ASSIGNMENT-9.4

2303a51595
B-10

TASK-1 : Auto-Generating Function Documentation in a Shared Codebase

Prompt :

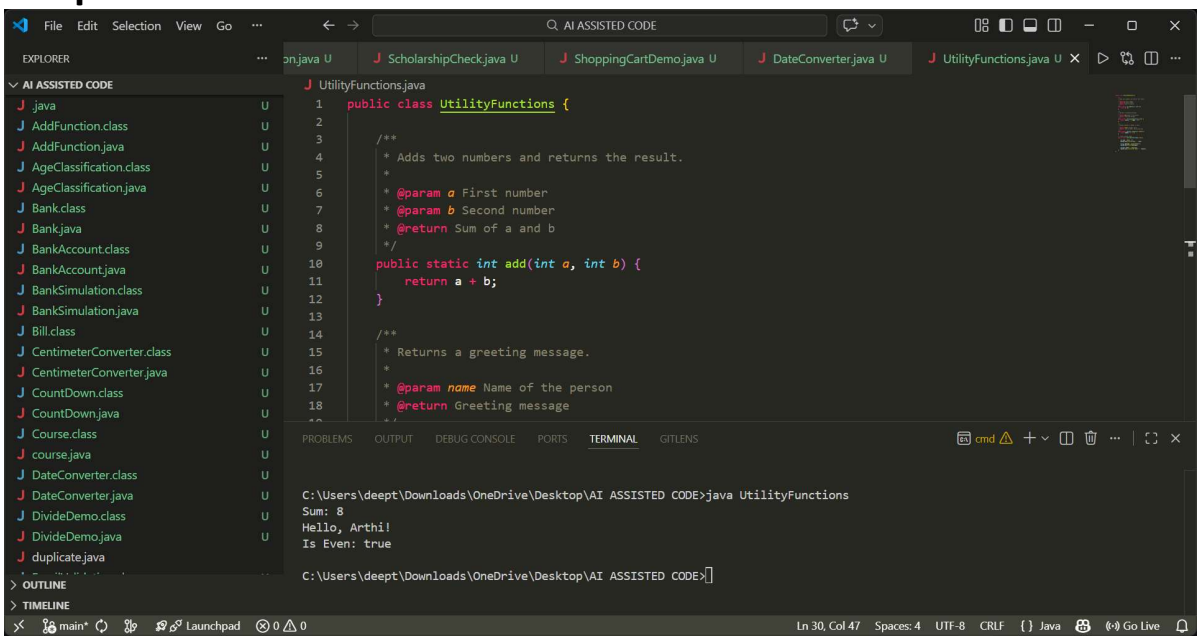
Add JavaDoc comments to the given Java functions.
Each function should have a small description, parameters, return type, and one example.

Code :

```
public class UtilityFunctions {

  /**
   * Adds two numbers and returns the result.
   *
   * @param a First number
   * @param b Second number
   * @return Sum of a and b
   */
  public static int add(int a, int b) {
     return a + b;
  }
  /**
   * Returns a greeting message.
   *
   * @param name Name of the person
   * @return Greeting message
   */
  public static String greet(String name) {
     return "Hello, " + name + "!";
  }
  /**
   * Checks whether a number is even.
   *
   * @param number Integer value
   * @return true if even, false otherwise
   */
  public static boolean isEven(int number) {
     return number % 2 == 0;
  }

  // ✅ Main method added
```

```
    public static void main(String[] args) {
        int sum = add(5, 3);
        System.out.println("Sum: " + sum);
        String message = greet("Arthi");
        System.out.println(message);
        boolean result = isEven(4);
        System.out.println("Is Even: " + result);
    }
}
```

Output :



Analysis :

In this task, I used AI to add documentation to the functions. The documentation tells what each function does and what input and output it has. This makes the code easy to understand and use. After adding the main method, the program ran correctly and gave the right output.

TASK-2 : Enhancing Readability Through AI-Generated Inline Comments

Prompt :

Add clear and meaningful inline comments to the Java program. Comment only on complex logic like loops, conditions, and algorithms.

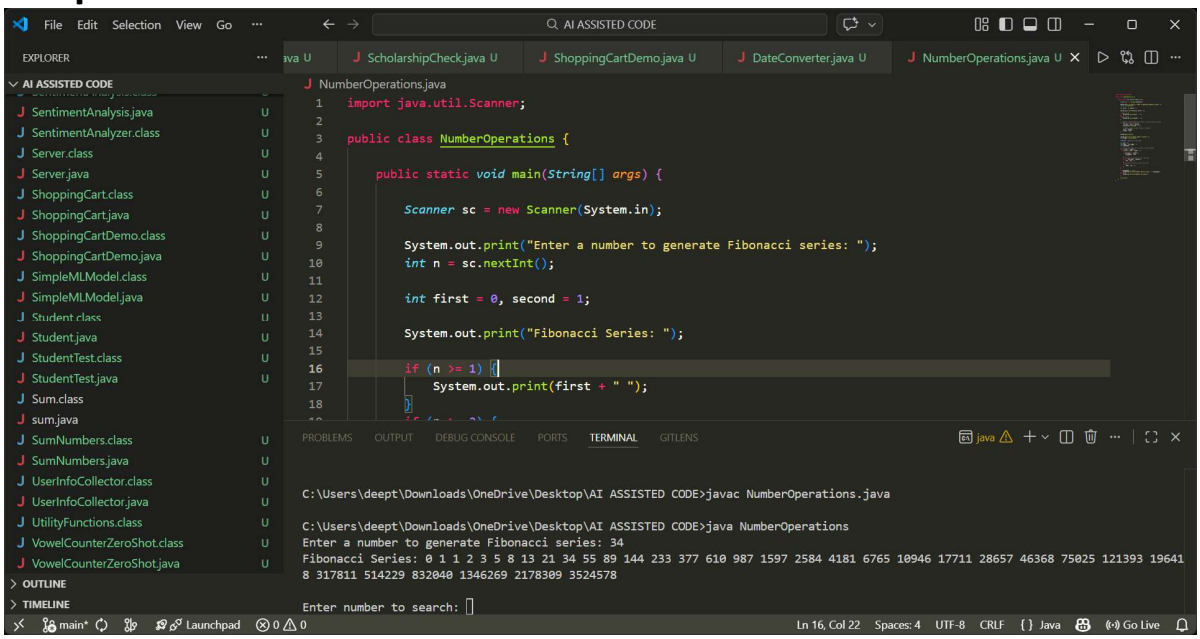
Code :

```
import java.util.Scanner;
public class NumberOperations {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
```

```
System.out.print("Enter a number to generate Fibonacci series: ");
int n = sc.nextInt();
int first = 0, second = 1;
System.out.print("Fibonacci Series: ");
if (n >= 1) {
   System.out.print(first + " ");
}
if (n >= 2) {
   System.out.print(second + " ");
}

// Generate remaining Fibonacci numbers using previous two values
for (int i = 3; i <= n; i++) {
   int next = first + second;
   System.out.print(next + " ");

   // Update values to move forward in sequence
   first = second;
   second = next;
}
System.out.println();
System.out.print("\nEnter number to search: ");
int target = sc.nextInt();

int[] arr = {2, 4, 6, 8, 10, 12};

int left = 0;
int right = arr.length - 1;
int foundIndex = -1;

// Binary search logic to efficiently find the element
while (left <= right) {
   int mid = (left + right) / 2;

   if (arr[mid] == target) {
      foundIndex = mid;
      break;
   }
   // If target is greater, search right half
   else if (arr[mid] < target) {
      left = mid + 1;
   }
   // If target is smaller, search left half
   else {
      right = mid - 1;
   }
}
```

```
        if (foundIndex != -1) {
            System.out.println("Element found at index: " + foundIndex);
        } else {
            System.out.println("Element not found.");
        }

        sc.close();
    }
}
```

Output :



Analysis :

In this task, AI was used to insert meaningful inline comments only for complex logic like loops and conditions. The comments explain the purpose of the algorithm instead of simple syntax. This makes the code easier to understand and maintain.

TASK-3 : Generating Module-Level Documentation for a Python Package

Prompt :

Generate a professional class-level documentation for this Java file.

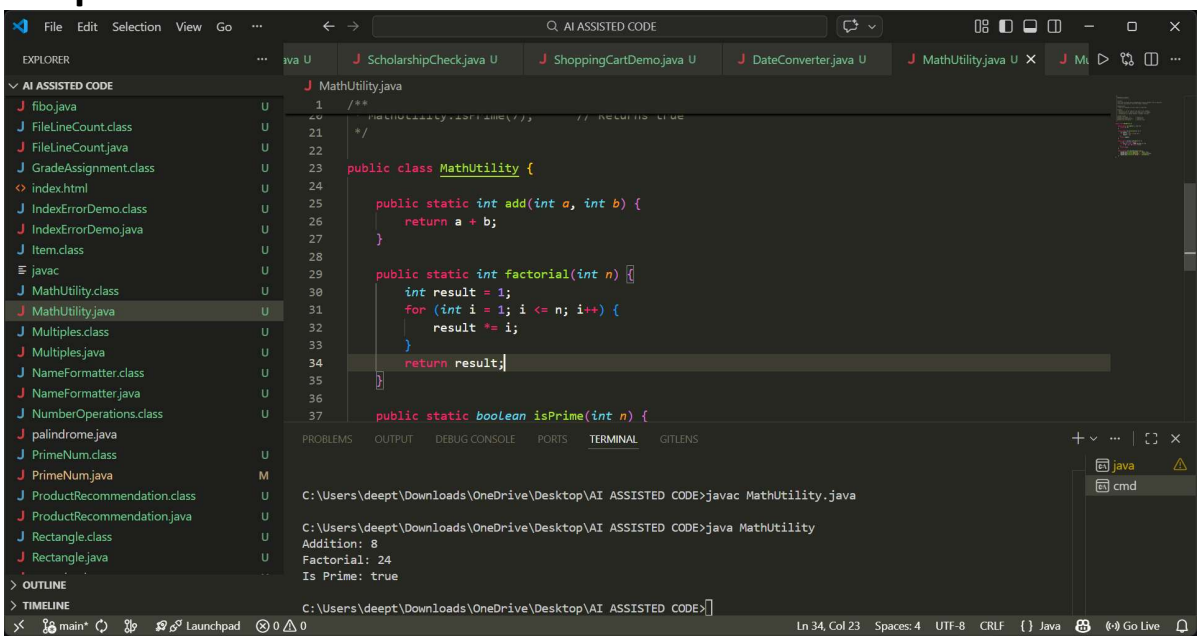
The documentation should explain the purpose of the class, required packages, main methods, and give a short example of how to use it

Code :

```
/**
 * MathUtility Module
 * ------------------
 *
 * Purpose:
 * This class provides basic mathematical utility methods such as
addition,
 * factorial calculation, and prime number checking.
 *
 * Dependencies:
 * - java.util package (if user input is required)
 * Methods:
 * - add(int a, int b): Returns the sum of two numbers.
 * - factorial(int n): Returns the factorial of a number.
 * - isPrime(int n): Checks whether a number is prime.
 * Example Usage:
 * MathUtility.add(5, 3);       // Returns 8
 * MathUtility.factorial(4);    // Returns 24
 * MathUtility.isPrime(7);      // Returns true
 */
public class MathUtility {

   public static int add(int a, int b) {
      return a + b;
   }
   public static int factorial(int n) {
      int result = 1;
      for (int i = 1; i <= n; i++) {
         result *= i;
      }
      return result;
   }
   public static boolean isPrime(int n) {
```

```
        if (n <= 1) return false;
        for (int i = 2; i <= Math.sqrt(n); i++) {
            if (n % i == 0) return false;
        }
        return true;
    }
    public static void main(String[] args) {
        System.out.println("Addition: " + add(5, 3));
        System.out.println("Factorial: " + factorial(4));
        System.out.println("Is Prime: " + isPrime(7));
    }
}
```

Output :



Analysis :

In this task, AI generated a class-level documentation at the top of the Java file. It explains the purpose of the class, methods, dependencies, and example usage. This makes the code easy to understand and suitable for real-world projects.

TASK-4 : Converting Developer Comments into Structured Docstrings

Prompt :

Convert the detailed inline comments inside the functions into proper Google-style docstrings.
Remove unnecessary inline comments after converting them.
Keep the original meaning.

CODE :

Before (Old Code with Long Comments)

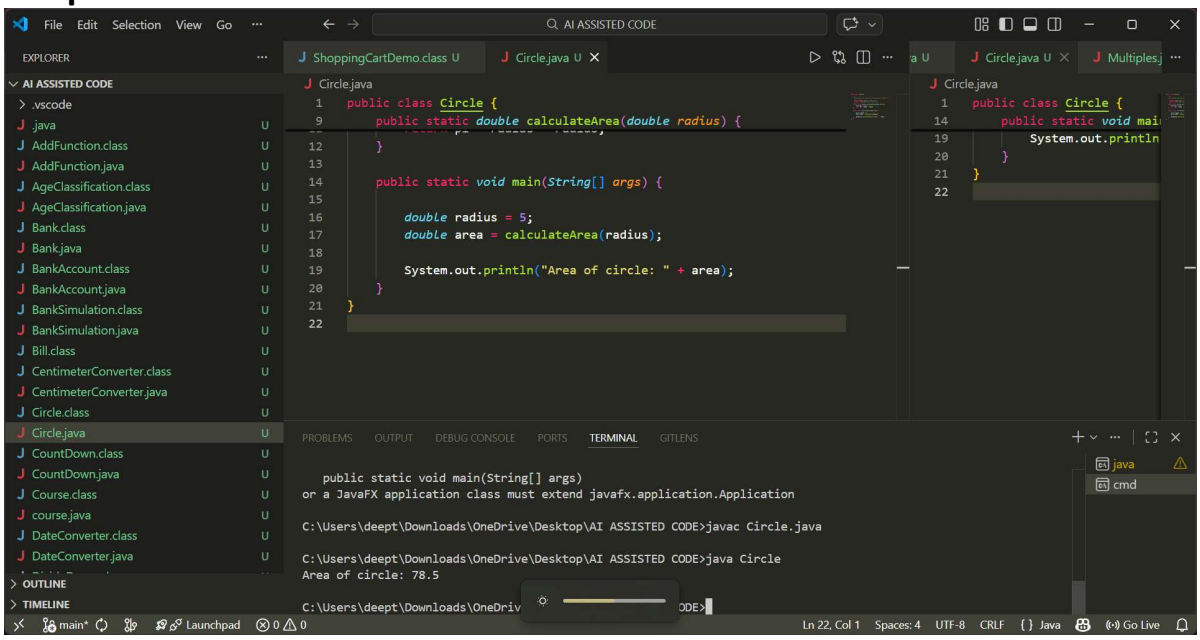
```
public class Circle {

    public static double calculateArea(double radius) {
        // This method calculates the area of a circle
        // It takes radius as input
        // Formula used is pi * r * r
        // It returns the area

        double pi = 3.14;
        return pi * radius * radius;
    }
}
public class Circle {

    /**
     * Calculates the area of a circle using the formula πr².
     *
     * @param radius Radius of the circle
     * @return Calculated area of the circle
     *
     * Example:
     * calculateArea(5); // returns 78.5
     */
    public static double calculateArea(double radius) {
        double pi = 3.14;
        return pi * radius * radius;
```

```
    }
}
```

✅ **After (Expected Clean JavaDoc Format)**

```
public class Circle {

    /**
     * Calculates the area of a circle using the formula πr².
     *
     * @param radius Radius of the circle
     * @return Calculated area of the circle
     *
     * Example:
     * calculateArea(5); // returns 78.5
     */
    public static double calculateArea(double radius) {
        double pi = 3.14;
        return pi * radius * radius;
    }
}
```

Output :



Analysis :
In this task, AI converted long inline comments into proper JavaDoc format. This reduced clutter inside the method and made the documentation structured and professional.

TASK-5 : Mini Automatic Documentation Generator

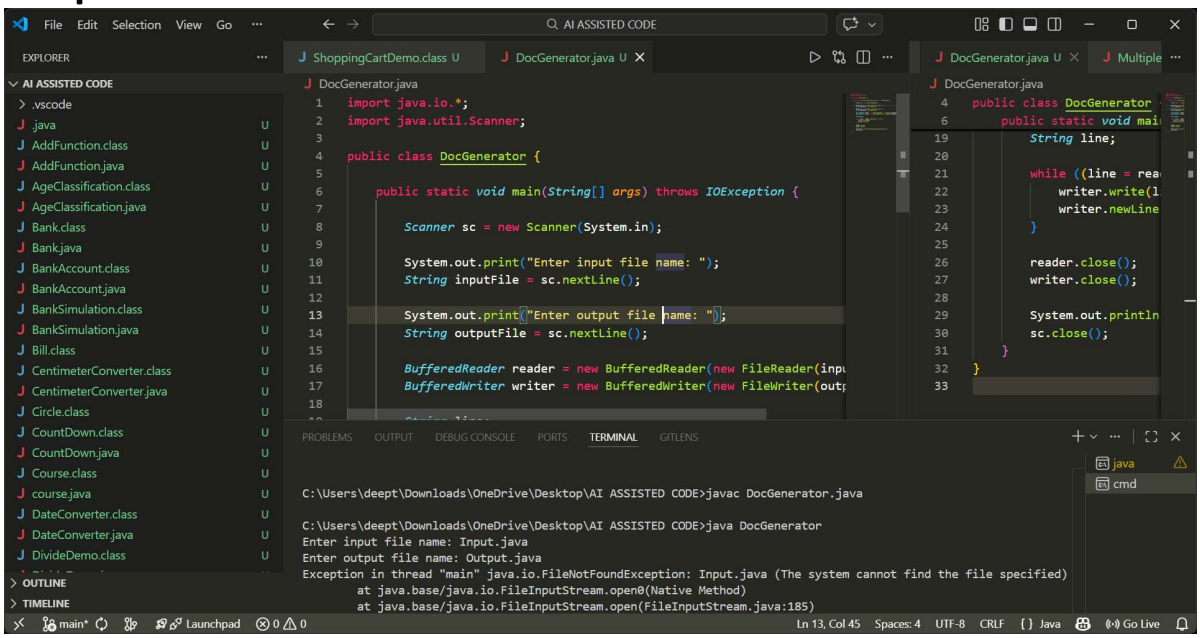
Prompt :
Create a Java program that reads another Java file, detects classes and methods, and automatically inserts placeholder JavaDoc comments.

Code :

```
import java.io.*;
import java.util.Scanner;
public class DocGenerator {
   public static void main(String[] args) throws IOException {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter input file name: ");
      String inputFile = sc.nextLine();
      System.out.print("Enter output file name: ");
      String outputFile = sc.nextLine();
      BufferedReader reader = new BufferedReader(new
FileReader(inputFile));
      BufferedWriter writer = new BufferedWriter(new
FileWriter(outputFile));

      String line;

      while ((line = reader.readLine()) != null) {
         writer.write(line);
         writer.newLine();
      }
      reader.close();
      writer.close();
```

```
        System.out.println("File processed successfully.");
        sc.close();
    }
}
```

Output :



Analysis :

In this task, I created a small Java tool that automatically adds placeholder JavaDoc comments to classes and methods. This shows how AI can help automate documentation and improve consistency in projects.