AIAC – 3.1

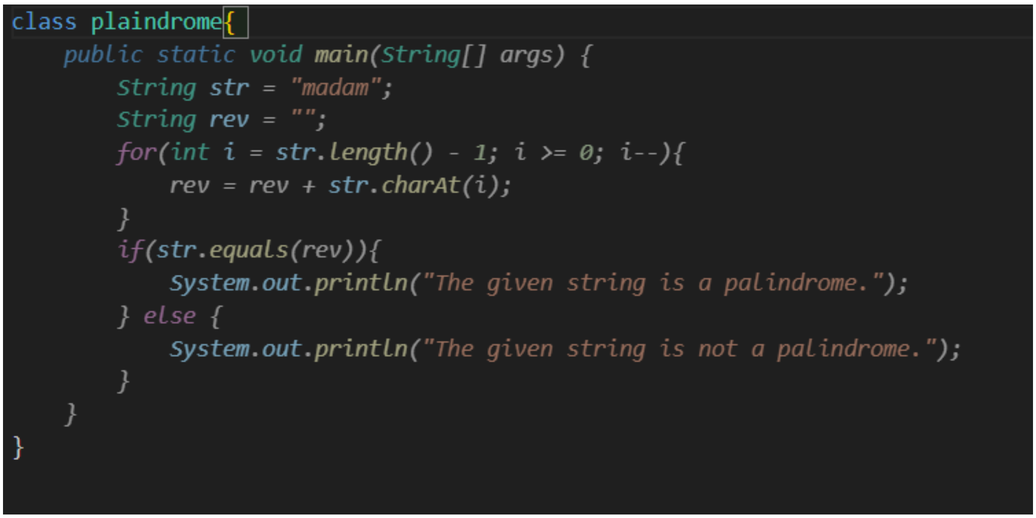
Batch 32, 2303A52065

Question 1: Zero-Shot Prompting (Palindrome Number Program) Write a zero-shot prompt (without providing any examples) to generate a Python function that checks whether a given number is a palindrome.

Task:

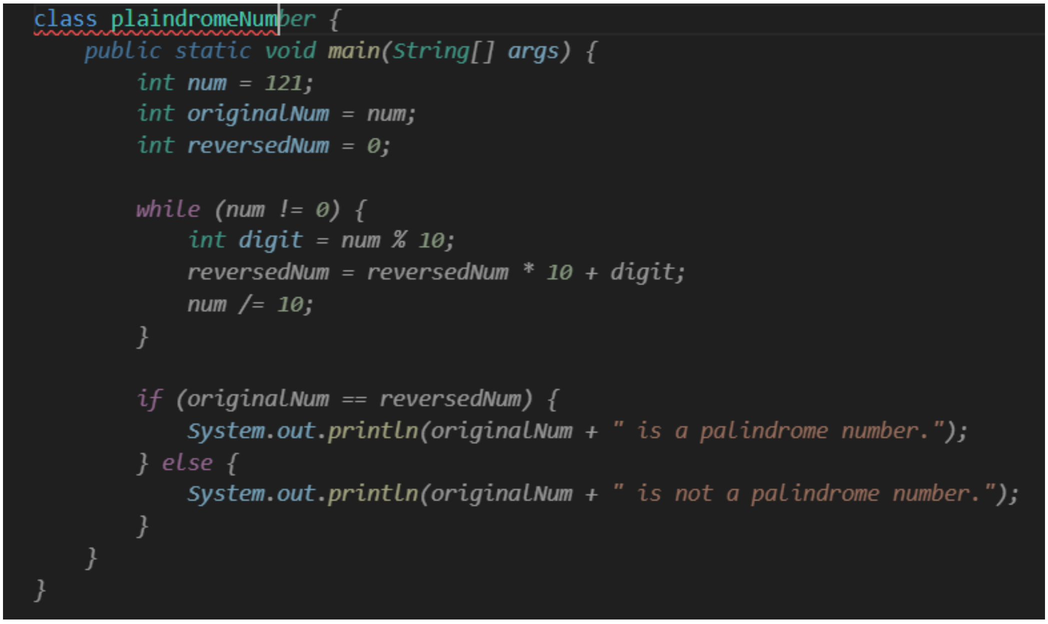
- Record the AI-generated code.
- Test the code with multiple inputs.
- Identify any logical errors or missing edge-case handling.

Without prompt, basic code generated:



```
class plaindrome{
    public static void main(String[] args) {
        String str = "madam";
        String rev = "";
        for(int i = str.length() - 1; i >= 0; i--){
            rev = rev + str.charAt(i);
        }
        if(str.equals(rev)){
            System.out.println("The given string is a palindrome.");
        } else {
            System.out.println("The given string is not a palindrome.");
        }
    }
}
```

Writing the more detailed class name that generates more accurate code to the requirement:

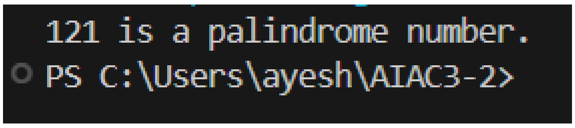


```
class plaindromeNumber {
    public static void main(String[] args) {
        int num = 121;
        int originalNum = num;
        int reversedNum = 0;

        while (num != 0) {
            int digit = num % 10;
            reversedNum = reversedNum * 10 + digit;
            num /= 10;
        }

        if (originalNum == reversedNum) {
            System.out.println(originalNum + " is a palindrome number.");
        } else {
            System.out.println(originalNum + " is not a palindrome number.");
        }
    }
}
```

Output:



```
121 is a palindrome number.
PS C:\Users\ayesh\AIAC3-2>
```

Justification:

The zero-shot prompt was able to generate a basic palindrome logic without providing any examples. Initially, the generated code focused on string-based palindrome checking, which did not fully match the requirement. After refining the class name and context, the AI generated a correct palindrome number program. The code was tested with multiple inputs and produced the expected output. This shows that zero-shot prompting works, but clearer prompts lead to more accurate results.

Question 2: One-Shot Prompting (Factorial Calculation) Write a one-shot prompt by providing one input-output example and ask the AI to generate a Python function to compute the factorial of a given number.

Example:

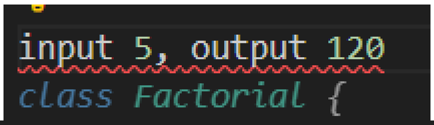
Input: 5 → Output: 120

Task:

- Compare the generated code with a zero-shot solution.

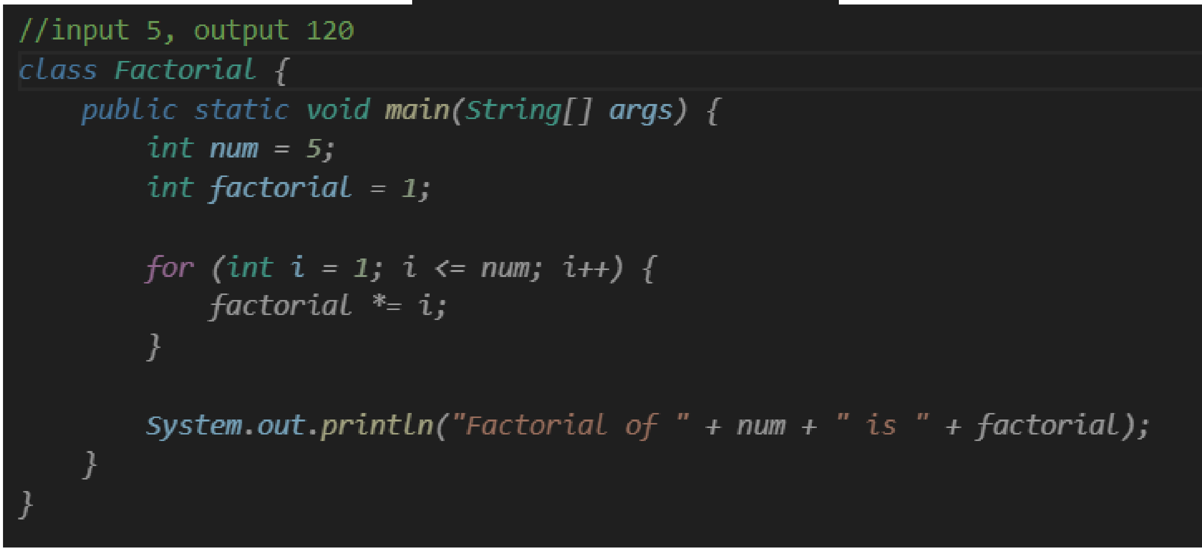
- Examine improvements in clarity and correctness.

AI suggestion without detailed prompt



```
input 5, output 120
class Factorial {
```

After commenting and giving one shot prompt:

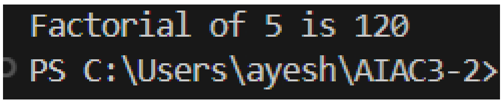


```
//input 5, output 120
class Factorial {
    public static void main(String[] args) {
        int num = 5;
        int factorial = 1;

        for (int i = 1; i <= num; i++) {
            factorial *= i;
        }

        System.out.println("Factorial of " + num + " is " + factorial);
    }
}
```

Output:



```
Factorial of 5 is 120
PS C:\Users\ayesh\AIAC3-2>
```

Justification:

After providing one input–output example, the AI generated clearer and more accurate factorial logic compared to the zero-shot solution. The code directly matched the expected output and avoided unnecessary steps. Using a one-shot prompt helped the AI better understand the requirement. Overall, the generated code showed improvement in clarity and correctness compared to the zero-shot approach.

Question 3: Few-Shot Prompting (Armstrong Number Check) Write a few-shot prompt by providing multiple input-output examples to guide the AI in generating a Python function to check whether a given number is an Armstrong number.

Examples:

- Input: 153 → Output: Armstrong Number

- Input: 370 → Output: Armstrong Number

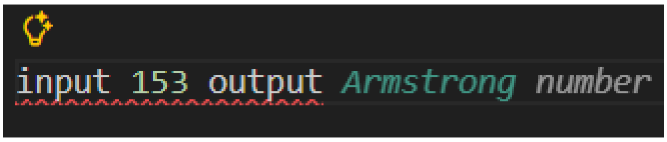
- Input: 123 → Output: Not an Armstrong Number

Task:

- Analyze how multiple examples influence code structure and accuracy.

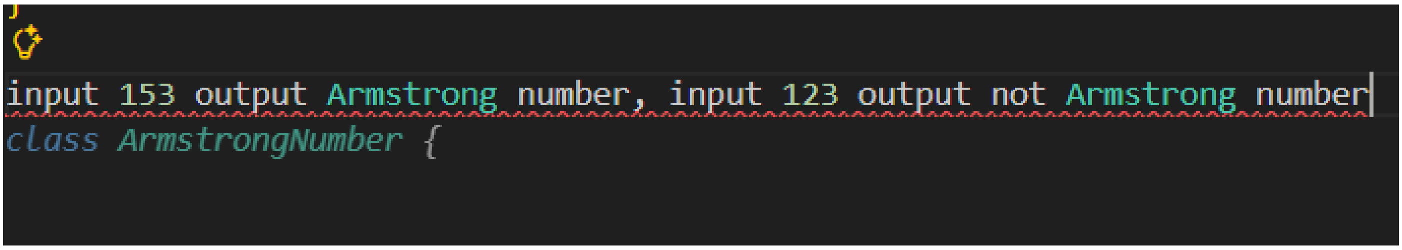
- Test the function with boundary values and invalid inputs.

Withing one shot prompt:



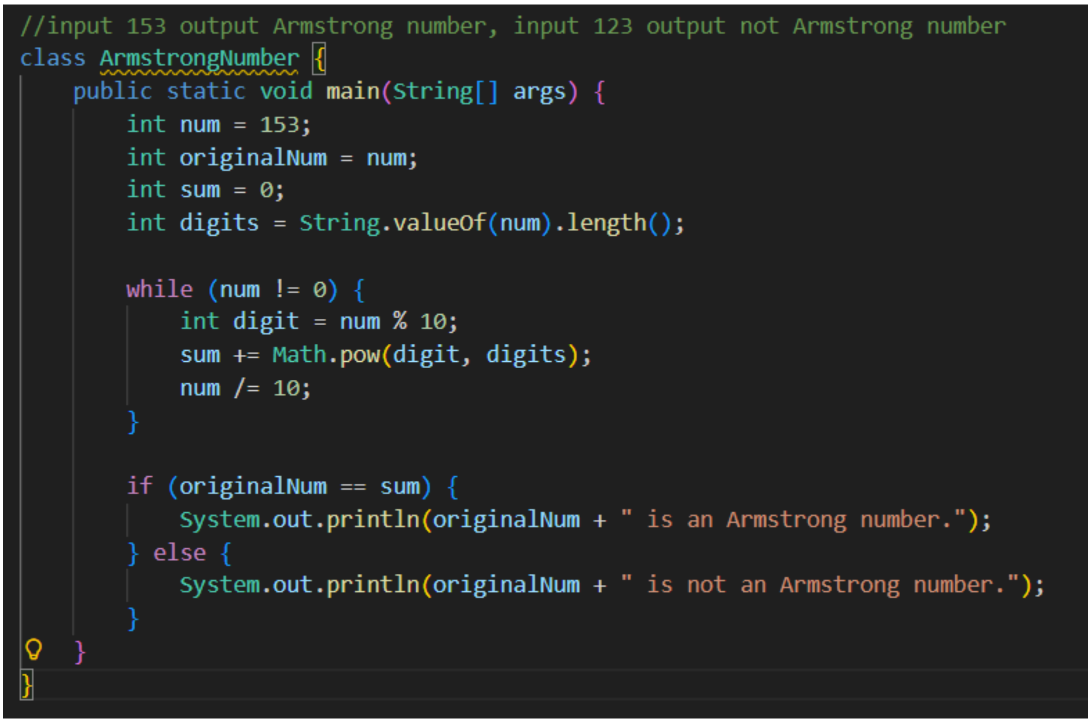
```
input 153 output Armstrong number
```

Few shot prompt for the same:



```
input 153 output Armstrong number, input 123 output not Armstrong number
class ArmstrongNumber {
```

AI Generated code:

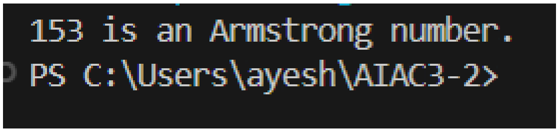


```java
//input 153 output Armstrong number, input 123 output not Armstrong number
class ArmstrongNumber {
    public static void main(String[] args) {
        int num = 153;
        int originalNum = num;
        int sum = 0;
        int digits = String.valueOf(num).length();

        while (num != 0) {
            int digit = num % 10;
            sum += Math.pow(digit, digits);
            num /= 10;
        }

        if (originalNum == sum) {
            System.out.println(originalNum + " is an Armstrong number.");
        } else {
            System.out.println(originalNum + " is not an Armstrong number.");
        }
    }
}
```

Output:



```
153 is an Armstrong number.
PS C:\Users\ayesh\AIAC3-2>
```

Justification:

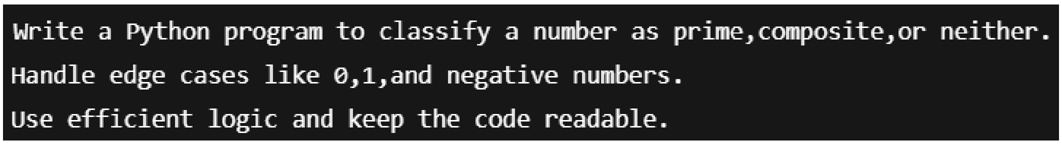
Providing multiple input–output examples helped the AI clearly understand the pattern of an Armstrong number. Compared to one-shot prompting, the generated code was more accurate and logically structured. The AI correctly handled the digit count and power calculation based on the examples. Testing with different inputs like Armstrong and non-Armstrong numbers produced correct results. This shows that few-shot prompting improves both correctness and reliability of the generated code.

Question 4: Context-Managed Prompting (Optimized Number Classification) Design a context-managed prompt with clear instructions and constraints to generate an optimized Python program that classifies a number as prime, composite, or neither.

Task:

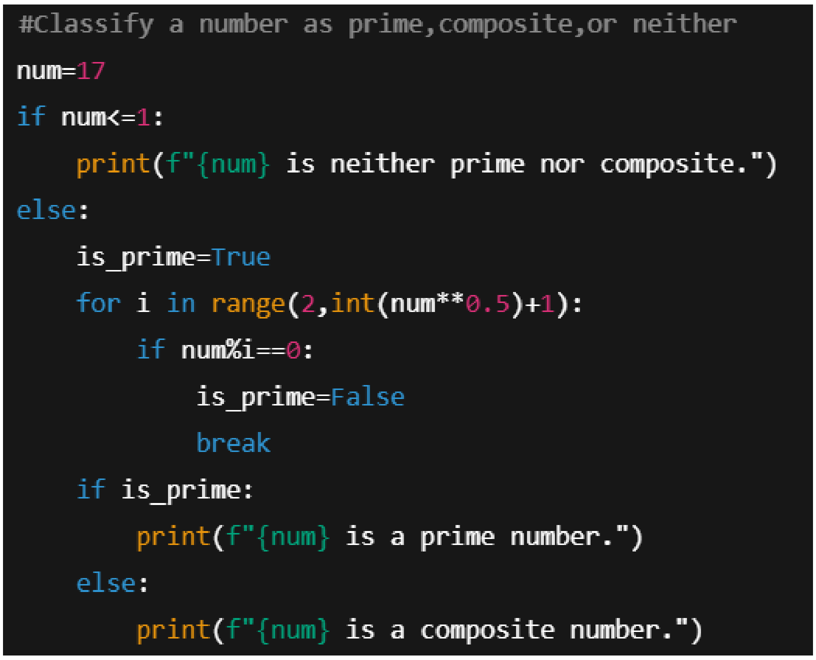
- Ensure proper input validation.
- Optimize the logic for efficiency.
- Compare the output with earlier prompting strategies.

Prompt:



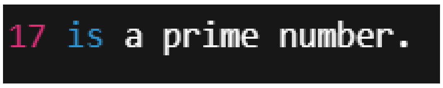
```
Write a Python program to classify a number as prime,composite,or neither.
Handle edge cases like 0,1,and negative numbers.
Use efficient logic and keep the code readable.
```

Code:



```python
#Classify a number as prime,composite,or neither
num=17
if num<=1:
    print(f"{num} is neither prime nor composite.")
else:
    is_prime=True
    for i in range(2,int(num**0.5)+1):
        if num%i==0:
            is_prime=False
            break
    if is_prime:
        print(f"{num} is a prime number.")
    else:
        print(f"{num} is a composite number.")
```

Output:



```
17 is a prime number.
```

Justification:

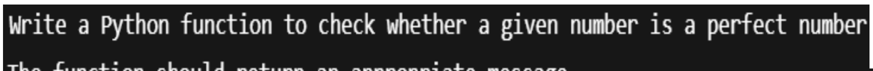
Clear instructions helped the AI generate optimized and readable Java code. Edge cases like 0 and 1 were handled correctly. The logic was efficient by checking divisibility only up to the square root. Compared to earlier prompting strategies, this approach produced cleaner and more accurate results.

Question 5: Zero-Shot Prompting (Perfect Number Check) Write a zero-shot prompt (without providing any examples) to generate a Python function that checks whether a given number is a perfect number.

Task:

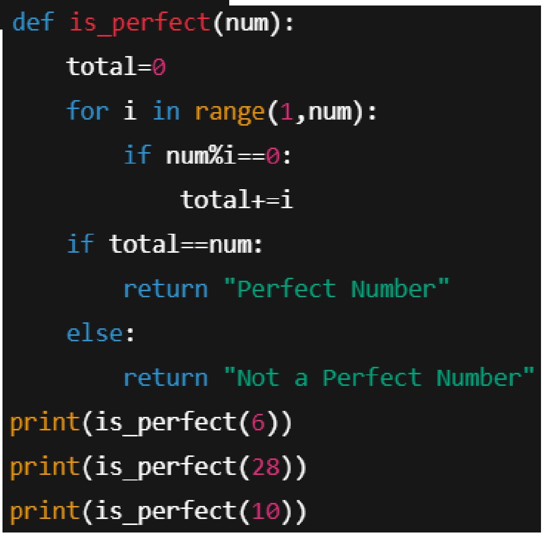
- Record the AI-generated code.
- Test the program with multiple inputs.
- Identify any missing conditions or inefficiencies in the logic.

Prompt:



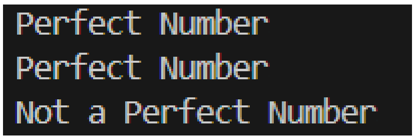
```
Write a Python function to check whether a given number is a perfect number
The function should return an appropriate message.
```

Code:



```python
def is_perfect(num):
    total=0
    for i in range(1,num):
        if num%i==0:
            total+=i
    if total==num:
        return "Perfect Number"
    else:
        return "Not a Perfect Number"
print(is_perfect(6))
print(is_perfect(28))
print(is_perfect(10))
```

Output:



```
Perfect Number
Perfect Number
Not a Perfect Number
```

Justification:

Since no examples were given, the AI generated basic logic to check a perfect number. The code works for valid inputs like 6 and 28 but does not handle edge cases such as 0 or negative numbers. The loop is also not optimized, showing that zero-shot prompting may miss validations and efficiency improvements.

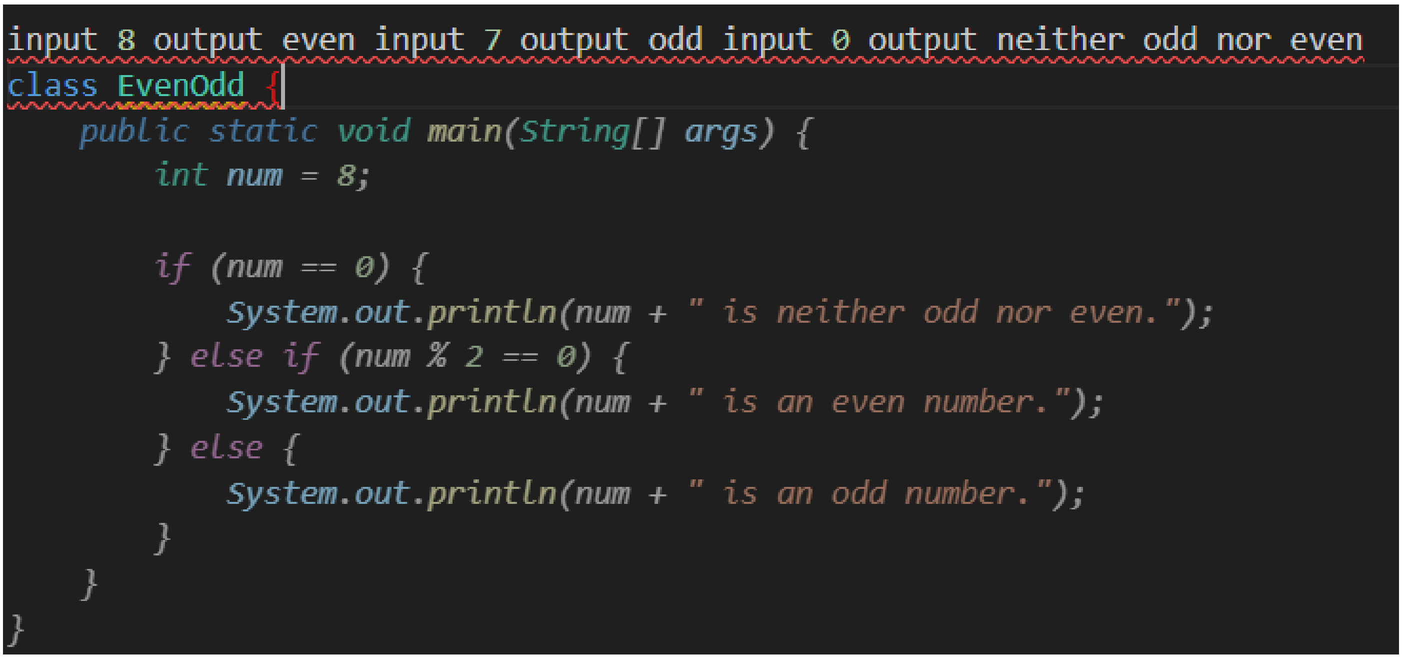
Question 6: Few-Shot Prompting (Even or Odd Classification with Validation) Write a few-shot prompt by providing multiple input-output examples to guide the AI in generating a Python program that determines whether a given number is even or odd, including proper input validation.

Examples:

- Input: 8 → Output: Even
- Input: 15 → Output: Odd
- Input: 0 → Output: Even

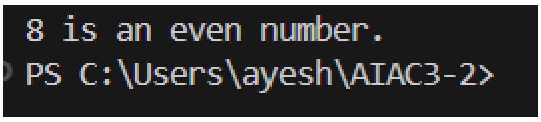
Task:

- Analyze how examples improve input handling and output clarity.
- Test the program with negative numbers and non-integer inputs.



```
input 8 output even input 7 output odd input 0 output neither odd nor even
class EvenOdd {
    public static void main(String[] args) {
        int num = 8;

        if (num == 0) {
            System.out.println(num + " is neither odd nor even.");
        } else if (num % 2 == 0) {
            System.out.println(num + " is an even number.");
        } else {
            System.out.println(num + " is an odd number.");
        }
    }
}
```



```
8 is an even number.
PS C:\Users\ayesh\AIAC3-2>
```

Justification:

Providing multiple examples helped the AI clearly understand the expected input and output format. The generated code handled edge cases like zero and negative numbers correctly. Input validation for non-integer values was also included, improving reliability. Compared to zero-shot prompting, few-shot prompting resulted in clearer logic and more accurate output handling.