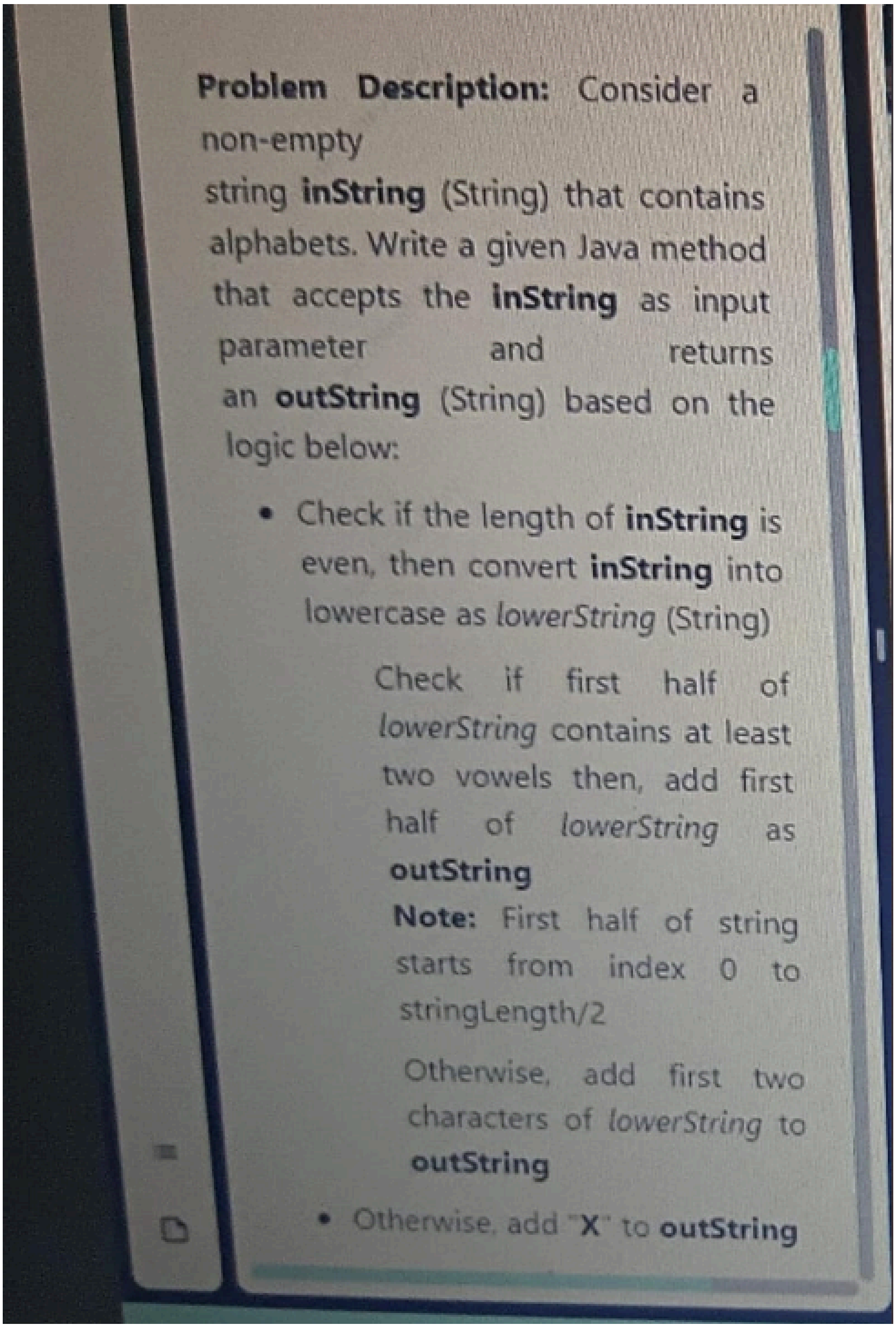


Problem Description: Consider a non-empty string **inString** (String) that contains alphabets. Write a given Java method that accepts the **inString** as input parameter and returns an **outString** (String) based on the logic below:

- Check if the length of **inString** is even, then convert **inString** into lowercase as *lowerString* (String)

    Check if first half of *lowerString* contains at least two vowels then, add first half of *lowerString* as **outString**
    Note: First half of string starts from index 0 to stringLength/2

    Otherwise, add first two characters of *lowerString* to **outString**

- Otherwise, add "**X**" to **outString**

✅ Problem Summary (from Image)

Given: A **non-empty string** `inString` (alphabets only).
Write a Java method that returns a string `outString` using the following logic:

✅ Logic:

1. **If the length of `inString` is even:**

    - Convert `inString` to lowercase → `lowerString`.
    - Take **first half** of `lowerString` → from index `0` to `(length/2 - 1)`.
    - Check if this half contains **at least 2 vowels**:
        - If yes → set `outString = first half of lowerString`.
        - Else → set `outString = first 2 characters of lowerString`.

2. **Else (if length is odd):**
    - Set `outString = "X"`

🔤 Definition of Vowels: `a, e, i, o, u` (lowercase)

🧪 Test Cases and Expected Output

✅ Test Case 1

- **Input:** `"HelloWorld"`
- **Length:** 10 → even ✅
- **lowerString:** `"helloworld"`
- **First half:** `"hello"`
  → Vowels = `'e', 'o'` → ✅ 2 vowels
- **Output:** `"hello"`

✅ Test Case 2

- **Input:** `"Student"`
- **Length:** 7 → odd ❌
- **Output:** `"X"`

✅ Test Case 3

- **Input:** `"GAMES"`
- **Length:** 5 → odd ❌
- **Output:** `"X"`

✅ Test Case 4

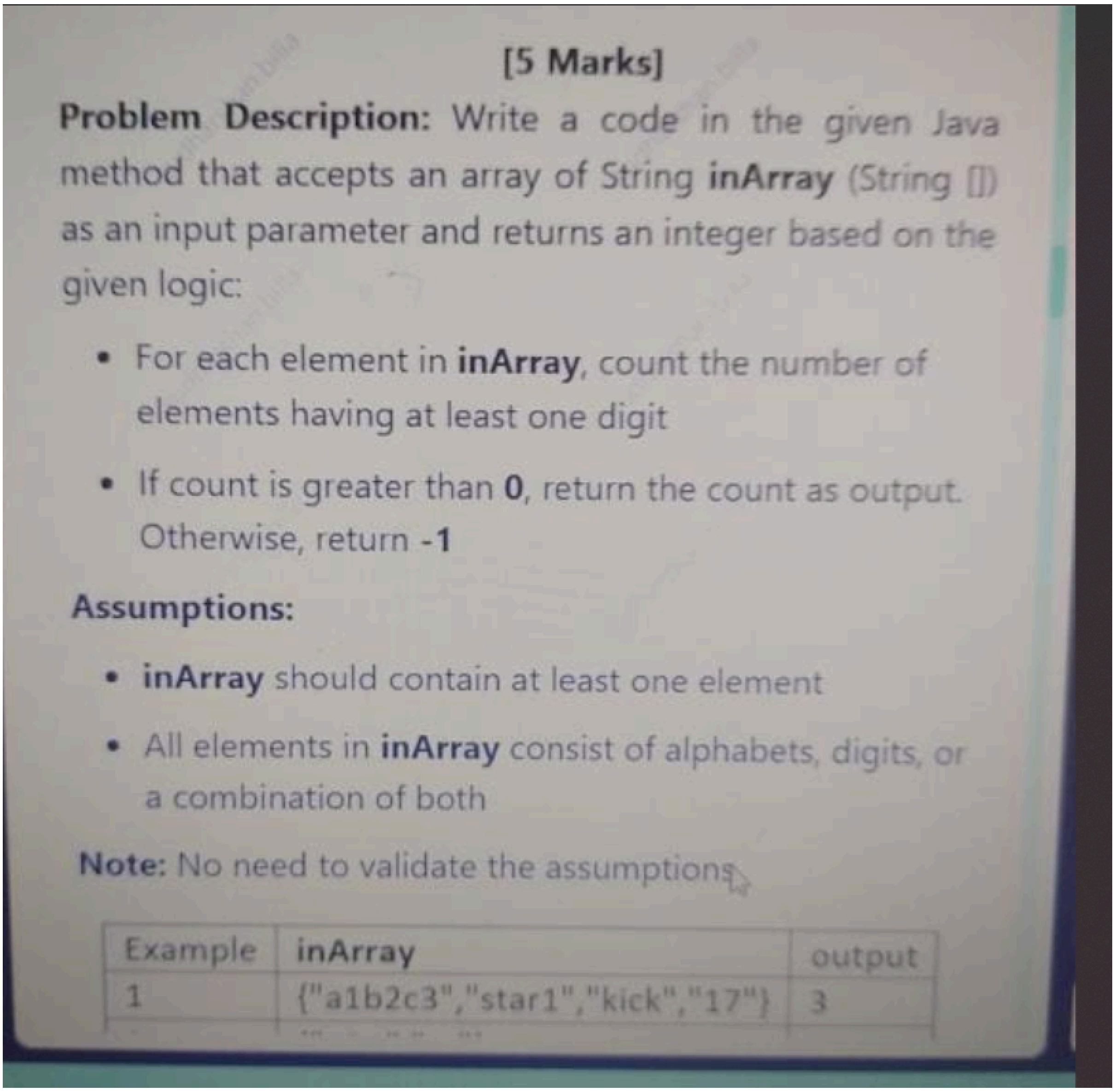
- **Input:** `"BINARY"`
- **Length:** 6 → even ✅
- **lowerString:** `"binary"`
- **First half:** `"bin"`
  → Vowels = `'i'` → only 1 vowel ❌
- **Output:** `"bi"` (first 2 characters of `lowerString`)

✅ Test Case 5

- **Input:** `"Abcdefgh"`
- **Length:** 8 → even ✅
- **lowerString:** `"abcdefgh"`
- **First half:** `"abcd"` → Vowels = `'a'` → only 1 vowel ❌
- **Output:** `"ab"`

✅ Test Case 6

- **Input:** `"Education"`
- **Length:** 9 → odd ❌
- **Output:** `"X"`



[5 Marks]

Problem Description: Write a code in the given Java method that accepts an array of String **inArray** (String []) as an input parameter and returns an integer based on the given logic:

- For each element in **inArray**, count the number of elements having at least one digit
- If count is greater than **0**, return the count as output. Otherwise, return **-1**

Assumptions:

- **inArray** should contain at least one element
- All elements in **inArray** consist of alphabets, digits, or a combination of both

Note: No need to validate the assumptions

Example	inArray	output
1	{"a1b2c3","star1","kick","17"}	3

📥 Example Test Case from Image:

🔹 **Input:**

java
CopyEdit

```
String[] inArray = {"a1b2c3", "star1", "kick", "17"};
```

🔍 **Processing:**

- "a1b2c3" → ✅ has digits
- "star1" → ✅ has digit
- "kick" → ❌

- "17" → ✅

✅ **Output:** **3**

🔁 More Test Cases:

Test Case 1:

Input: `{"apple", "mango", "banana"}`

- None have digits → `Output: -1`

Test Case 2:

Input: `{"code123", "java9", "dev"}`

- "code123" ✅
- "java9" ✅
- "dev" ❌
  → `Output: 2`

Test Case 3:

Input: `{"1", "2", "3", "4"}`
→ All have digits
→ `Output: 4`

1️⃣Problem A: String Array – Count Strings with Vowels & Digits

Description:
Given an array of strings, write a method that returns:

- The count of strings that contain **both at least one vowel** (`a, e, i, o, u`, case-insensitive) **and at least one digit**.
- If no string meets these criteria, return `-1`.

🧪 Test Cases:

Input	Explanation	Output
`{"a1", "hello", "3e", "B2b", "sky"}`	`"a1"` (has vowel & digit); `"3e"` (vowel `e` + digit `3`)	`2`
`{"abc", "def", "ghi"}`	No digits	`-1`
`{"X9u", "Z1A", "no5"}`	`"X9u"`, `"Z1A"`, `"no5"` all qualify	`3`
`{"Ae", "12", "b6"}`	`"b6"` has digit but no vowel; `"Ae"` has vowels, no digit	`-1`

🧠 Problem: Count Prime Numbers in an Integer Array

Description:
Given an integer array `arr[]`, write a method to count how many elements are **prime numbers**.
If the count is greater than 0, return the count; otherwise, return `-1`.

🔁 Sample Test Cases

Input	Explanation	Output
`[2, 3, 4, 5, 7, 8]`	Primes are `{2, 3, 5, 7}` ⇒ count = 4	`4`
`[1, 4, 6, 8, 9]`	No primes	`-1`
`[11, 13, 17, 19]`	All are primes	`4`
`[15, 21, 22, 23, 24]`	Only `23` is prime ⇒ count = 1	`1`
`[0, 1, 2]`	Only `2` is prime	`1`

✅ Question 1: String Array Based (Similar to Image 1)

Problem Description:
Write a method that accepts a `String[] inputArray` as an input. For each element, check whether it **starts with an uppercase letter** and contains **at least one digit**.

- Count the number of such strings.

- If count > 0, return the count.
- Otherwise, return `-1`.

Assumptions:

- `inputArray` contains at least one element.
- All strings may contain letters, digits, or both.
- No need to validate the assumptions.

Example Test Cases:

`inputArray`	**Output**	**Explanation**
`{"A1test", "B23", "hello", "Test2", "c1"}`	`3`	Only first three start with uppercase and have digit
`{"abc", "def", "ghi"}`	`-1`	No digit or no uppercase start
`{"Z9", "X", "Y3test"}`	`2`	`Z9` and `Y3test` match

✅ Question 2: Integer Array Based (Similar to Image 2)

Problem Description:
Write a Java method that accepts an `int[] nums` and returns:

- The **count of numbers that are divisible by both 2 and 3**.
- If none found, return `-1`.

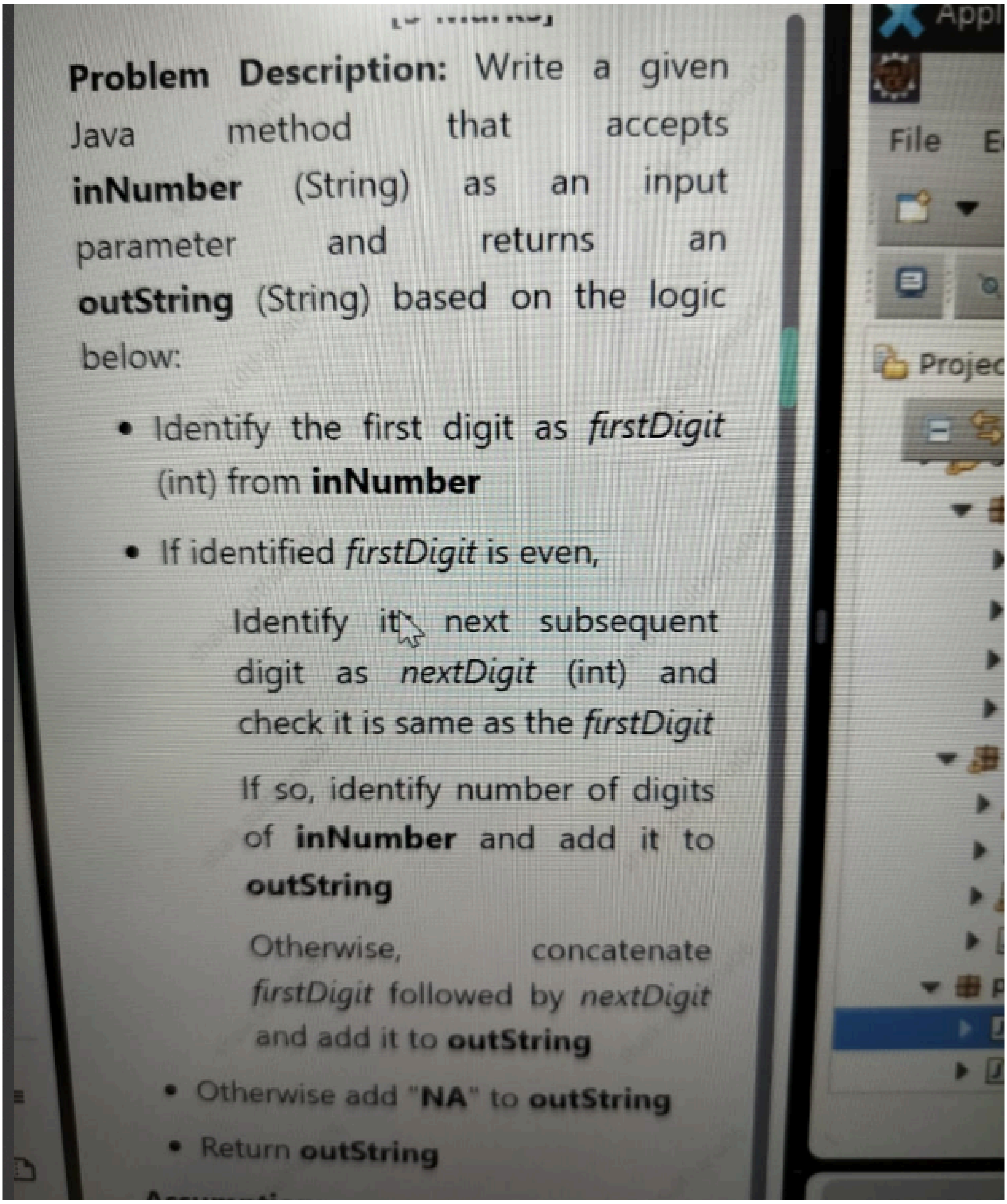
Assumptions:

- The array is not empty.

Example Test Cases:

`nums`	**Output**	**Explanation**
`{6, 12, 18, 20, 25}`	`3`	6, 12, 18 are divisible by 2 & 3
`{5, 7, 11, 13}`	`-1`	No such numbers

`{30, 60, 90}`	`3`	All divisible by 2 & 3



Problem Description: Write a given Java method that accepts **inNumber** (String) as an input parameter and returns an **outString** (String) based on the logic below:

- Identify the first digit as *firstDigit* (int) from **inNumber**
- If identified *firstDigit* is even,

  Identify it next subsequent digit as *nextDigit* (int) and check it is same as the *firstDigit*

  If so, identify number of digits of **inNumber** and add it to **outString**

  Otherwise, concatenate *firstDigit* followed by *nextDigit* and add it to **outString**
- Otherwise add "**NA**" to **outString**
- Return **outString**

✅ Problem Summary (as per image)

You are given a string `inNumber` (which consists of digits). Your task is:

1. Extract the **first digit** → `firstDigit`

2. If `firstDigit` is **even**:
    - Extract the **next digit** → `nextDigit`
    - If `nextDigit == firstDigit` →
      Add the **length** of the string (number of digits in `inNumber`) to `outString`
    - Else → Add `firstDigit` + `nextDigit` to `outString` (as String)
3. If `firstDigit` is **odd**, return `"NA"`

🧪 Test Case 1

Input: `"44678"`

- firstDigit = 4 (even)
- nextDigit = 4 → (same as firstDigit)
- length = 5 → ✅
  Output: `"5"`

🧪 Test Case 2

Input: `"42890"`

- firstDigit = 4 (even)
- nextDigit = 2 → not same
  Output: `"42"`

🧪 Test Case 3

Input: `"75921"`

- firstDigit = 7 (odd)
  Output: `"NA"`

🧪 Test Case 4

Input: `"28888"`

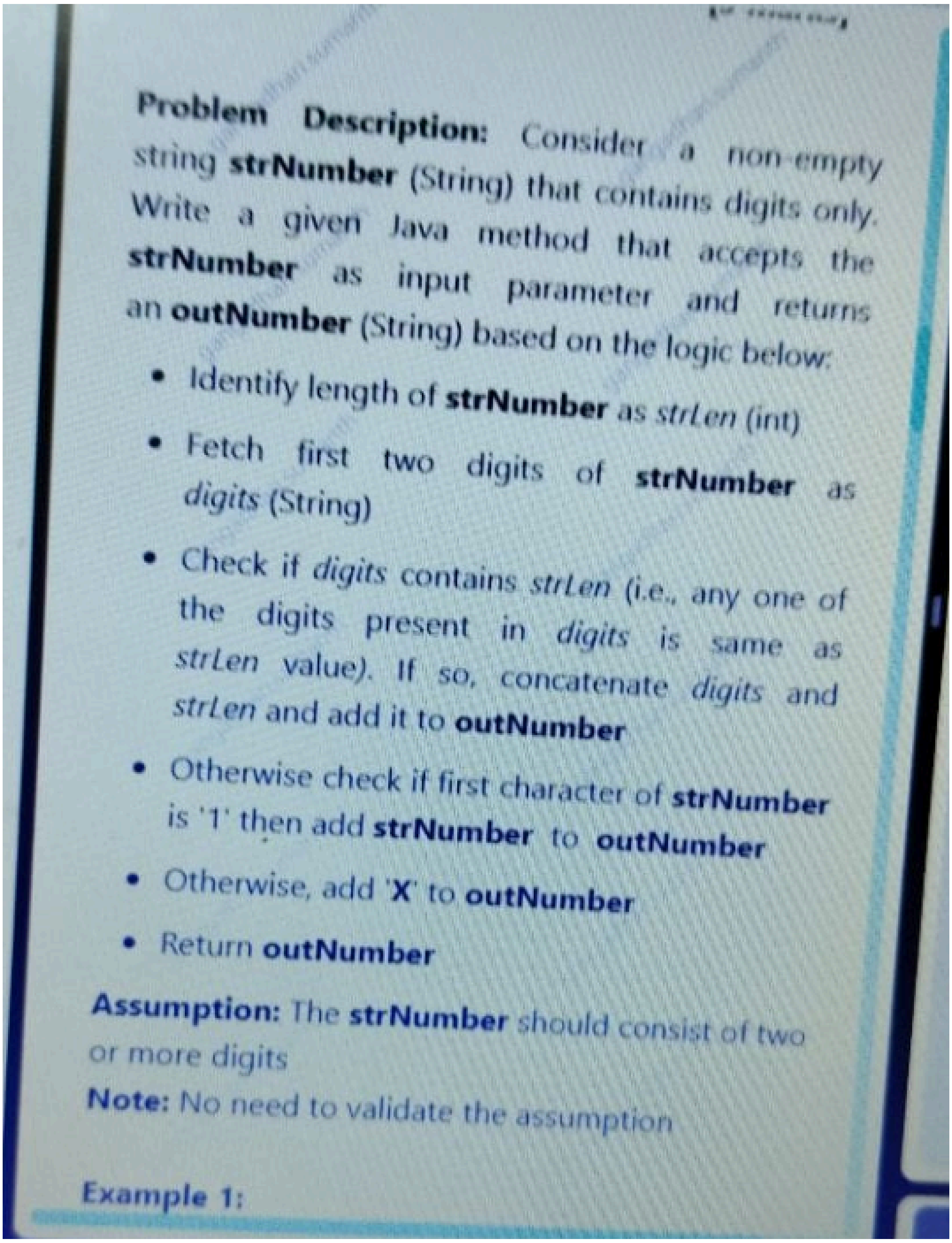
- firstDigit = 2 (even)
- nextDigit = 8 → not same
  Output: `"28"`

🧪 Test Case 5

Input: `"22222"`

- firstDigit = 2 (even)
- nextDigit = 2 → same
- length = 5
  Output: `"5"`

```
public class DigitStringProcessor {

    public static String processNumber(String inNumber) {
        int length = inNumber.length();

        // Get last digit
        int lastDigit = Character.getNumericValue(inNumber.charAt(length - 1));

        // Case 1: Last digit is odd
        if (lastDigit % 2 != 0) {
            // Ensure there's a second last digit
            if (length < 2) return "OddEvenMismatch";

            int secondLastDigit = Character.getNumericValue(inNumber.charAt(length - 2));

            if (secondLastDigit % 2 == 0) {
                // Sum of all digits
                int sum = 0;
                for (int i = 0; i < length; i++) {
                    sum += Character.getNumericValue(inNumber.charAt(i));
                }
                return String.valueOf(sum);
            } else {
                return "OddEvenMismatch";
            }
        }

        // Case 2: Last digit is even
        else {
            return "EndsEven";
        }
    }

    public static void main(String[] args) {
        System.out.println(processNumber("24681"));    // Output: 21
        System.out.println(processNumber("78543"));    // Output: 27
        System.out.println(processNumber("23475"));    // Output: OddEvenMismatch
        System.out.println(processNumber("8902"));     // Output: EndsEven
        System.out.println(processNumber("97"));       // Output: OddEvenMismatch
    }
}
```



Problem Description: Consider a non-empty string **strNumber** (String) that contains digits only. Write a given Java method that accepts the **strNumber** as input parameter and returns an **outNumber** (String) based on the logic below:

- Identify length of **strNumber** as *strLen* (int)
- Fetch first two digits of **strNumber** as *digits* (String)
- Check if *digits* contains *strLen* (i.e., any one of the digits present in *digits* is same as *strLen* value). If so, concatenate *digits* and *strLen* and add it to **outNumber**
- Otherwise check if first character of **strNumber** is '1' then add **strNumber** to **outNumber**
- Otherwise, add '**X**' to **outNumber**
- Return **outNumber**

Assumption: The **strNumber** should consist of two or more digits

Note: No need to validate the assumption

Example 1:

✅ Problem Summary:

Given:
A string `strNumber` consisting of digits only (at least two digits).

Return:
A string `outNumber` based on the following rules:

✅ Logic:

1. Let `strLen` be the length of `strNumber`.
2. Let `digits` = first 2 characters of `strNumber`.
3. **If** `digits` contain `strLen` as a digit → `outNumber = digits + strLen`
4. **Else if** first digit of `strNumber` is `'1'` → `outNumber = strNumber`
5. **Else** → `outNumber = "X"`

✅ Test Cases:

Test Case #	`strNumber`	Length (`strLen`)	First 2 Digits (`digits`)	Condition Met	`outNumber`
1	`"541289"`	6	"54"	❌ Not in "54" ❌ Not starting with 1	`"X"`
2	`"169023"`	6	"16"	✅ "6" in "16"	`"166"`
3	`"10876"`	5	"10"	❌ ✅ Starts with 1	`"10876"`
4	`"23789"`	5	"23"	❌ ❌	`"X"`
5	`"712"`	3	"71"	❌ ❌	`"X"`
6	`"1245"`	4	"12"	✅ "4" in "12" ❌	`"X"`
7	`"4312"`	4	"43"	✅ "4" in "43"	`"434"`

🧪 Sample Inputs and Expected Outputs:

java
CopyEdit

```
processStrNumber("541289") → "X"
processStrNumber("169023") → "166"
processStrNumber("10876")  → "10876"
processStrNumber("23789")  → "X"
processStrNumber("712")    → "X"
processStrNumber("4312")   → "434"
```