Question: 1
Write a Java program that takes a string input from the user and counts how many vowels (a, e, i, o, u) are present in it.

Example:

- **Input:** "Hello World"
- **Output:** Number of vowels: 3

Methods to be used:

1. String str = sc.nextLine();

Explanation:

nextLine() is a method from Scanner class

nextLine() reads the **whole line** typed by the user (including spaces) **while hit 'Enter'.**

2. toLowerCase();

Explanation:

toLowerCase() is a String method that converts all uppercase letters into lowercase. No matter the char is 'A' or 'a' it always counts as vowel, so convert the string to reduce conditions like {if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') }

Example:
"Hello World".toLowerCase() → "hello world"

3. charAt(i);

Explanation:

charAt(i) gives the character at index i of the string.

In short:

- nextLine() → takes full user input string.
- toLowerCase() → makes comparison easy.
- charAt(i) → picks each character for checking.

Question 2:

Question:

Write a Java program that takes **two strings** from the user and performs the following tasks:

1. Print the length of both strings.
2. Check if both strings are equal (case-sensitive).
3. Check if both strings are equal (ignoring case).
4. Compare both strings lexicographically (case-sensitive).
5. Compare both strings lexicographically (ignoring case).

Example Run:

```
Enter first string: Hello
Enter second string: hello
Length of first string: 5
Length of second string: 5
Are they equal (case-sensitive)? false
Are they equal (ignore case)? true
compareTo result: -32
compareToIgnoreCase result: 0
```

Methods used:

1. length()

- **Definition:** Returns the total number of characters in the string (including spaces, digits, and special characters).
- **Syntax:** str.length()
- **Return type:** int

Example:

String s = "Hello";

System.out.println(s.length()); // Output: 5

"Hello" has 5 characters.

2. equals(Object obj)

- **Definition:** Compares the **content** of two strings, **case-sensitive** (capital vs small matters).

- **Syntax:** str1.equals(str2)
- **Return type:** boolean (true or false)

Example:

```
String a = "Hello";

String b = "hello";

System.out.println(a.equals(b));  // Output: false
```

Because "H" ≠ "h" (case-sensitive).

3. equalsIgnoreCase(String another)

- **Definition:** Compares the **content** of two strings but **ignores case differences**.
- **Syntax:** str1.equalsIgnoreCase(str2)
- **Return type:** boolean

Example:

```
String a = "Hello";

String b = "hello";

System.out.println(a.equalsIgnoreCase(b));  // Output: true
```

Here, "Hello" and "hello" are considered equal because case is ignored.

4. compareTo(String another)

- **Definition:** Compares two strings **lexicographically** (dictionary order).
- It checks character by character using **Unicode values**.
- **Return values:**
  - 0 → both strings are equal
  - +ve → first string is greater
  - -ve → first string is smaller

👉 Example:

```
String a = "Hello";

String b = "World";

System.out.println(a.compareTo(b));  // Output: -15
```

Because 'H' (72) is smaller than 'W' (87) → difference = 72 - 87 = **-15**.

What compareTo actually does

compareTo compares **two strings in dictionary order** by looking at them **character by character** (left to right). It uses each character's **UTF-16 code unit value** (you can think of this like a numeric ID for the character).

It returns:

- 0 → strings are exactly equal
- **positive** → this string is greater (comes **after**)
- **negative** → this string is smaller (comes **before**)

Only the **sign** (negative / zero / positive) is meaningful. Don't rely on the **exact number**.

Step-by-step algorithm (mental model)

1. Let s1 be the first string and s2 be the second.
2. Compare characters at index i = 0, 1, 2, ...:
   - If s1[i] == s2[i], continue.
   - If s1[i] != s2[i], return s1[i] - s2[i] (their numeric difference).
3. If all compared chars are equal up to the end of the shorter string, return s1.length() - s2.length()
   (because a longer string with the same prefix is considered **greater**).

Dry runs (see how the number comes)

1. **Different at the first character**

- "Hello".compareTo("World")
  - Compare 'H'(72) with 'W'(87) → 72 - 87 = -15 → **-15** (so "Hello" comes before "World").

2. **Different later**

- "car".compareTo("cat")
  - 'c' == 'c', 'a' == 'a', now compare 'r'(114) vs 't'(116) → 114 - 116 = -2 → **-2** (so "car" comes before "cat").

3. **Prefix case**

- "app".compareTo("apple")
  - All first 3 chars match, but lengths differ → 3 - 5 = -2 → **-2** (shorter prefix comes before its longer extension).

4. **Upper vs lower case matters (case-sensitive)**

- “Zebra”.compareTo(“apple”)
  - 'Z'(90) vs 'a'(97) → 90 - 97 = -7 → **-7** (uppercase ‘Z’ comes before lowercase ‘a’ in Unicode order).
- “a”.compareTo(“A”) → 97 - 65 = 32 → **+32** (so "a" > "A").

Important details & edge cases

- **Case-sensitive:** compareTo does **not** ignore case. For case-insensitive order, use compareToIgnoreCase.
- **Null:** Calling s.compareTo(null) throws **NullPointerException**.
- **Speed:** Runs in **O(min(n, m))** where n and m are lengths of the two strings.
- **Unicode / UTF-16:** Java Strings are UTF-16. The comparison uses **code units**. For normal text (including emoji as valid surrogate pairs), this still matches Unicode code point order. If a string had **unpaired** surrogates (rare / invalid), ordering is purely by those code units.
- **Locale:** This is **binary Unicode order**, **not** human, language-aware collation. For proper dictionary sorting in a language (e.g., accents, “ch” in Spanish), use Collator from java.text

Quick mini-table (intuition)

s1 vs s2	First difference	Result (why)
"abc" vs "abc"	none	0 (equal)
"abc" vs "abd"	'c'(99) - 'd'(100) = -1	negative (before)
"abd" vs "abc"	'd'(100) - 'c'(99) = +1	positive (after)
"app" vs "apple"	length 3 - 5 = -2	negative (prefix)
"Z" vs "a"	90 - 97 = -7	negative (case matters)
"a" vs "A"	97 - 65 = +32	positive (case matters)

Question: 3

Write a Java program that takes a sentence from the user and performs the following searches:

1. **Check if the sentence contains the word "java".**
2. **Check if the sentence starts with "Hello".**
3. **Check if the sentence ends with "world".**
4. **Find the first index of the character 'a'.**
5. **Find the last index of the character 'a'.**

Example Run:

```
Input:
   Enter a sentence: Hello java world

Output:
  Does the sentence contain "java"? true
  Does the sentence start with "Hello"? true
  Does the sentence end with "world"? true
  First index of 'a': 7
  Last index of 'a': 8
```

Methods used:

- **contains(CharSequence s)**
- **startsWith(String prefix)**
- **endsWith(String suffix)**
- **indexOf(char/str)**
- **lastIndexOf(char/str)**

Explanation of Methods

1. contains(CharSequence s)

- Checks if the substring exists inside the string.
- Returns true if found, false if not.
  Example: "Hello java world".contains("java") → true.

2. startsWith(String prefix)

- Checks whether the string begins with the given prefix.
- Case-sensitive.
  Example: "Hello java world".startsWith("Hello") → true.

3. endsWith(String suffix)

- Checks whether the string ends with the given suffix.
- Case-sensitive.
  Example: "Hello java world".endsWith("world") → true.

4. indexOf(char/str)

- Returns the first index where a character or substring appears.
- If not found → returns -1.
  Example: "Hello java world".indexOf('a') → 7 (the first 'a').

5. lastIndexOf(char/str)

- Returns the last index where a character or substring appears.
- If not found → returns -1.
  Example: "Hello java world".lastIndexOf('a') → 8 (the last 'a').

Question: 4

Write a Java program that takes a string input from the user and performs the following tasks using substring() methods:

1. Print the substring from a given index till the end.
2. Print the substring between two given indexes.

Example Run:

```
Enter a string: SkillBridge
Enter starting index: 5
Substring from index 5: Bridge

Enter starting index: 0
Enter ending index: 5
Substring from index 0 to 5: Skill
```

Explanation of Methods

1. substring(int beginIndex)

- Returns part of the string **from beginIndex till end**.
- Index starts at 0.
  Example:
  "SkillBridge".substring(5) → "Bridge"

---

2. substring(int beginIndex, int endIndex)

- Returns part of the string **from beginIndex to (endIndex - 1)**.
- The character at endIndex is **excluded**.
  Example:
  "SkillBridge".substring(0, 5) → "Skill"

Question: 5

Write a Java program that takes a string input from the user and performs the following tasks:

1. Remove all leading and trailing spaces from the string using trim().
2. Replace all spaces in the string with underscores _ using replace().
3. Replace all occurrences of a character entered by the user with another character.

Example Run:

Enter a string: Hello World Java

After trim(): "Hello World Java"

After replacing spaces with _: "Hello_World_Java"

Enter character to replace: a

Enter new character: @

After replacing 'a' with '@': "Hello_World_J@v@"

Explanation of Methods

1. trim()

- Removes **leading and trailing spaces** from a string.
- Spaces in the **middle** of the string are **not removed**.
  Example:

  "   Hello World ".trim() → "Hello World"

2. replace(char oldChar, char newChar)

- Replaces **all occurrences** of oldChar with newChar.
- Works on the whole string.
  Example:

  "Hello World".replace(' ', '_') → "Hello_World"

  "Hello World".replace('o', '@') → "Hell@ W@rld"

Question: 6

Write a Java program that takes a **sentence** from the user and performs the following tasks:

1. Split the sentence into words using split() (split by spaces).
2. Print all words in **separate lines**.
3. Join the words back together using join() with a hyphen - between them.

Example Run:

```
Enter a sentence: Java is fun

Words after splitting:

Java

is

fun


Sentence after joining with hyphens: Java-is-fun
```

Explanation of Methods

1. split(String regex)

- Splits the string into an array of substrings **based on the given regex**.
- Commonly used: " " (space) to split words.

Example:

"Java is fun".split(" ") → ["Java", "is", "fun"]

2. join(CharSequence delimiter, CharSequence... elements)

- Joins array elements (or multiple strings) **using the specified delimiter**.
- Returns a single combined string.

Example:

String.join("-", "Java", "is", "fun") → "Java-is-fun"