# @With

## Immutable 'setters' - methods that create a clone but with one changed field.

`@Wither` was introduced as experimental feature in lombok v0.11.4.

`@Wither` was renamed to `@With`, and moved out of experimental and into the core package, in lombok v1.18.10.

## Overview

The next best alternative to a setter for an immutable property is to construct a clone of the object, but with a new value for this one field. A method to generate this clone is precisely what `@With` generates: a `withFieldName(newValue)` method which produces a clone except for the new value for the associated field.

For example, if you create `public class Point { private final int x, y; }`, setters make no sense because the fields are final. `@With` can generate a `withX(int newXValue)` method for you which will return a new point with the supplied value for `x` and the same value for `y`.

The `@With` relies on a constructor for all fields in order to do its work. If this constructor does not exist, your `@With` annotation will result in a compile time error message. You can use Lombok's own `@AllArgsConstructor`, or as `Value` will automatically produce an all args constructor as well, you can use that too. It's of course also acceptable if you manually write this constructor. It must contain all non-static fields, in the same lexical order.

Like `@Setter`, you can specify an access level in case you want the generated with method to be something other than `public`: `@With(level = AccessLevel.PROTECTED)`. Also like `@Setter`, you can also put a `@With` annotation on a type, which means a `with` method is generated for each field (even non-final fields).

To put annotations on the generated method, you can use `onMethod=@__({@AnnotationsHere})`. Be careful though! This is an experimental feature. For more details see the documentation on the onX feature.

javadoc on the field will be copied to generated with methods. Normally, all text is copied, and `@param` is *moved* to the with method, whilst `@return` lines are stripped from the with method's javadoc. Moved means: Deleted from the field's javadoc. It is also possible to define unique text for the with method's javadoc. To do that, you create a 'section' named `WITH`. A section is a line in your javadoc containing 2 or more dashes, then the text 'WITH', followed by 2 or more dashes, and nothing else on the line. If you use sections, `@return` and `@param` stripping / copying for that section is no longer done (move the `@param` line into the section).

## With Lombok

```
import lombok.AccessLevel;
import lombok.NonNull;
import lombok.With;

public class WithExample {
  @With(AccessLevel.PROTECTED) @NonNull private final String name;
  @With private final int age;

  public WithExample(@NonNull String name, int age) {
    this.name = name;
    this.age = age;
  }
}
```

## Vanilla Java

```
import lombok.NonNull;

public class WithExample {
  private @NonNull final String name;
  private final int age;

  public WithExample(String name, int age) {
    if (name == null) throw new NullPointerException();
    this.name = name;
    this.age = age;
  }

  protected WithExample withName(@NonNull String name) {
    if (name == null) throw new java.lang.NullPointerException("name");
    return this.name == name ? this : new WithExample(name, age);
  }

  public WithExample withAge(int age) {
    return this.age == age ? this : new WithExample(name, age);
  }
}
```

## Supported configuration keys:

<>
`lombok.accessors.prefix` **+=** ***a field prefix*** **(default: empty list)**
This is a list property; entries can be added with the `+=` operator. Inherited prefixes from parent config files can be removed with the `-=` operator. Lombok will strip any matching field prefix from the name of a field in order to determine the name of the getter/setter to generate. For example, if `m` is one of the prefixes listed in this setting, then a field named `mFoobar` will result in a getter named `getFoobar()`, not `getMFoobar()`. An explicitly configured `prefix` parameter of an `@Accessors` annotation takes precedence over this setting.
`lombok.accessors.capitalization` **= [** `basic` **|** `beanspec` **] (default: basic)**
Controls how tricky cases like `uShaped` (one lowercase letter followed by an upper/titlecase letter) are capitalized. `basic` capitalizes that to `withUShaped`, and `beanspec` capitalizes that to `withuShaped` instead.
Both strategies are commonly used in the java ecosystem, though `beanspec` is more common.
`lombok.with.flagUsage` **= [** `warning` **|** `error` **] (default: not set)**
Lombok will flag any usage of `@With` as a warning or error if configured.

## Small print

With methods cannot be generated for static fields because that makes no sense.

With methods can be generated for abstract classes, but this generates an abstract method with the appropriate signature.

When applying `@With` to a type, static fields and fields whose name start with a $ are skipped.

For generating the method names, the first character of the field, if it is a lowercase character, is title-cased, otherwise, it is left unmodified. Then, `with` is prefixed.

No method is generated if any method already exists with the same name (case insensitive) and same parameter count. For example, `withX(int x)` will not be generated if there's already a method `withX(String... x)` even though it is technically possible to make the method. This caveat exists to prevent confusion. If the generation of a method is skipped for this reason, a warning is emitted instead. Varargs count as 0 to N parameters.

Various well known annotations about nullity cause null checks to be inserted and will be copied to the parameter. See Getter/Setter documentation's small print for more information.

If you have configured a nullity annotation flavour via `lombok.config` key `lombok.addNullAnnotations`, the method or return type (as appropriate for the chosen flavour) is annotated with a non-null annotation.