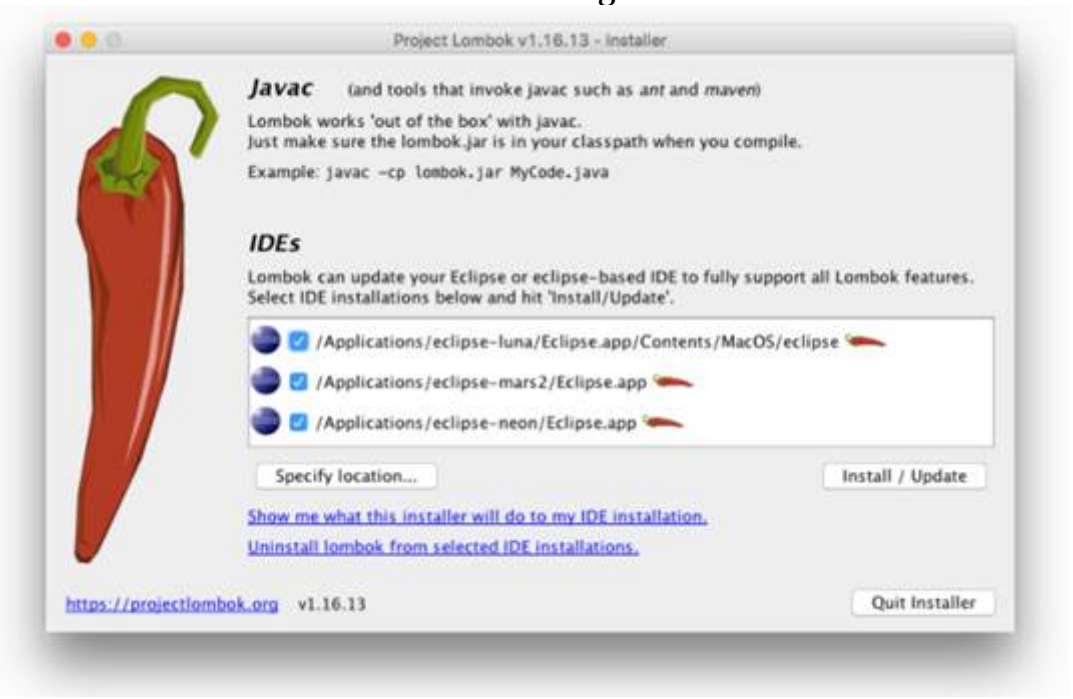
Un poco de Java y +

Otra forma de hablar de nuestro día a día…

ECLIPSE, HERRAMIENTAS, JAVA, LIBRERÍAS, QUÉ ES

¿Cómo eliminar el código repetitivo en Java con Lombok?

Date: 22 diciembre 2017 Author: Luis Miguel Gracia 0 Comentarios



Lombok es una librería Java que permite eliminar el código repetitivo al que nos obliga (aún hoy) Java, permitiendo por ejemplo generar los gets, los sets, los Logs, el toString,… con una anotación (lástima que Spring Roo no evolucionase lo suficiente J).

Lombok se integra en nuestro IDE ayudándonos a desarrollar más rápido.

Veamos cómo instalarlo y algunos ejemplos de uso.

Lombok en Maven:

Si uso Maven me bastará con definir la dependencia:

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>1.16.18</version>

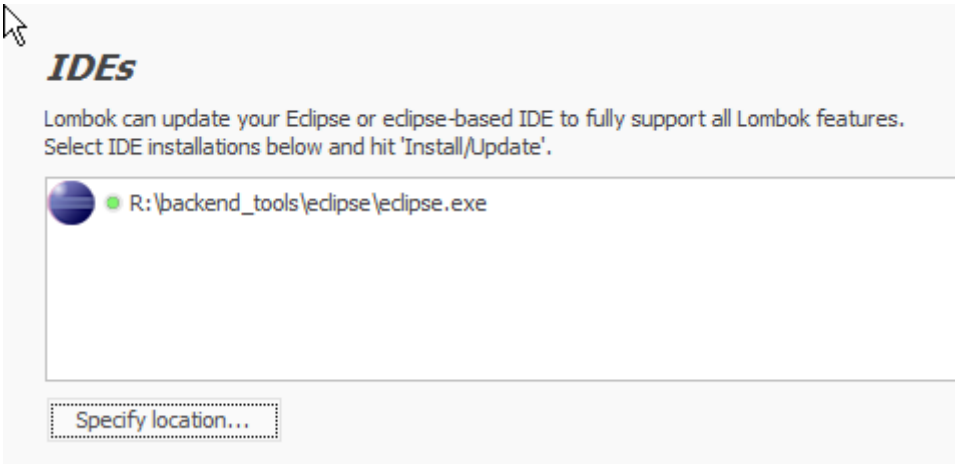
<scope>provided</scope>

</dependency>

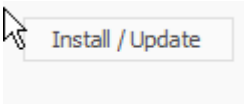
Para instalarlo en Eclipse basta con descargar la última versión del JAR (https://projectlombok.org/downloads/lombok.jar) y ejecutar

>java –jar lombok.jar

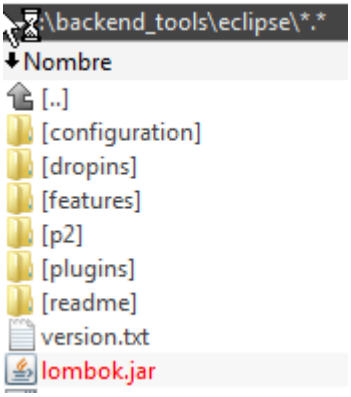
(con Eclipse cerrado). El JAR intentaré encontrar los diferentes Eclipses instalados, aunque podéis especificar la localización del Eclipse a actualizar:



Tras esto pincharé



La instalación lo que hace es copiar el JAR lombok.jar a la carpeta de mi Eclipse:



Si lanzo Eclipse (como es mi caso) desde un bat tendré que añadir al script de lanzamiento:

```
call R:
call R:\scripts\setenv.bat
call cd R:\backend_tools\eclipse\
START /B  eclipse.exe -clean -nl en -refresh -data R:\workspace\backend_ide -vmargs -javaagent:lombok.jar
echo .
echo . Dont close this window
```

Una vez instalado veamos algunos posibles usos de Lombok:

Lombok incluye estas anotaciones:

val

Finally! Hassle-free final local variables.

@NonNull

or: How I learned to stop worrying and love the NullPointerException.

@Cleanup

Automatic resource management: Call your close() methods safely with no hassle.

@Getter/@Setter

Never write public int getFoo() {return foo;} again.

@ToString

No need to start a debugger to see your fields: Just let lombok generate a toString for you!

@EqualsAndHashCode

Equality made easy: Generates hashCode and equals implementations from the fields of your object..

@NoArgsConstructor, @RequiredArgsConstructor and @AllArgsConstructor

Constructors made to order: Generates constructors that take no arguments, one argument per final / non-nullfield, or one argument for every field.

@Data

All together now: A shortcut for @ToString, @EqualsAndHashCode, @Getter on all fields, and @Setter on all non-final fields, and @RequiredArgsConstructor!

@Value

Immutable classes made very easy.

@Builder

… and Bob's your uncle: No-hassle fancy-pants APIs for object creation!

@SneakyThrows

To boldly throw checked exceptions where no one has thrown them before!

@Synchronized

synchronized done right: Don't expose your locks.

@Getter(lazy=true)

Laziness is a virtue!

@Log

Captain's Log, stardate 24435.7: "What was that line again?"

La anotación imprescindible es la de **@Getter @Setter** que se usa así:

```
public class GetterSetterExample {
  /**
   * Age of the person. Water is wet.
   *
   * @param age New value for this person's age. Sky is blue.
   * @return The current value of this person's age. Circles are round.
   */
  @Getter @Setter private int age = 10;

  /**
   * Name of the person.
   * -- SETTER --
   * Changes the name of this person.
   *
   * @param name The new value.
   */
  @Setter(AccessLevel.PROTECTED) private String name;
```

También es útil el **@ToString:**

```
@ToString(exclude="id")
public class ToStringExample {
  private static final int STATIC_VAR = 10;
  private String name;
  private Shape shape = new Square(5, 10);
  private String[] tags;
  private int id;
```

Esto nos lleva a la anotación **@Data** es un wrapper que equivale a incluir las anotaciones: **@ToString, @EqualsAndHashCode, @Getter on all fields, and @Setter on all non-final fields, and @RequiredArgsConstructor**

Este código:

```
@Data public class DataExample {
  private final String name;
  @Setter(AccessLevel.PACKAGE) private int age;
  private double score;
  private String[] tags;
```

Equivale en Java a:

```
public class DataExample {
  private final String name;
  private int age;
  private double score;
  private String[] tags;

  public DataExample(String name) {
    this.name = name;
  }

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

  void setAge(int age) {
    this.age = age;
  }

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

  public void setScore(double score) {
    this.score = score;
  }
```

…….

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

@Override public boolean equals(Object o) {
  if (o == this) return true;
  if (!(o instanceof DataExample)) return false;
  DataExample other = (DataExample) o;
  if (!other.canEqual((Object)this)) return false;
  if (this.getName() == null ? other.getName() != null : !this.getName().equals(other.getName())) return false;
  if (this.getAge() != other.getAge()) return false;
  if (Double.compare(this.getScore(), other.getScore()) != 0) return false;
  if (!Arrays.deepEquals(this.getTags(), other.getTags())) return false;
  return true;
}

@Override public int hashCode() {
  final int PRIME = 59;
  int result = 1;
  final long temp1 = Double.doubleToLongBits(this.getScore());
  result = (result*PRIME) + (this.getName() == null ? 43 : this.getName().hashCode());
  result = (result*PRIME) + this.getAge();
  result = (result*PRIME) + (int)(temp1 ^ (temp1 >>> 32));
  result = (result*PRIME) + Arrays.deepHashCode(this.getTags());
  return result;
}
```

Otra anotación que nos ahorrará código y dolores de cabeza (ya veréis como sí): @NonNull

Cuando la uso en este método:

```
public NonNullExample(@NonNull Person person) {
  super("Hello");
  this.name = person.getName();
}
```

Equivale a:

```
public NonNullExample(@NonNull Person person) {
  super("Hello");
  if (person == null) {
    throw new NullPointerException("person");
  }
  this.name = person.getName();
}
```

Interesante el soporte para Logs vía un conjunto de anotaciones:

```
@CommonsLog
Creates
private static final org.apache.commons.logging.Log log = org.apache.commons.logging.LogFactory.getLog(LogExample.class);
@JBossLog
Creates
private static final org.jboss.logging.Logger log = org.jboss.logging.Logger.getLogger(LogExample.class);
@Log
Creates
private static final java.util.logging.Logger log = java.util.logging.Logger.getLogger(LogExample.class.getName());
@Log4j
Creates private static final org.apache.log4j.Logger log = org.apache.log4j.Logger.getLogger(LogExample.class);
@Log4j2
Creates
private static final org.apache.logging.log4j.Logger log = org.apache.logging.log4j.LogManager.getLogger(LogExample.class);
@Slf4j
Creates private static final org.slf4j.Logger log = org.slf4j.LoggerFactory.getLogger(LogExample.class);
@XSlf4j
Creates
private static final org.slf4j.ext.XLogger log = org.slf4j.ext.XLoggerFactory.getXLogger(LogExample.class);
```

Este código:

```
@CommonsLog(topic="CounterLog")
public class LogExampleCategory {

  public static void main(String... args) {
    log.error("Calling the 'CounterLog' with a message");
  }
}
```

Equivale a:

```
public class LogExampleCategory {
  private static final org.apache.commons.logging.Log log = org.apache.commons.logging.LogFactory.getLog("CounterLog"

  public static void main(String... args) {
    log.error("Calling the 'CounterLog' with a message");
  }
```

La última anotación que quería comentar es una de las más interesantes y que más tiempo ahorran, **@Builder** que crea Builders para nuestra aplicación. Por ejemplo:

```java
@Builder
public class Usuario implements Serializable{

    private static final long serialVersionUID = 2

    private long id;
    private String nombre;
    private String primerApellido;
    private String segundoApellido;
    private String mail;
    private String telefono;
    private boolean activo;
}
```

Me permite hacer esto:

```java
private Usuario cliente
            = new Usuario.UsuarioBuilder()
                                .id(102)
                                .nombre("Alejandro")
                                .primerApellido("Magno")
                                .mail("correo@ejemplo.com")
                                .build();
```

Podéis consultar el resto de características de Lombok aquí:

https://projectlombok.org/features/all