# Using @Nullable and @Var in Java with Error Prone

## Introduction

Google's Error Prone is a static code analysis tool that helps developers catch common Java mistakes at compile time. Two important annotations supported by Error Prone are @Nullable and @Var.

## Using @Nullable

@Nullable indicates that a method return value or variable may be null. This helps developers and tools like Error Prone identify potential NullPointerExceptions (NPEs) during development.

Example:

```
@Nullable
public String getUserName(String userId) {
    if (userId == null) {
        return null;
    }
    return "Kalven";
}
```

Without a null check, using the return value directly may cause an NPE:

```
String name = getUserName("123");
System.out.println(name.toUpperCase()); // May throw NPE!
```

# Using @Nullable and @Var in Java with Error Prone

Correct way:

```
String name = getUserName("123");
if (name != null) {
    System.out.println(name.toUpperCase());
}
```

## Using @Var

@Var indicates that a local variable is intentionally mutable (i.e., it will be modified after being initialized).

Example:

```
import com.google.errorprone.annotations.Var;

public void process() {
    @Var int counter = 0;
    counter += 1; // OK: @Var allows reassignment
}
```

Without @Var, Error Prone will warn you if you reassign the variable:

```
int counter = 0;
counter += 1; // Warning: reassigned but not marked @Var
```

# Using @Nullable and @Var in Java with Error Prone

## Conclusion

- Use @Nullable to mark potentially null values and avoid NPEs.

- Use @Var to signal that a variable is expected to be mutable.

- Error Prone improves code quality by catching common bugs during compilation.

These annotations help enforce best practices and make your Spring Boot Java application safer and more maintainable.