CSD-4464 Java EE

Class 7: Lombok

Lambton College

Lombok

- Project Lombok is a java library that spices up your development life by saving you from writing a bunch of boilerplate code
- Compile time annotation based framework
- Never write a setter/getter/basic constructor again!

@Getter / @Setter

- Can be applied at class level, or at member level
- Generates Setters/Getters at compile time so you don't have to create them and have them cluttering your code!
- https://projectlombok.org/features/GetterSetter

Example

@Getter

Public class Person {

Or

@Getter private final String Name

@EqualsAndHashcode + @ToString

- Used to write .equals(), .hashcode() and .toString()
- Applied to class definition
- https://projectlombok.org/features/EqualsAndHashCode

Example

@ToString

@EqualsAndHashcode

Public class Person {

@NonNull

- You can use @NonNull to have Lombok generate a null-check statement for you
- Applied to Constructor or Method parameters

Example

Public Person(@NonNull String name)

@AllArgsConstructor / @NoArgsConstructor

- Creates constructors for either all members of the class, or a no argument constructor
- Applied to class definition
- Members variables marked with @NonNull will result with null checks on those parameters

Example

```
@AllArgsConstructor
Public class Person {
```

@RequiredArgsConstructor

- Similar to the previous constructors, however it creates a constructor with a parameter for each field that requires special handling (e.x any non initialized final marked or NonNull marked variables)
- You can optionally pass in access level to change the scope of the constructor

Example

```
@RequiredArgsConstructor(access = AccessLevel.PRIVATE)
public class Person {
```

@Data

- Convenient shorthand annotation for @ToString, @EqualsAndHashCode, @Getter, @Setter, @RequiredArgsConstructor

Example

```
@Data
Public class Person {
```

Builder pattern

- The Builder pattern is a creational pattern (used to create and configure objects)
- Helps provide compiler safety to object creation
- Keeps the number of constructors needed to a minimum (one)
- Allows for very verbose but very clear code

Consider the following

- Given the class

```
Public class Person {
   private String firstName;
   private String middleName;
   private String lastName;
   private String nickName;
}
```

Consider the following cont.

Given that nicknames/middlenames are generally optional,
we would have the following constructors

public Person(String fname, String lname)
public Person(String fname, String mname, String lname, String nname)

But what about cased where we only have a middle name or nickname

Public Person(String fname, String lname, String nname)
public Person(String fname, String mname, String lname)

^will not compile since the signatures are ambiguous

Builder style construction

- Person p1 = Person.builder()
  .firstName(“jane”)
  .middleName(“mary”)
  .lastName(“doe”)
  .build()

- Person p2 = Person.builder()
  .firstName(“jane”)
  .nickname(“mrs popular”)
  .lastName(“doe”)
  .build()

@Builder

- Can be placed on a Class, Constructor, or a Method
- Produces easy to use Builder API's for your classes
- https://projectlombok.org/features/Builder

Example

```
@Builder
Public Class Person {
```

Or

```
@Builder
private Person of(String name…);
```