# Error Prone

Google’s **Error Prone** is a static analysis tool that runs during Java compilation to find common programming mistakes early, preventing bugs before your code runs. Two important annotations it supports are:

* @Nullable — to mark that a variable or method may return null
* @Var — to explicitly mark a variable as mutable (changeable)

This guide explains these annotations clearly with examples so you can improve your Spring Boot or Java project code quality.

## Why Use @Nullable?

* Java’s default assumption is that variables or method returns are **not null**, but many bugs happen due to **null values**.
* @Nullable tells the compiler and static tools:  
  “Hey, this value can be null. So, be careful when using it!”
* Helps prevent **NullPointerExceptions (NPEs)** — a very common runtime error in Java apps.
* Makes your code **self-documenting** about nullability, improving readability and maintainability.

### Example 1: Method that may return null

import javax.annotation.Nullable;

@Nullable

public String findUserName(String userId) {

if (userId == null || userId.isEmpty()) {

return null; // Explicitly returning null

}

return "Kalven"; // Some valid username

}

### Example 2: Unsafe usage that may cause NPE

String name = findUserName("123");

System.out.println(name.toUpperCase()); // Risk of NullPointerException if name is null

If *name* is null, this will throw an NPE at runtime.

### Example 3: Safe usage with explicit null check

java

CopyEdit

String name = findUserName("123");

if (name != null) {

System.out.println(name.toUpperCase()); // Safe access after null check

} else {

System.out.println("Name not found");

}

## Why Use @Var?

* By default, Google Error Prone prefers **immutable variables** (variables that don’t change after assignment).
* If you want to **reassign or change** a local variable, you must explicitly mark it as mutable using @Var.
* This helps catch unintentional variable reassignments and promotes clearer, safer code.

### Example 1: Correct use of @Var for mutable variable

import com.google.errorprone.annotations.Var;

public void countItems() {

@Var int count = 0; // Marking count as mutable

count += 1; // Allowed reassignment

System.out.println(count);

}

### Example 2: Missing @Var causes Error Prone warning

public void countItems() {

int count = 0;

count += 1; // Warning: Reassignment without @Var annotation

}

## How Does This Help Our Project?

* Using @Nullable helps you **explicitly handle null values**, reducing bugs and crashes.
* Using @Var enforces **intentional mutability**, making your code more predictable.
* Combined, these annotations **increase code safety and maintainability** in your Spring Boot or Java app.
* Google Error Prone will catch mistakes at **compile time**, saving debugging time later.