

Assignment: Getting Started with Lombok

[Lombok](#) is a Java library that makes you more productive by auto-generating [boilerplate code](#).

Difficulty: ☆☆

Learning objectives:

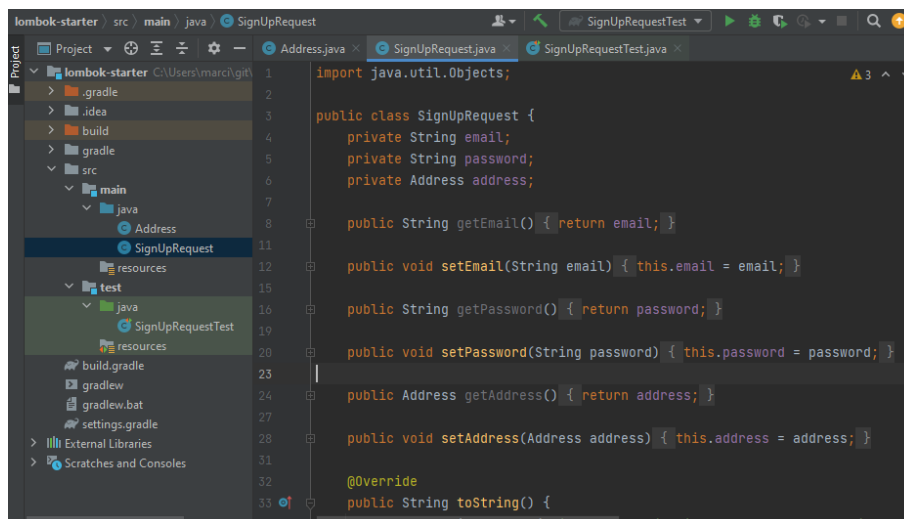
- I can auto-generate getter and setter methods
- I can auto-generate *equals* and *hashCode* methods
- I can auto-generate an all arguments constructor
- I can auto-generate a required arguments constructor
- I can auto-generate a builder for a class

Estimated time required: 45 minutes

Step 1: Getting the start-up code

Download the [starter code from Canvas](#) and load it into IntelliJ.

The project has a couple of classes and one unit test. Take your time and explore it.



```
import java.util.Objects;

public class SignUpRequest {
    private String email;
    private String password;
    private Address address;

    public String getEmail() { return email; }

    public void setEmail(String email) { this.email = email; }

    public String getPassword() { return password; }

    public void setPassword(String password) { this.password = password; }

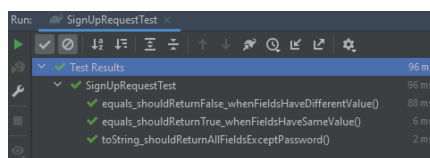
    public Address getAddress() { return address; }

    public void setAddress(Address address) { this.address = address; }

    @Override
    public String toString() {
        return "SignUpRequest(email=" + email + ", password=" + password + ", address=" + address + ")";
    }
}
```

You might have noticed that great part of the code is just boilerplate: getters, setters, toString, equals and hashCode implementations. Let's improve that on the following sections.

But before we start [refactoring](#) make sure test *SignUpRequestTest* is passing.



Test Results	Time
SignUpRequestTest	96 ms
equals_shouldReturnFalse_whenFieldsHaveDifferentValue()	88 ms
equals_shouldReturnTrue_whenFieldsHaveSameValue()	6 ms
toString_shouldReturnAllFieldsExceptPassword()	2 ms

Field Code Changed

Field Code Changed

Field Code Changed

Formatted: English (United States)

Formatted: English (United States)

Field Code Changed

Step 2: Getters and Setters

The getters and setters in this code are not doing anything special. They simply provide access to the fields. That makes them really good candidates for auto-generation with Lombok.

Delete all getters and setters from class *SignUpRequest* and then annotate the class with `@Getter` and `@Setter` (see below).

Important tip: you can always “Ctrl + click” on the annotation to understand better what it is doing, you might need to click on “Download sources” so the IDE downloads the rest of the class information. There you can see the comments explaining the annotation and probably some usage examples. Another way would be to look for the documentation online.

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

...

@Getter
@Setter
public class SignUpRequest {
    private String email;
    private String password;
    private Address address;

    ...
}
```

Now run *SignUpRequestTest*. All tests should still be passing.

Please do the same for class *Address*.

Note 1: don't forget running the tests after each [refactoring](#) round from now on.

Note 2: It's also good to notice that in general Lombok annotations don't need to be on the class level. For instance many times it will actually make sense to have getters and setters only for specific fields of a class. In that case you can use `@Getter` / `@Setter` on the field level having fine-control on what can be retrieved and changed respectively. [Explore Lombok's documentation](#) in order to learn more.

Step 3: equals and hashCode

Same reasoning applies to the implementations of *equals* and *hashCode*.

Let's delete these two methods and add [annotation @EqualsAndHashCode](#) on both classes. This is how class *Address* will look like:

```
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
@EqualsAndHashCode
public class Address {
```

Field Code Changed

Field Code Changed

Field Code Changed

```

private String street;
private String number;
private String complement;
private String postalCode;
private String city;
private String state;
}

```

Step 4: toString

Time to use [Lombok's toString annotation](#).

Please delete the *toString* method from class *SignUpRequest* and then annotate it with *@ToString*. Also do add the *@ToString* annotation to class *Address*.

As defined in the unit test, *toString*'s result shouldn't contain the field password. This kind of customization is easy with Lombok. Check the example below:

```

import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

@Getter
@Setter
@EqualsAndHashCode
@ToString
public class SignUpRequest {
    private String email;
    @ToString.Exclude
    private String password;
    private Address address;
}

```

Field password is annotated with *@ToString.Exclude*. The same kind of thing can be done with *@EqualsAndHashCode*.

As usual, make sure the tests are passing.

Step 5: @Data annotation

It's common that classes are annotated with *@Getter*, *@Setter*, *@EqualsAndHashCode* and *@ToString* at the same time (just like *SignUpRequest* and *Address*).

A simpler way of achieving the same is by using [@Data annotation](#).

After applying it this is how our classes will look like:

```

import lombok.Data;
import lombok.ToString;

@Data
public class SignUpRequest {
    private String email;
    @ToString.Exclude
    private String password;
    private Address address;
}

```

Field Code Changed

Field Code Changed

```
import lombok.Data;

@Data
public class Address {
    private String street;
    private String number;
    private String complement;
    private String postalCode;
    private String city;
    private String state;
}
```

What a difference, right?? As a software developer your time will be valuable (and expensive), so when it makes sense do take advantage of tools like Lombok.

Step 6: Constructors

Implementing constructors is something recurrent in object-oriented programming. Many times, they are really simple (just assigning arguments to fields), what makes them a good candidate for auto-generation.

Lombok helps us with the following annotations: [@AllArgsConstructor](#), [@RequiredArgsConstructor](#) and [@NoArgsConstructor](#).

Let's digest them quickly (for more details check the link above):

- [@AllArgsConstructor](#): generates a constructor with all the class's fields.
- [@RequiredArgsConstructor](#): generates a constructor for the class's *final* fields.
- [@NoArgsConstructor](#): generates a constructor with no arguments, just like the [default constructor](#).

Let's try it out by adding a [@AllArgsConstructor](#) to class *Address*. This change will cause a compilation error in *SignUpRequestTest* class. Can you figure out why?? Please fix the compilation error (two options here, feel free to pick).

Final Step: Builder

When a class has many fields, constructors start to look a bit messy. For instance, it would be easy to switch values and initialize the wrong fields in the code below:

```
Address address = new Address("My street", "19A", null, "1234AB",
    "Eindhoven", "North Brabant");
```

The [Builder pattern](#) together with [method chaining](#) provides an elegant solution for this, but it's a lot of work (and boilerplate code) in order to make use of it.

Again, Lombok for the rescue! By using [@Builder annotation](#) you get both. Check how the code looks like below and try it yourself:

```
@AllArgsConstructor
@Data
@Builder
public class Address {
```

Field Code Changed

Field Code Changed

Field Code Changed

Field Code Changed

Field Code Changed

```
} ...
```

```
Address address = Address.builder()  
    .street("My street")  
    .number("19A")  
    .postalCode("1234AB")  
    .city("Eindhoven")  
    .state("North Brabant")  
    .build();
```

Now it's crystal clear what each argument means.

As usual after the changes make sure the tests are passing. And that's it! Welcome to Lombok!