Exploration Report: Kotlin Documentation Tool (KDoc)

Lamia Binta Latif
Nuzhat Nairy Afrin

September 18, 2024

1 Introduction

As part of our project development process, Lamia Binte Latif and I, Nuzhat Nairy Afrin, explored Kotlin's documentation tool, KDoc, to understand how to effectively document Kotlin code. Proper documentation is a crucial part of any development project, especially for maintaining clarity and collaboration among team members. KDoc is Kotlin's version of Java's Javadoc, enabling developers to generate documentation from code comments. This report details the findings from our exploration of KDoc and discusses its relevance for the *KII-KINBO* supershop app.

2 How To Install KDoc

Click here: Installation Guide

3 Overview of KDoc

KDoc is designed to generate human-readable documentation directly from code comments. It follows a similar structure to Javadoc, making it familiar for developers transitioning from Java to Kotlin. KDoc comments are written using /** ... */ and can be placed above classes, functions, and properties.

3.1 KDoc Syntax

Just like with Javadoc, KDoc comments start with /** and end with */. Every line of the comment may begin with an asterisk, which is not considered part of the comment contents.

By convention, the first paragraph of the documentation text (the block of text until the first blank line) is the summary description of the element, and

the following text is the detailed description. Every block tag begins on a new line and starts with the @ character.

3.2 Example of a KDoc Comment

Here's an example of a class documented using KDoc:

```
/**
 * A group of *members*.
 *
 * This class has no useful logic; it's just a documentation example.
 *
 * @param T the type of a member in this group.
 * @property name the name of this group.
 * @constructor Creates an empty group.
 */
class Group<T>(val name: String) {
    /**
     * Adds a [member] to this group.
     * @return the new size of the group.
     */
    fun add(member: T): Int { ... }
}
```

4 Block Tags in KDoc

KDoc currently supports the following block tags:

- **@param name**: Documents a value parameter of a function or a type parameter of a class, property, or function.
- **@return**: Documents the return value of a function.
- **@constructor**: Documents the primary constructor of a class.
- **@receiver**: Documents the receiver of an extension function.
- **@property name**: Documents the property of a class which has the specified name.
- **@throws class**, **@exception class**: Documents an exception which can be thrown by a method.
- **@sample identifier**: Embeds the body of the function to show an example.
- **@see identifier**: Adds a link to the specified class or method.
- **@author**: Specifies the author of the element.

- **@since**: Specifies the version in which the element was introduced.
- **@suppress**: Excludes the element from the generated documentation.

5 Markdown Support for Formatting

KDoc supports basic Markdown syntax, allowing developers to format their documentation with bold, italic, code blocks, and hyperlinks. This ensures that the documentation is not only informative but also easy to read.

Example:

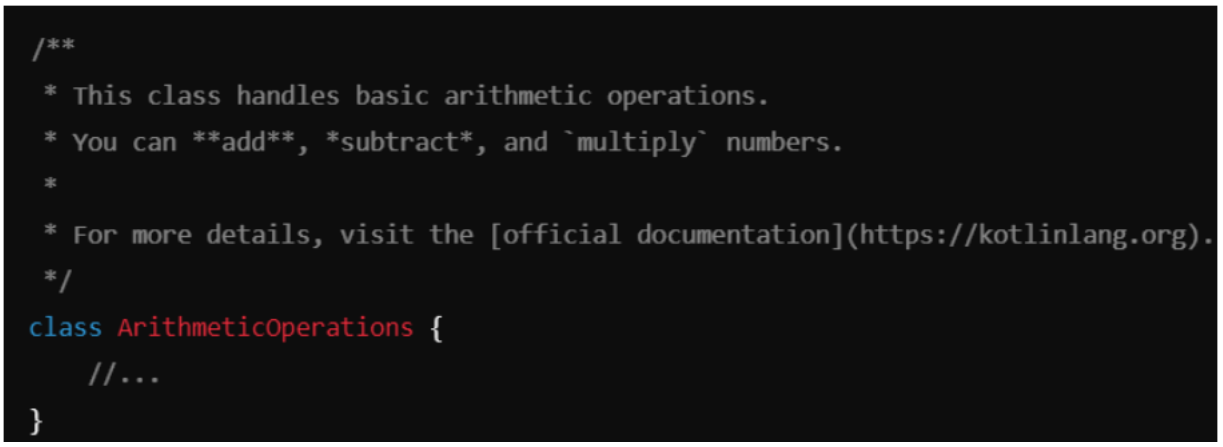

Figure 1: An example image

6 Dokka – Generating Documentation

Dokka is Kotlin's documentation engine, which allows developers to generate documentation in HTML, Javadoc, or Markdown format from KDoc comments. It is integrated with Gradle and can be easily added to a Kotlin project for automated documentation generation.

6.1 Steps to Use Dokka

1. Add the Dokka plugin to the Gradle configuration.
2. Run the following Gradle command to generate the documentation:

```
./gradlew dokkaHtml
```

3. The generated documentation will be saved in the build/dokka/html directory.

```
plugins {
    id 'org.jetbrains.dokka' version '1.7.20'
}
```

Figure 2: Adding the Dokka plugin in Gradle

```
./gradlew dokkaHtml
```

Figure 3: Running the Gradle command

7 Best Practices for Using KDoc

Through our exploration, we identified several best practices to ensure effective use of KDoc:

- **Clarity and Conciseness**: Documentation should be clear and to the point.
- **Meaningful Descriptions**: Tags such as @param and @return should provide detailed descriptions.
- **Consistency**: Consistent documentation across the codebase ensures readability.
- **Leverage Markdown**: Using Markdown elements helps emphasize important parts of the documentation.

8 Why Use KDoc for the *KII-KINBO* Online Supershop App?

For *KII-KINBO*, our online supershop app, KDoc offers several benefits:

- **Improved Collaboration**: KDoc ensures that the code is well-documented, making it easier for team members to understand and contribute.
- **Code Readability and Maintenance**: The codebase will remain well-organized, making it easier to maintain and scale.
- **Enhanced Debugging**: By documenting edge cases using @throws, developers can better understand and handle bugs.
- **Seamless Integration with Dokka**: Dokka generates polished documentation in various formats from KDoc comments.

- **Time-Saving**: Well-documented code reduces debugging and maintenance time in the long run.

9 Findings and Conclusion

Kotlin's KDoc is a powerful tool for documenting code, especially when combined with Dokka for generating polished documentation. It improves the quality of the codebase and enhances team collaboration.

For *KII-KINBO*, KDoc will be invaluable in documenting classes, functions, and properties to ensure that the app is easy to understand and maintain. With Dokka, we can generate well-organized HTML documentation, useful for future developers and stakeholders.

Incorporating KDoc into the development process will make *KII-KINBO* better organized, more scalable, and easier to maintain as it evolves.