# @NonNull

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

`@NonNull` was introduced in lombok v0.11.10.

## Overview

You can use `@NonNull` on a record component, or a parameter of a method or constructor. This will cause to lombok generate a null-check statement for you.

Lombok has always treated various annotations generally named `@NonNull` on a field as a signal to generate a null-check if lombok generates an entire method or constructor for you, via for example `@Data`. However, using lombok's own `@lombok.NonNull` on a parameter or record component results in the insertion of the null-check at the top of that method.

The null-check looks like `if (param == null) throw new NullPointerException("param is marked non-null but is null");` and will be inserted at the very top of your method. For constructors, the null-check will be inserted immediately following any explicit `this()` or `super()` calls. For record components, the null-check will be inserted in the 'compact constructor' (the one that has no argument list at all), which will be generated if you have no constructor. If you have written out the record constructor in long form (with parameters matching your components exactly), then nothing happens - you'd have to annotate the parameters of this long-form constructor instead.

If a null-check is already present at the top, no additional null-check will be generated.

## With Lombok

```
import lombok.NonNull;

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

## Vanilla Java

```
import lombok.NonNull;

public class NonNullExample extends Something {
  private String name;
  
  public NonNullExample(@NonNull Person person) {
    super("Hello");
    if (person == null) {
      throw new NullPointerException("person is marked non-null but is null");
    }
    this.name = person.getName();
  }
}
```

## Supported configuration keys:

**`lombok.nonNull.exceptionType` = [`NullPointerException` | `IllegalArgumentException` | `JDK` | `Guava` | `Assertion`] (default: `NullPointerException`).**
When lombok generates a null-check `if` statement, by default, a `java.lang.NullPointerException` will be thrown with '*field name* is marked non-null but is null' as the exception message. However, you can use `IllegalArgumentException` in this configuration key to have lombok throw that exception with this message instead. By using `Assertion`, an `assert` statement with the same message will be generated. The keys `JDK` or `Guava` result in an invocation to the standard nullcheck method of these two frameworks: `java.util.Objects.requireNonNull([field name here], "[field name here] is marked non-null but is null");` or `com.google.common.base.Preconditions.checkNotNull([field name here], "[field name here] is marked non-null but is null");` respectively.

**`lombok.nonNull.flagUsage` = [`warning` | `error`] (default: not set)**
Lombok will flag any usage of `@NonNull` as a warning or error if configured.

## Small print

Lombok's detection scheme for already existing null-checks consists of scanning for if statements or assert statements that look just like lombok's own. Any 'throws' statement as the 'then' part of the if statement, whether in braces or not, counts. Any invocation to any method named `requireNonNull` or `checkNotNull` counts. The conditional of the if statement *must* look exactly like `PARAMNAME == null`; the assert statement *must* look exactly like `PARAMNAME != null`. The invocation to a `requireNonNull`-style method must be on its own (a statement which just invokes that method), or must be the expression of an assignment or variable declaration statement. The first statement in your method that is not such a null-check stops the process of inspecting for null-checks.

While `@Data` and other method-generating lombok annotations will trigger on various well-known annotations that signify the field must never be `@NonNull`, this feature only triggers on lombok's own `@NonNull` annotation from the `lombok` package.

A `@NonNull` on a primitive parameter results in a warning. No null-check will be generated.

A `@NonNull` on a parameter of an abstract method used to generate a warning; starting with version 1.16.8, this is no longer the case, to acknowledge the notion that `@NonNull` also has a documentary role. For the same reason, you can annotate a method as `@NonNull`; this is allowed, generates no warning, and does not generate any code.

credits | Copyright © 2009-2024 The Project Lombok Authors, licensed under the MIT license.