1. **Input.py**

import java.nio.ByteBuffer;

import java.nio.ByteOrder;

import java.util.Scanner;

public class Assign6 {

    // Step 1: Define shift amounts for each round

    private static final int[] rotateAmounts = {

            7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,

            5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20,

            4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,

            6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21

    };

    // Step 2: Define Constants (K values derived from the sine function)

    private static final int[] constants = new int[64];

    // Step 3: Initialize MD Buffer (A, B, C, D)

    private static final int[] initValues = {

            0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476

    };

    // Step 2: Compute the K values

    static {

        for (int i = 0; i < 64; i++) {

            constants[i] = (int) ((long) (Math.abs(Math.sin(i + 1)) \* (1L << 32)) & 0xFFFFFFFFL);

        }

    }

    // Step 4: Function to perform left rotation

    private static int leftRotate(int x, int amount) {

        return (x << amount) | (x >>> (32 - amount));

    }

    // Step 5: Main MD5 function

    public static byte[] md5(byte[] message) {

        // Step 5.1: Padding the message to make its length congruent to 448 mod 512

        int originalLength = message.length;

        long originalLengthBits = (long) originalLength \* 8;

        int newLength = ((originalLength + 8) / 64 + 1) \* 64;

        byte[] paddedMessage = new byte[newLength];

        System.arraycopy(message, 0, paddedMessage, 0, originalLength);

        // Append the '1' bit (0x80 in hexadecimal)

        paddedMessage[originalLength] = (byte) 0x80;

        // Append the length of the original message (in bits) at the end

        for (int i = 0; i < 8; i++) {

            paddedMessage[newLength - 8 + i] = (byte) (originalLengthBits >>> (8 \* i));

        }

        // Step 5.2: Initialize hash values

        int[] hashPieces = initValues.clone();

        // Step 5.3: Process each 512-bit block (64 bytes per block)

        for (int chunkOffset = 0; chunkOffset < newLength; chunkOffset += 64) {

            int[] words = new int[16];

            for (int i = 0; i < 16; i++) {

                words[i] = ByteBuffer.wrap(paddedMessage, chunkOffset + i \* 4, 4)

                        .order(ByteOrder.LITTLE\_ENDIAN)

                        .getInt();

            }

            // Step 5.4: Initialize variables for this chunk

            int a = hashPieces[0], b = hashPieces[1], c = hashPieces[2], d = hashPieces[3];

            // Step 5.5: Main loop (64 operations)

            for (int i = 0; i < 64; i++) {

                int f, g;

                if (i < 16) {

                    f = (b & c) | (~b & d);

                    g = i;

                } else if (i < 32) {

                    f = (d & b) | (~d & c);

                    g = (5 \* i + 1) % 16;

                } else if (i < 48) {

                    f = b ^ c ^ d;

                    g = (3 \* i + 5) % 16;

                } else {

                    f = c ^ (b | ~d);

                    g = (7 \* i) % 16;

                }

                // Step 5.6: Perform transformation

                long temp = ((long) a + f + constants[i] + words[g]) & 0xFFFFFFFFL;

                a = d;

                d = c;

                c = b;

                b = (int) ((b + leftRotate((int) temp, rotateAmounts[i])) & 0xFFFFFFFFL);

            }

            // Step 5.7: Add this chunk's hash to result

            hashPieces[0] = (hashPieces[0] + a) & 0xFFFFFFFF;

            hashPieces[1] = (hashPieces[1] + b) & 0xFFFFFFFF;

            hashPieces[2] = (hashPieces[2] + c) & 0xFFFFFFFF;

            hashPieces[3] = (hashPieces[3] + d) & 0xFFFFFFFF;

        }

        // Step 6: Produce final hash value (128-bit digest)

        ByteBuffer buffer = ByteBuffer.allocate(16).order(ByteOrder.LITTLE\_ENDIAN);

        for (int value : hashPieces) {

            buffer.putInt(value);

        }

        return buffer.array();

    }

    // Step 7: Convert hash bytes to hexadecimal string

    public static String md5ToHex(byte[] digest) {

        StringBuilder hexString = new StringBuilder();

        for (byte b : digest) {

            hexString.append(String.format("%02x", b & 0xFF));

        }

        return hexString.toString();

    }

    // Step 8: Main method to accept user input and compute MD5 hash

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a string to hash using MD5: ");

        String input = scanner.nextLine();

        scanner.close();

        byte[] hashedBytes = md5(input.getBytes());

        String hashedValue = md5ToHex(hashedBytes);

        System.out.println("MD5 hash: " + hashedValue);

    }

}

1. **Output**

