**Solutions**

Question 1:

The code defines a Tokenization class with a main method.

Two Java code fragments (code1 and code2) are provided as strings.

The tokenize method uses a Scanner to tokenize each code fragment.

The resulting token sequences are printed for both code fragments.

Idea:

Tokenize each code fragment by breaking it into individual words or symbols.

Approach:

Use a Scanner to scan and collect tokens from the code.

Print the resulting tokens for both code fragments.

Question 2:

The code defines a VectorRepresentation class with a main method.

Two Java code fragments (code1 and code2) are provided as strings.

The tokenize method is used to tokenize each code fragment.

The createVector method creates a vector representation by counting token occurrences.

The resulting vector representations are printed for both code fragments.

Idea:

Represent each code fragment as a vector, where each dimension is a token type.

Approach:

Tokenize each code fragment using a Scanner.

Create a vector representation by counting the occurrences of each token in the code.

Print the resulting vectors for both code fragments.

Question 4:

Code Breakdown:

AST Representation:

The abstract syntax trees (ASTs) of Code 1 and Code 2 are essentially trees that represent the structure of the code. Here, we've expressed them as simple strings where different operations or actions in the code are separated by spaces.

Creating Vectors:

We have a method called createVector that takes an AST string (like the ones we created for Code 1 and Code 2) and turns it into a vector.

This vector is like a shopping list where each item is a type of operation or action from the code.

Counting Occurrences:

Inside the createVector method, we're essentially counting how many times each operation or action appears in the code. This count is stored in a data structure called a HashMap.

Displaying Vectors:

In the main method, we use the createVector method to get vectors for both Code 1 and Code 2.

We then print out these vectors so that we can visually compare them.

Question 5:

Code Fragments:

Two Java code fragments (code1 and code2) are given as strings.

Text Transformation:

The transformText method takes a code fragment and removes non-alphabetic characters, keeping only letters and spaces.

Tokenization:

The tokenize method uses a Scanner to tokenize the transformed text, creating a list of words (tokens).

Transformation and Tokenization:

The main method applies the text transformation and tokenization to both code fragments.

It prints the transformed text and tokens for each code fragment.

The primary idea is to simplify the code by focusing on the textual content, removing symbols and special characters, and then tokenize the simplified text for comparison. The approach involves regular expressions for text transformation and a Scanner for tokenization.

Analysis:

The transformed text for both Code 1 and Code 2 reveals a high degree of similarity. Both code fragments follow a similar structure and perform analogous operations, such as checking the ending of the input, extracting a substring, and evaluating letter types.

Therefore, based on the textual content after transformation, we can conclude that the two code fragments are still highly similar.