# REPORT NO. 6: Coding Standard and Documentation Tool

## Course Code: CSE 404

## Course Title: Software Engineering and ISD Laboratory

Submitted by

SHANJIDA ALAM(ID: 353)

Submitted to

Dr. Md. MUSHFIQUE ANWAR, Professor
Dr. Md. HUMAYUN KABIR, Professor

Computer Science and Engineering
Jahangirnagar University
Dhaka, Bangladesh

September 27, 2024

# Contents

# Chapter 1

# Introduction

The documentation tool and coding standard are focused on whether the code is readable, maintainable, and easy to understand for other developers, both within a team. This report analyzes the key coding standard and documentation tool that have been selected for our project. In this report, I have applied our selected coding standard and documentation tool to the '`Multiplication`' class.

# Chapter 2

# Objectives

Based on the report on Coding Standard and Documentation, the following objectives are:

- Improve the code readability using the consistent coding style, including consistent spacing, indentation, and naming conventions.
- Improve the quality of Javadoc comments, including descriptive comments, tags, and formatting.
- Well-documented code helps to identify the bugs, as detailed comments and explanations make it easier to trace issues back to their source.
- To improve communication and teamwork.
- Documentation and coding standards provide the necessary context for understanding complex logic or architecture.

# Chapter 3

# Source Code: Multiplication Class

In this chapter, we will explore an example Java class that demonstrates coding standards and documentation practices. The class provided is called `Multiplication` and supports both `int` and `double` data types for multiplication.

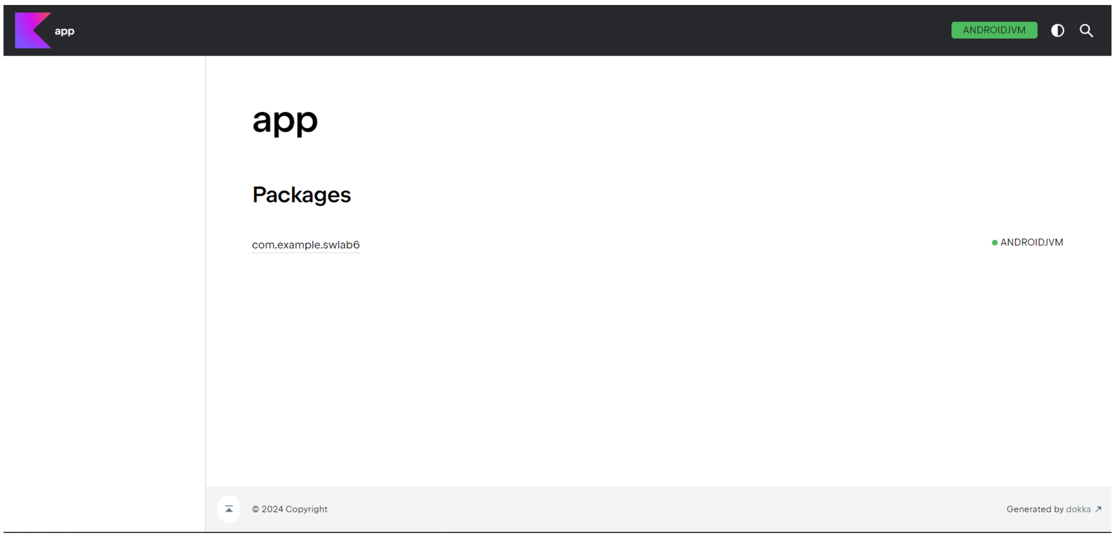
```java
package com.example.swlab6;

/**
 * The {@code Multiplication} class provides methods to multiply two
   numbers.
 * It supports both {@code int} and {@code double} data types.
 *
 * <p>This class contains four instance variables:</p>
 * <ul>
 *     <li>{@code int firstNumber} - an {@code int} representing the
   first integer to multiply</li>
 *     <li>{@code int secondNumber} - an {@code int} representing the
    second integer to multiply</li>
 *     <li>{@code double number1} - a {@code double} representing the
    first floating-point number to multiply</li>
 *     <li>{@code double number2} - a {@code double} representing the
    second floating-point number to multiply</li>
 * </ul>
 *
 * <p>This class offers constructors for initializing the numbers and
    methods for performing
 * multiplication of two integers or two doubles.</p>
 *
 * @author Shanjida Alam
 * @version 1.0
 * @since 26/09/2024
 */
```

```java
public class Multiplication {
    private int firstNumber;
    private int secondNumber;
    private double number1;
    private double number2;

    /**
     * Constructor that initializes the object with two {@code int}
    values to be multiplied.
     *
     * @param firstNumber  The first {@code int} to multiply, stored
    in the {@code firstNumber} variable.
     * @param secondNumber The second {@code int} to multiply, stored
     in the {@code secondNumber} variable.
     */
    public Multiplication(int firstNumber, int secondNumber) {
        this.firstNumber = firstNumber;
        this.secondNumber = secondNumber;
    }

    /**
     * Constructor that initializes the object with two {@code double
    } values to be multiplied.
     *
     * @param number1 The first {@code double} to multiply, stored in
     the {@code number1} variable.
     * @param number2 The second {@code double} to multiply, stored
    in the {@code number2} variable.
     */
    public Multiplication(double number1, double number2) {
        this.number1 = number1;
        this.number2 = number2;
    }

    /**
     * Multiplies two {@code int} values and returns the result.
     *
     * @param firstNumber  The first {@code int} to multiply.
     * @param secondNumber The second {@code int} to multiply.
     * @return The result of multiplying {@code firstNumber} and {
    @code secondNumber}, as an {@code int}.
     */
    public int multiplyTwoNumbers(int firstNumber, int secondNumber)
    {
        return firstNumber * secondNumber;
    }

    /**
     * Multiplies two {@code double} values and returns the result.
     *
     * @param firstNumber  The first {@code double} to multiply.
     * @param secondNumber The second {@code double} to multiply.
     * @return The result of multiplying {@code firstNumber} and {@code secondNumber}, as a {@code double}.
     */
    public double multiplyTwoNumbers(double firstNumber, double secondNumber) {
        return firstNumber * secondNumber;
    }
}
```

Listing 3.1: Multiplication Class in Java

# Chapter 4

# Image of the Generated Documentation



app

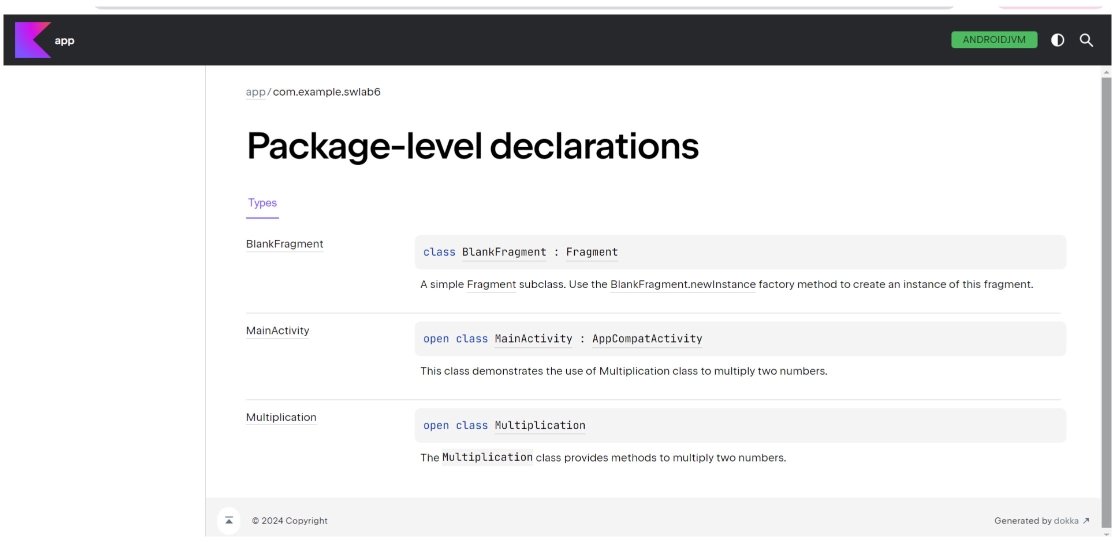
Packages

com.example.swlab6 ● ANDROIDJVM

© 2024 Copyright

Generated by dokka

Figure 4.1: Dokka-generated documentation for `com.example.swlab6` package


app ANDROIDJVM

app / com.example.swlab6

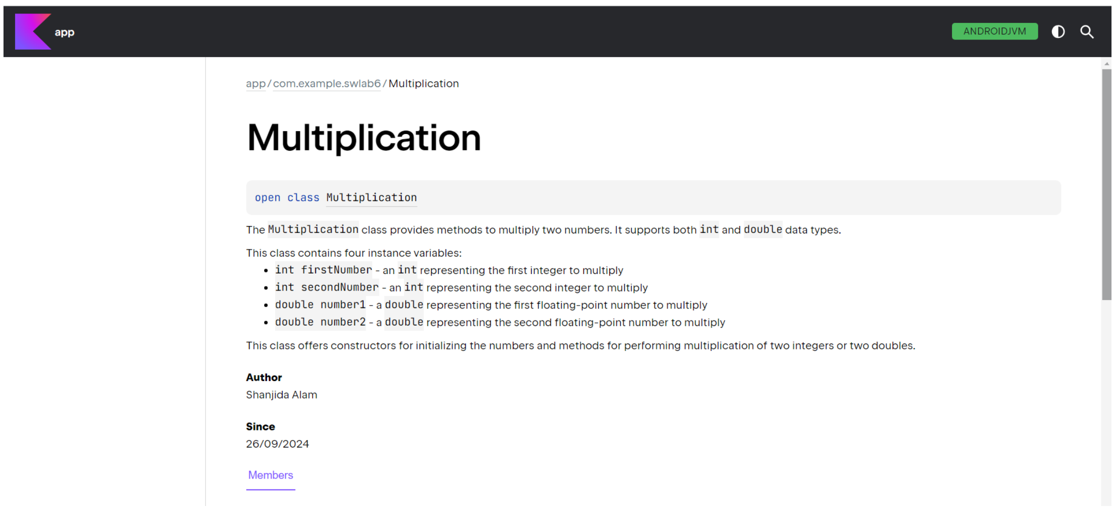
Package-level declarations

Types

| | |
|---|---|
| BlankFragment | `class BlankFragment : Fragment` A simple Fragment subclass. Use the BlankFragment.newInstance factory method to create an instance of this fragment. |
| MainActivity | `open class MainActivity : AppCompatActivity` This class demonstrates the use of Multiplication class to multiply two numbers. |
| Multiplication | `open class Multiplication` The `Multiplication` class provides methods to multiply two numbers. |

© 2024 Copyright  Generated by dokka

Figure 4.2: Package-level declarations in app documentation


app ANDROIDJVM

app / com.example.swlab6 / Multiplication

Multiplication

```
open class Multiplication
```

The `Multiplication` class provides methods to multiply two numbers. It supports both `int` and `double` data types.

This class contains four instance variables:
- `int firstNumber` - an `int` representing the first integer to multiply
- `int secondNumber` - an `int` representing the second integer to multiply
- `double number1` - a `double` representing the first floating-point number to multiply
- `double number2` - a `double` representing the second floating-point number to multiply

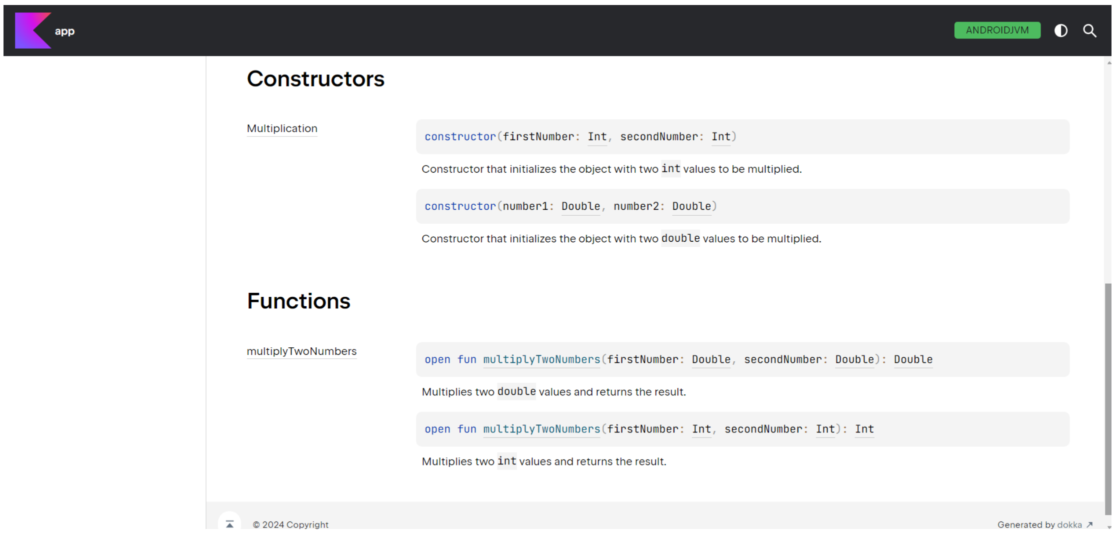
This class offers constructors for initializing the numbers and methods for performing multiplication of two integers or two doubles.

**Author**
Shanjida Alam

**Since**
26/09/2024

Members

Figure 4.3: `Multiplication` class documentation overview page



## Constructors

Multiplication

```
constructor(firstNumber: Int, secondNumber: Int)
```

Constructor that initializes the object with two `int` values to be multiplied.

```
constructor(number1: Double, number2: Double)
```

Constructor that initializes the object with two `double` values to be multiplied.

## Functions

multiplyTwoNumbers

```
open fun multiplyTwoNumbers(firstNumber: Double, secondNumber: Double): Double
```

Multiplies two `double` values and returns the result.

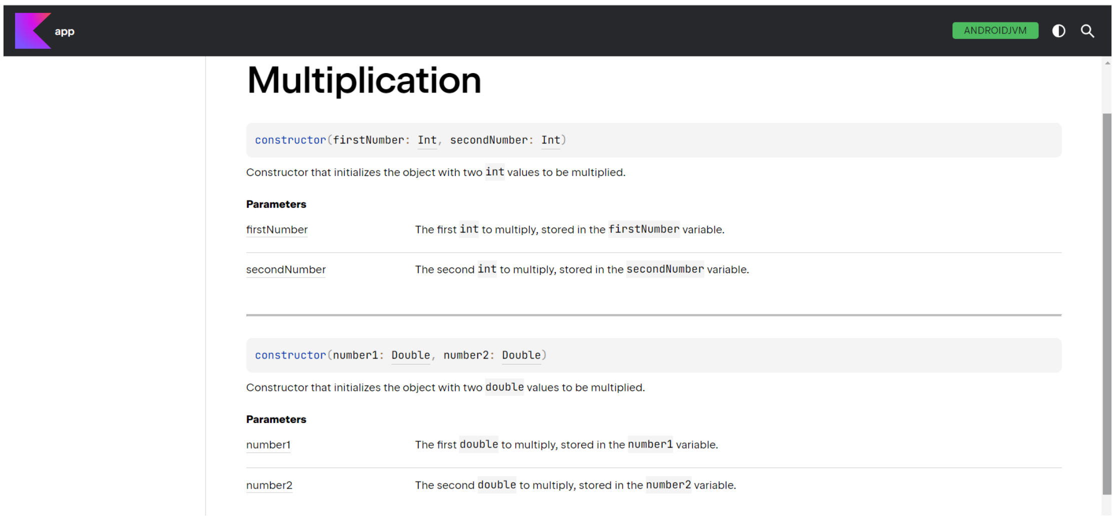
```
open fun multiplyTwoNumbers(firstNumber: Int, secondNumber: Int): Int
```

Multiplies two `int` values and returns the result.

© 2024 Copyright

Generated by dokka

Figure 4.4: Constructor and function details for the `Multiplication` class



## Multiplication

```
constructor(firstNumber: Int, secondNumber: Int)
```

Constructor that initializes the object with two `int` values to be multiplied.

**Parameters**

| | |
|---|---|
| firstNumber | The first `int` to multiply, stored in the `firstNumber` variable. |
| secondNumber | The second `int` to multiply, stored in the `secondNumber` variable. |

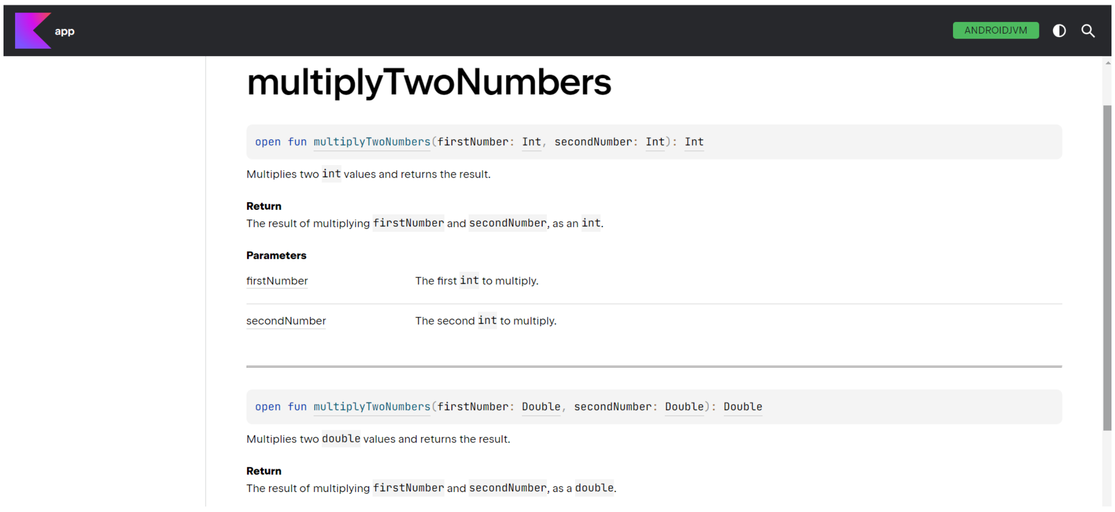
```
constructor(number1: Double, number2: Double)
```

Constructor that initializes the object with two `double` values to be multiplied.

**Parameters**

| | |
|---|---|
| number1 | The first `double` to multiply, stored in the `number1` variable. |
| number2 | The second `double` to multiply, stored in the `number2` variable. |

Figure 4.5: `Multiplication` class constructor details



## multiplyTwoNumbers

```
open fun multiplyTwoNumbers(firstNumber: Int, secondNumber: Int): Int
```

Multiplies two `int` values and returns the result.

**Return**
The result of multiplying `firstNumber` and `secondNumber`, as an `int`.

**Parameters**

| | |
|---|---|
| firstNumber | The first `int` to multiply. |
| secondNumber | The second `int` to multiply. |

```
open fun multiplyTwoNumbers(firstNumber: Double, secondNumber: Double): Double
```

Multiplies two `double` values and returns the result.

**Return**
The result of multiplying `firstNumber` and `secondNumber`, as a `double`.

Figure 4.6: Details of `multiplyTwoNumbers` function

# Chapter 5

# Coding Standard

In this part, I would have described which parts of the given code implement the coding standard.

## 5.1 Naming Conventions

The class follows the usual name standards used by Java:

- **Class Name:** The class name `Multiplication` follows **PascalCase**, where the first letter of each word is capitalized. This is the standard naming convention for Java classes.
- **Variable Names:** The instance variable names `firstNumber`, `secondNumber`, `number1`, and `number2` are in **camelCase**, where the first word is lowercase, and each subsequent word begins with a capital letter. These names are also descriptive and clarify the purpose of each variable (i.e., storing integer and floating-point numbers).
- **Function Names:** The function names `multiplyTwoNumbers(int, int)` and `multiplyTwoNumbers(double, double)` follow **camelCase** and are descriptive of their functionality. The method names clearly indicate that they multiply two numbers.

## 5.2 Layout Conventions

The class follows the usual layout conventions used by Java:

- **Indentation:** Use 4 spaces per indentation level. Do not use tabs.

- **Braces:** Always put the opening brace on the same line as the statement.

## 5.3 Member Order

The member order defines the structure inside a class, ensuring consistency across the codebase.Recommended Order Inside a Class:

- **Variables:** `firstNumber`,`secondNumber`,`number1`,`number2`
- **Constructors:** `public Multiplication(int firstNumber, int secondNumber)`
  `Multiplication(double number1, double number2)`
- **Functions:** `public int multiplyTwoNumbers(int firstNumber, int secondNumber)`
  `public double multiplyTwoNumbers(double firstNumber, double secondNumber)`

## 5.4 Code Comments

- **Function and Class Comments:** Each function/class should have a Javadoc comment to describe its purpose, parameters, and return value (for functions).Here is the segment of code where I have implemented the Javadoc comments:

```java
/**
 * Multiplies two {@code int} values and returns the result.
 *
 * @param firstNumber  The first {@code int} to multiply.
 * @param secondNumber The second {@code int} to multiply.
 * @return The result of multiplying {@code firstNumber} and {
   @code secondNumber}, as an {@code int}.
 */
public int multiplyTwoNumbers(int firstNumber, int secondNumber)
     {
    return firstNumber * secondNumber;
}
```

Listing 5.1: Function: multiplyTwoNumbers

# Chapter 6

# Documentation

In this part, I would have described which parts of the given code implement the Documentation.

## 6.1 Class-Level Documentation

The class is documented using Javadoc, a widely-used Java documentation tool **Dokka**. The class-level Javadoc provides:

- A brief description of the purpose of the class (`The Multiplication class provides methods to multiply two numbers`).
- A list of the instance variables used in the class (`int firstNumber, int secondNumber, double number1, and double number2`).
- An overview of the **constructors** and **functions** that the class offers. Here is the segment of code where I have implemented the class-level Javadoc:

```
/**
 * The {@code Multiplication} class provides methods to multiply
     two numbers.
 * It supports both {@code int} and {@code double} data types.
 *
 * <p>This class contains four instance variables:</p>
 * <ul>
 *     <li>{@code int firstNumber} - an {@code int} representing
     the first integer to multiply</li>
 *     <li>{@code int secondNumber} - an {@code int}
    representing the second integer to multiply</li>
 *     <li>{@code double number1} - a {@code double}
   representing the first floating-point number to multiply</li>
 *     <li>{@code double number2} - a {@code double}
   representing the second floating-point number to multiply</li
   >
 * </ul>
 *
 * <p>This class offers constructors for initializing the
   numbers and methods for performing
 * multiplication of two integers or two doubles.</p>
 *
 * @author Shanjida Alam
 * @version 1.0
 * @since 26/09/2024
 */
```

Listing 6.1: Class-Level Javadoc

## 6.2 Constructor-Level Documentation

Each constructor is documented, explaining the parameters passed to initialize the class's variables:

- **Integer constructor:** The constructor that initializes two integers `(firstNumber and secondNumber)` is clearly documented with an explanation of the parameters.
- **Double constructor:** Similarly, the constructor that initializes two double numbers `(number1 and number2)` is also documented with a description of the parameters. Here is the segment of code where I have implemented the constructor-level Javadoc:

```
    /**
     * Constructor that initializes the object with two {@code
   int} values to be multiplied.
     *
     * @param firstNumber  The first {@code int} to multiply,
   stored in the {@code firstNumber} variable.
     * @param secondNumber The second {@code int} to multiply,
   stored in the {@code secondNumber} variable.
     */
    public Multiplication(int firstNumber, int secondNumber) {
        this.firstNumber = firstNumber;
```

```java
        this.secondNumber = secondNumber;
    }

    /**
     * Constructor that initializes the object with two {@code
    double} values to be multiplied.
     *
     * @param number1 The first {@code double} to multiply,
    stored in the {@code number1} variable.
     * @param number2 The second {@code double} to multiply,
    stored in the {@code number2} variable.
     */
    public Multiplication(double number1, double number2) {
        this.number1 = number1;
        this.number2 = number2;
    }

```

Listing 6.2: Constructor-Level Javadoc

## 6.3 Function-Level Documentation

Each function is documented with a description of what it does, the parameters it takes, and the return value:

- **`multiplyTwoNumbers(int, int)` method:** This method is described as multiplying two integers and returning the result. Both input parameters `(firstNumber and secondNumber)` are explained, and the return type `(int)` is clarified.
- **`multiplyTwoNumbers(double, double)` method:** This method is similarly documented, explaining how it multiplies two double numbers and returns the result as a `double`. Here is the segment of code where I have implemented the function-level Javadoc:

```java
    /**
 * Multiplies two {@code int} values and returns the result.
 *
 * @param firstNumber  The first {@code int} to multiply.
 * @param secondNumber The second {@code int} to multiply.
 * @return The result of multiplying {@code firstNumber} and {
   @code secondNumber}, as an {@code int}.
 */
```

```java
public int multiplyTwoNumbers(int firstNumber, int secondNumber) {
    return firstNumber * secondNumber;
}

 /**
     * Multiplies two {@code double} values and returns the result.
     *
     * @param firstNumber  The first {@code double} to multiply.
     * @param secondNumber The second {@code double} to multiply.
     * @return The result of multiplying {@code firstNumber} and {@code secondNumber}, as a {@code double}.
     */
    public double multiplyTwoNumbers(double firstNumber, double secondNumber) {
        return firstNumber * secondNumber;
    }


```

Listing 6.3: Function-Level Javadoc

# Chapter 7

# Conclusion

The multiplication class demonstrates good alignment with coding standards and documentation best practices. The code is well-structured, readable, and maintainable, with clear variable and method names. The use of Javadoc comments provides clear and detailed documentation for all components of the class.

To make the class even better, adding error handling for things like integer overflow would be helpful. Overall, the class is a great example of how following coding standards and proper documentation can lead to well-written, high-quality software.