Computer Science Lab Exercises

This document contains all the CS lab exercises with detailed explanations and implementations.

Exercises

Exercise 1.1: Caesar Cipher

Aim: To write a Java program to perform encryption and decryption using Caesar cipher algorithm.

Algorithm:

1. Read the plaintext from user
2. Read the shift key (default: 3)
3. For encryption:
    - Convert text to lowercase
    - For each character, find its position in alphabet
    - Apply formula: `newPos = (pos + key) % 26`
    - Replace with character at new position
4. For decryption:
    - Apply formula: `newPos = (pos - key + 26) % 26`
    - Handle negative values by adding 26
5. Display original, encrypted, and decrypted text

Concept:

- Caesar cipher shifts each letter by a fixed number of positions in the alphabet
- Encryption: `E(P,k) = (P + k) mod 26`
- Decryption: `D(C,k) = (C - k) mod 26`

Java Code:

```
import java.util.Scanner;

/**
 * Caesar Cipher implementation for encryption and decryption
 * Shifts letters by a fixed key value in the alphabet
 */
class CaesarCipher {
    private static final String ALPHABET = "abcdefghijklmnopqrstuvwxyz";

    // Encrypt text using Caesar cipher
    public static String encrypt(String text, int key) {
        text = text.toLowerCase();
        StringBuilder result = new StringBuilder();

        for (char c : text.toCharArray()) {
            int pos = ALPHABET.indexOf(c);
            if (pos != -1) { // Only process alphabetic characters
                int newPos = (pos + key) % 26;
```

```
                result.append(ALPHABET.charAt(newPos));
            } else {
                result.append(c); // Keep non-alphabetic characters unchanged
            }
        }
        return result.toString();
    }

    // Decrypt text using Caesar cipher
    public static String decrypt(String text, int key) {
        text = text.toLowerCase();
        StringBuilder result = new StringBuilder();

        for (char c : text.toCharArray()) {
            int pos = ALPHABET.indexOf(c);
            if (pos != -1) { // Only process alphabetic characters
                int newPos = (pos - key + 26) % 26; // +26 to handle negative
values
                result.append(ALPHABET.charAt(newPos));
            } else {
                result.append(c); // Keep non-alphabetic characters unchanged
            }
        }
        return result.toString();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter text to encrypt: ");
        String text = sc.nextLine();

        System.out.print("Enter shift key (default 3): ");
        int key = sc.hasNextInt() ? sc.nextInt() : 3;

        String encrypted = encrypt(text, key);
        String decrypted = decrypt(encrypted, key);

        System.out.println("\nResults:");
        System.out.println("Original:  " + text);
        System.out.println("Encrypted: " + encrypted);
        System.out.println("Decrypted: " + decrypted);

        sc.close();
    }
}
```

Sample Output:

```
Enter text to encrypt: computer
Enter shift key (default 3): 3

Results:
```

```
Original:  computer
Encrypted: frpsxwhu
Decrypted: computer
```

Result: Thus a Java program to perform encryption and decryption using Caesar cipher algorithm was executed successfully.

Exercise 1.2: Playfair Cipher

Aim: To write a Java program to perform encryption and decryption using Playfair cipher technique.

Algorithm:

1. Read the plaintext and keyword from user
2. Create 5x5 matrix using keyword:
    - Fill keyword letters (remove duplicates) from left to right, top to bottom
    - Fill remaining positions with alphabetic order (I and J count as one letter)
3. Process plaintext in pairs:
    - Add filler 'x' if odd length or repeating letters in same pair
4. Apply encryption rules for each pair:
    - Same row: Replace with letter to the right (wrap around)
    - Same column: Replace with letter below (wrap around)
    - Rectangle: Replace with letter in same row but other's column
5. For decryption, reverse the process

Concept:

- Playfair cipher encrypts pairs of letters using a 5x5 key matrix
- Uses position-based substitution with three different rules
- More secure than simple substitution ciphers

Java Code:

```
import java.util.*;

/**
 * Playfair Cipher implementation
 * Encrypts/decrypts text using 5x5 key matrix and pair-based rules
 */
class PlayfairCipher {
    private static char[][] matrix = new char[5][5];

    // Create 5x5 matrix from keyword
    public static void createMatrix(String key) {
        String alphabet = "abcdefghiklmnopqrstuvwxyz"; // j is omitted
        boolean[] used = new boolean[26];
        int row = 0, col = 0;

        // Add keyword letters first
        for (char c : key.toLowerCase().toCharArray()) {
```

```
            if (c == 'j') c = 'i'; // treat j as i
            if (!used[c - 'a']) {
                matrix[row][col] = c;
                used[c - 'a'] = true;
                if (++col == 5) { col = 0; row++; }
            }
        }

        // Fill remaining with alphabet
        for (char c : alphabet.toCharArray()) {
            if (!used[c - 'a']) {
                matrix[row][col] = c;
                if (++col == 5) { col = 0; row++; }
            }
        }
    }

    // Find position of character in matrix
    public static int[] findPosition(char c) {
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                if (matrix[i][j] == c) return new int[]{i, j};
            }
        }
        return new int[]{-1, -1};
    }

    // Encrypt text using Playfair rules
    public static String encrypt(String text) {
        text = text.toLowerCase().replace("j", "i");
        StringBuilder processed = new StringBuilder();

        // Process pairs
        for (int i = 0; i < text.length(); i += 2) {
            char first = text.charAt(i);
            char second = (i + 1 < text.length()) ? text.charAt(i + 1) : 'x';

            // Add filler if same letters
            if (first == second) {
                processed.append(first).append('x');
                i--; // reprocess second char
            } else {
                processed.append(first).append(second);
            }
        }

        // Add padding if odd length
        if (processed.length() % 2 == 1) processed.append('x');

        StringBuilder result = new StringBuilder();
        for (int i = 0; i < processed.length(); i += 2) {
            char c1 = processed.charAt(i);
            char c2 = processed.charAt(i + 1);

            int[] pos1 = findPosition(c1);
```

```
            int[] pos2 = findPosition(c2);

            if (pos1[0] == pos2[0]) { // Same row
                result.append(matrix[pos1[0]][(pos1[1] + 1) % 5]);
                result.append(matrix[pos2[0]][(pos2[1] + 1) % 5]);
            } else if (pos1[1] == pos2[1]) { // Same column
                result.append(matrix[(pos1[0] + 1) % 5][pos1[1]]);
                result.append(matrix[(pos2[0] + 1) % 5][pos2[1]]);
            } else { // Rectangle
                result.append(matrix[pos1[0]][pos2[1]]);
                result.append(matrix[pos2[0]][pos1[1]]);
            }
        }
        return result.toString();
    }

    // Decrypt text using Playfair rules
    public static String decrypt(String text) {
        StringBuilder result = new StringBuilder();

        for (int i = 0; i < text.length(); i += 2) {
            char c1 = text.charAt(i);
            char c2 = text.charAt(i + 1);

            int[] pos1 = findPosition(c1);
            int[] pos2 = findPosition(c2);

            if (pos1[0] == pos2[0]) { // Same row
                result.append(matrix[pos1[0]][(pos1[1] + 4) % 5]);
                result.append(matrix[pos2[0]][(pos2[1] + 4) % 5]);
            } else if (pos1[1] == pos2[1]) { // Same column
                result.append(matrix[(pos1[0] + 4) % 5][pos1[1]]);
                result.append(matrix[(pos2[0] + 4) % 5][pos2[1]]);
            } else { // Rectangle
                result.append(matrix[pos1[0]][pos2[1]]);
                result.append(matrix[pos2[0]][pos1[1]]);
            }
        }
        return result.toString();
    }

    // Display the 5x5 matrix
    public static void displayMatrix() {
        System.out.println("\nThe matrix:");
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                System.out.print(matrix[i][j] + "\t");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
```

```
        System.out.print("Enter the message: ");
        String message = sc.nextLine();

        System.out.print("Enter the key: ");
        String key = sc.nextLine();

        createMatrix(key);
        displayMatrix();

        String encrypted = encrypt(message);
        String decrypted = decrypt(encrypted);

        System.out.println("\nPlayfair Cipher Text: " + encrypted);
        System.out.println("Playfair Plain Text: " + decrypted);

        sc.close();
    }
}
```

Sample Output:

```
Enter the message: cryptography
Enter the key: security

The matrix:
s e c u r
i t y a b
d f g h k
l m n o p
q v w x z

Playfair Cipher Text: usbnamkcboga
Playfair Plain Text: cryptography
```

Result: Thus a Java program to perform encryption and decryption using Playfair cipher algorithm was executed successfully.

Exercise 1.3: Hill Cipher

Aim: To write a Java program to perform encryption using Hill cipher algorithm.

Algorithm:

1. Read the plaintext from user
2. Define the key matrix (3x3 for processing 3 letters at a time)
3. Convert plaintext to numerical values (a=0, b=1, ..., z=25)
4. Group plaintext into blocks of 3 characters
5. For each block, apply matrix multiplication:
    - C = PK mod 26
    - Where P is plaintext vector, K is key matrix, C is ciphertext vector

6. Convert numerical results back to characters
7. Display encrypted text

Concept:

- Hill cipher uses linear algebra for encryption
- Takes m successive plaintext letters and substitutes them with m ciphertext letters
- Uses matrix multiplication: `C = PK mod 26`
- For 3x3 matrix: `c[i] = (k[i][0]*p[0] + k[i][1]*p[1] + k[i][2]*p[2]) mod 26`

Java Code:

```
import java.util.Scanner;

/**
 * Hill Cipher implementation
 * Encrypts text using matrix multiplication with a key matrix
 */
class HillCipher {
    // 3x3 key matrix
    private static final int[][] KEY_MATRIX = {
        {17, 17, 5},
        {21, 18, 21},
        {2, 2, 19}
    };

    // Convert text to numerical array
    public static int[] textToNumbers(String text) {
        int[] numbers = new int[text.length()];
        for (int i = 0; i < text.length(); i++) {
            numbers[i] = text.charAt(i) - 'a';
        }
        return numbers;
    }

    // Convert numerical array to text
    public static String numbersToText(int[] numbers, int length) {
        StringBuilder result = new StringBuilder();
        for (int i = 0; i < length; i++) {
            result.append((char) ((numbers[i] % 26) + 'a'));
        }
        return result.toString();
    }

    // Encrypt text using Hill cipher
    public static String encrypt(String plaintext) {
        plaintext = plaintext.toLowerCase().replace(" ", "");

        // Pad text to multiple of 3
        while (plaintext.length() % 3 != 0) {
            plaintext += "x";
        }
```

```
        int[] plainArray = textToNumbers(plaintext);
        int[] cipherArray = new int[plaintext.length()];

        // Process in blocks of 3
        for (int block = 0; block < plaintext.length() / 3; block++) {
            int startIndex = block * 3;

            // Matrix multiplication for each position in block
            for (int i = 0; i < 3; i++) {
                cipherArray[startIndex + i] = 0;
                for (int j = 0; j < 3; j++) {
                    cipherArray[startIndex + i] += KEY_MATRIX[i][j] *
plainArray[startIndex + j];
                }
            }
        }

        return numbersToText(cipherArray, plaintext.length());
    }

    // Display key matrix
    public static void displayKeyMatrix() {
        System.out.println("Key Matrix:");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(KEY_MATRIX[i][j] + "\t");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter plain text: ");
        String plaintext = sc.nextLine();

        displayKeyMatrix();

        String encrypted = encrypt(plaintext);

        System.out.println("\nOriginal Text: " + plaintext);
        System.out.println("Encrypted Text: " + encrypted);

        sc.close();
    }
}
```

Sample Output:

```
Enter plain text: paymoremoney
Key Matrix:
17  17  5
```

```
21  18  21
2   2   19

Original Text: paymoremoney
Encrypted Text: lnshdlewmtrw
```

Result: Thus a Java program to perform encryption using Hill cipher algorithm was executed successfully.

Exercise 1.4: Vigenère Cipher

Aim: To write a Java program to perform encryption and decryption using Vigenère cipher technique.

Algorithm:

1. Read the plaintext and keyword from user
2. Extend the keyword to match the length of plaintext by repeating it
3. Create Vigenère table (26x26 matrix) where each row is alphabet shifted by row index
4. For encryption:
    - For each character, find row using key character and column using plaintext character
    - Formula: `C[i] = (P[i] + K[i % keyLength]) mod 26`
5. For decryption:
    - Reverse the process to find original plaintext
    - Formula: `P[i] = (C[i] - K[i % keyLength] + 26) mod 26`
6. Display results

Concept:

- Vigenère cipher uses a repeating keyword for encryption
- Each letter is shifted by a different amount based on the key
- More secure than Caesar cipher due to varying shifts
- Uses modular arithmetic: `Encryption: C = (P + K) mod 26`, `Decryption: P = (C - K + 26) mod 26`

Java Code:

```java
import java.util.Scanner;

/**
 * Vigenère Cipher implementation
 * Encrypts/decrypts text using a repeating keyword
 */
class VigenereCipher {
    private static final String ALPHABET = "abcdefghijklmnopqrstuvwxyz";

    // Extend key to match text length
    public static String extendKey(String key, int textLength) {
        StringBuilder extendedKey = new StringBuilder(key);
        while (extendedKey.length() < textLength) {
            extendedKey.append(key);
        }
```

```
        return extendedKey.substring(0, textLength);
    }

    // Encrypt text using Vigenère cipher
    public static String encrypt(String plaintext, String key) {
        plaintext = plaintext.toLowerCase();
        key = key.toLowerCase();
        String extendedKey = extendKey(key, plaintext.length());

        StringBuilder ciphertext = new StringBuilder();

        for (int i = 0; i < plaintext.length(); i++) {
            if (Character.isLetter(plaintext.charAt(i))) {
                int plainIndex = plaintext.charAt(i) - 'a';
                int keyIndex = extendedKey.charAt(i) - 'a';
                int cipherIndex = (plainIndex + keyIndex) % 26;
                ciphertext.append((char) (cipherIndex + 'a'));
            } else {
                ciphertext.append(plaintext.charAt(i)); // Keep non-letters
unchanged
            }
        }

        return ciphertext.toString();
    }

    // Decrypt text using Vigenère cipher
    public static String decrypt(String ciphertext, String key) {
        ciphertext = ciphertext.toLowerCase();
        key = key.toLowerCase();
        String extendedKey = extendKey(key, ciphertext.length());

        StringBuilder plaintext = new StringBuilder();

        for (int i = 0; i < ciphertext.length(); i++) {
            if (Character.isLetter(ciphertext.charAt(i))) {
                int cipherIndex = ciphertext.charAt(i) - 'a';
                int keyIndex = extendedKey.charAt(i) - 'a';
                int plainIndex = (cipherIndex - keyIndex + 26) % 26;
                plaintext.append((char) (plainIndex + 'a'));
            } else {
                plaintext.append(ciphertext.charAt(i)); // Keep non-letters
unchanged
            }
        }

        return plaintext.toString();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the key: ");
        String key = sc.nextLine();
```

```
        System.out.print("Enter the plaintext: ");
        String plaintext = sc.nextLine();

        String extendedKey = extendKey(key, plaintext.length());
        String encrypted = encrypt(plaintext, key);
        String decrypted = decrypt(encrypted, key);

        System.out.println("\nResults:");
        System.out.println("Message:     " + plaintext);
        System.out.println("Key Text:    " + extendedKey);
        System.out.println("Cipher Text: " + encrypted);
        System.out.println("Plain Text:  " + decrypted);

        sc.close();
    }
}
```

Sample Output:

```
Enter the key: deceptive
Enter the plaintext: wearediscoveredsaveyourself

Results:
Message:     wearediscoveredsaveyourself
Key Text:    deceptivedeceptivedeceptive
Cipher Text: zicvtwqngrzgvtwavzhcqyglmgj
Plain Text:  wearediscoveredsaveyourself
```

Result: Thus a Java program to perform encryption and decryption using Vigenère cipher technique was executed successfully.

Exercise 2.1: Rail Fence Cipher

Aim: To write a Java program to perform encryption and decryption using Rail Fence cipher technique.

Algorithm:

1. Read the plaintext from user
2. For encryption:
    - Separate characters at even positions (0, 2, 4, ...) into first rail
    - Separate characters at odd positions (1, 3, 5, ...) into second rail
    - Concatenate first rail + second rail to form ciphertext
3. For decryption:
    - Split ciphertext into two halves
    - Interleave characters from both halves alternately
    - First half provides even positions, second half provides odd positions
4. Display encrypted and decrypted text

Concept:

- Rail Fence cipher writes plaintext in zigzag pattern across multiple "rails"
- Simple transposition cipher that rearranges character positions
- In 2-rail version: even positions form top rail, odd positions form bottom rail
- Reading rails sequentially produces the ciphertext

Java Code:

```
import java.util.Scanner;

/**
 * Rail Fence Cipher implementation
 * Encrypts text using zigzag pattern across two rails
 */
class RailFenceCipher {

    // Encrypt text using Rail Fence cipher
    public static String encrypt(String plaintext) {
        StringBuilder topRail = new StringBuilder();
        StringBuilder bottomRail = new StringBuilder();

        // Separate characters into two rails
        for (int i = 0; i < plaintext.length(); i++) {
            if (i % 2 == 0) {
                topRail.append(plaintext.charAt(i));
            } else {
                bottomRail.append(plaintext.charAt(i));
            }
        }

        // Concatenate top rail + bottom rail
        return topRail.toString() + bottomRail.toString();
    }

    // Decrypt text using Rail Fence cipher
    public static String decrypt(String ciphertext) {
        StringBuilder plaintext = new StringBuilder();
        int halfLength = (ciphertext.length() + 1) / 2; // Handle odd length

        String topRail = ciphertext.substring(0, halfLength);
        String bottomRail = ciphertext.substring(halfLength);

        // Interleave characters from both rails
        for (int i = 0; i < halfLength; i++) {
            plaintext.append(topRail.charAt(i));
            if (i < bottomRail.length()) {
                plaintext.append(bottomRail.charAt(i));
            }
        }

        return plaintext.toString();
    }

    // Display rail pattern visualization
```

```
    public static void displayRailPattern(String text) {
        System.out.println("\nRail Pattern:");
        StringBuilder topRail = new StringBuilder();
        StringBuilder bottomRail = new StringBuilder();

        for (int i = 0; i < text.length(); i++) {
            if (i % 2 == 0) {
                topRail.append(text.charAt(i)).append(" ");
                bottomRail.append("  ");
            } else {
                topRail.append("  ");
                bottomRail.append(text.charAt(i)).append(" ");
            }
        }

        System.out.println("Top Rail:    " + topRail.toString());
        System.out.println("Bottom Rail: " + bottomRail.toString());
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the message: ");
        String message = sc.nextLine().toLowerCase().replace(" ", "");

        displayRailPattern(message);

        String encrypted = encrypt(message);
        String decrypted = decrypt(encrypted);

        System.out.println("\nResults:");
        System.out.println("Original Text: " + message);
        System.out.println("Cipher Text:   " + encrypted);
        System.out.println("Decrypted:     " + decrypted);

        sc.close();
    }
}
```

Sample Output:

```
Enter the message: meetmeafterthetogaparty

Rail Pattern:
Top Rail:    m e e t m e a f t e r t h e t o g a p a r t y
Bottom Rail:   e   m   a   e   t   t   o   a   a   y

Results:
Original Text: meetmeafterthetogaparty
Cipher Text:   mematrhtgpretefeteoaaty
Decrypted:     meetmeafterthetogaparty
```

Result: Thus a Java program to perform encryption and decryption using Rail Fence cipher technique was executed successfully.

Exercise 2.2: Row Column Transposition Cipher

Aim: To write a Java program to perform encryption and decryption using Row Column Transposition technique.

Algorithm:

1. Read the plaintext and keyword from user
2. For encryption:
    - Remove spaces and convert to uppercase
    - Arrange plaintext in rows with keyword length as column count
    - Pad with dummy characters if needed
    - Assign numbers to keyword letters based on alphabetical order
    - Read columns in the order specified by keyword numbering
    - Concatenate column data to form ciphertext
3. For decryption:
    - Reverse the process by filling columns in keyword order
    - Reconstruct the original matrix row by row
    - Remove padding characters to get plaintext

Concept:

- Columnar transposition rearranges characters based on keyword column order
- More secure than simple Rail Fence as it uses keyword-based permutation
- Characters are written row-wise but read column-wise in specific order
- Key determines the column reading sequence

Java Code:

```
import java.util.Scanner;
import java.util.Arrays;

/**
 * Row Column Transposition Cipher implementation
 * Encrypts text by rearranging characters in keyword-ordered columns
 */
class RowColumnTransposition {

    // Assign numbers to keyword letters based on alphabetical order
    public static int[] getKeywordOrder(String keyword) {
        int[] order = new int[keyword.length()];
        char[] sortedKeyword = keyword.toCharArray();
        Arrays.sort(sortedKeyword);

        for (int i = 0; i < keyword.length(); i++) {
            char currentChar = keyword.charAt(i);
            for (int j = 0; j < sortedKeyword.length; j++) {
                if (currentChar == sortedKeyword[j]) {
```

```
                    order[i] = j + 1;
                    sortedKeyword[j] = '*'; // Mark as used
                    break;
                }
            }
        }
        return order;
    }

    // Get column indices in order of keyword numbering
    public static int[] getColumnOrder(String keyword, int[] keywordOrder) {
        int[] columnOrder = new int[keyword.length()];
        for (int i = 1; i <= keyword.length(); i++) {
            for (int j = 0; j < keywordOrder.length; j++) {
                if (keywordOrder[j] == i) {
                    columnOrder[i - 1] = j;
                    break;
                }
            }
        }
        return columnOrder;
    }

    // Encrypt text using columnar transposition
    public static String encrypt(String plaintext, String keyword) {
        plaintext = plaintext.toUpperCase().replace(" ", "");
        keyword = keyword.toUpperCase();

        // Pad text to fill complete rows
        int extraChars = plaintext.length() % keyword.length();
        if (extraChars != 0) {
            int padding = keyword.length() - extraChars;
            for (int i = 0; i < padding; i++) {
                plaintext += ".";
            }
        }

        int rows = plaintext.length() / keyword.length();
        char[][] matrix = new char[rows][keyword.length()];

        // Fill matrix row by row
        int index = 0;
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < keyword.length(); j++) {
                matrix[i][j] = plaintext.charAt(index++);
            }
        }

        // Display matrix
        System.out.println("\nMatrix:");
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < keyword.length(); j++) {
                System.out.print(matrix[i][j] + " ");
            }
            System.out.println();
```

```
        }

        // Get column reading order
        int[] keywordOrder = getKeywordOrder(keyword);
        int[] columnOrder = getColumnOrder(keyword, keywordOrder);

        // Read columns in keyword order
        StringBuilder ciphertext = new StringBuilder();
        for (int col : columnOrder) {
            for (int row = 0; row < rows; row++) {
                ciphertext.append(matrix[row][col]);
            }
        }

        return ciphertext.toString();
    }

    // Decrypt text using columnar transposition
    public static String decrypt(String ciphertext, String keyword) {
        keyword = keyword.toUpperCase();
        int rows = ciphertext.length() / keyword.length();
        char[][] matrix = new char[rows][keyword.length()];

        // Get column order
        int[] keywordOrder = getKeywordOrder(keyword);
        int[] columnOrder = getColumnOrder(keyword, keywordOrder);

        // Fill matrix column by column in keyword order
        int index = 0;
        for (int col : columnOrder) {
            for (int row = 0; row < rows; row++) {
                matrix[row][col] = ciphertext.charAt(index++);
            }
        }

        // Read matrix row by row
        StringBuilder plaintext = new StringBuilder();
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < keyword.length(); j++) {
                plaintext.append(matrix[i][j]);
            }
        }

        return plaintext.toString().replace(".", "");
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Row Column Transposition Cipher");
        System.out.print("1. Encryption\n2. Decryption\nChoose (1/2): ");
        int choice = sc.nextInt();
        sc.nextLine(); // consume newline

        if (choice == 1) {
```

```
            System.out.print("Enter message: ");
            String message = sc.nextLine();

            System.out.print("Enter keyword: ");
            String keyword = sc.nextLine();

            // Display keyword order
            int[] order = getKeywordOrder(keyword.toUpperCase());
            System.out.println("\nKeyword: " + keyword.toUpperCase());
            System.out.print("Order:   ");
            for (int num : order) {
                System.out.print(num + " ");
            }
            System.out.println();

            String encrypted = encrypt(message, keyword);
            System.out.println("\nCipher Text: " + encrypted);

        } else if (choice == 2) {
            System.out.print("Enter cipher text: ");
            String ciphertext = sc.nextLine();

            System.out.print("Enter keyword: ");
            String keyword = sc.nextLine();

            String decrypted = decrypt(ciphertext, keyword);
            System.out.println("Plain Text: " + decrypted);

        } else {
            System.out.println("Invalid choice!");
        }

        sc.close();
    }
}
```

Sample Output:

```
Row Column Transposition Cipher
1. Encryption
2. Decryption
Choose (1/2): 1
Enter message: I LIKE POTATOES BECAUSE THEY ARE TASTY
Enter keyword: POTATO

Keyword: POTATO
Order:   4 2 5 1 6 3

Matrix:
I L I K E P
O T A T O E
S B E C A U
S E T H E Y
```

```
A R E T A S
T Y . . . .

Cipher Text: KTCHT.LTBERYPEUYS.IOSSATIAETE.EOAEA.
```

Result: Thus a Java program to perform encryption and decryption using Row Column Transposition technique was executed successfully.

Exercise 3: DES (Data Encryption Standard)

Aim: To write a Java program to perform encryption using Data Encryption Standard (DES) algorithm.

Algorithm:

1. Generate a 56-bit DES secret key using KeyGenerator
2. Initialize cipher instances for encryption and decryption modes
3. For encryption:
    - Apply initial permutation to rearrange 64-bit input
    - Perform 16 rounds of substitution and permutation using subkeys
    - Swap left and right halves after final round
    - Apply final permutation (inverse of initial permutation)
4. For decryption:
    - Reverse the encryption process using the same key
    - Apply permutations and substitutions in reverse order
5. Display original text, encrypted data, and decrypted result

Concept:

- DES is a symmetric block cipher operating on 64-bit blocks
- Uses 56-bit key (8 bits for parity) processed through 16 rounds
- Each round involves: expansion, XOR with subkey, S-box substitution, permutation
- Subkeys generated by key schedule algorithm with circular shifts
- Same key used for both encryption and decryption

Java Code:

```
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;

/**
 * DES (Data Encryption Standard) implementation
 * Encrypts/decrypts data using 56-bit symmetric key
 */
```

```
class DESCipher {
    private static SecretKey secretKey;
    private static Cipher encryptCipher;
    private static Cipher decryptCipher;

    // Initialize DES cipher with generated key
    public static void initializeCipher() throws NoSuchAlgorithmException,
            NoSuchPaddingException, InvalidKeyException {

        // Generate DES key
        KeyGenerator keyGenerator = KeyGenerator.getInstance("DES");
        secretKey = keyGenerator.generateKey();

        // Initialize encryption cipher
        encryptCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
        encryptCipher.init(Cipher.ENCRYPT_MODE, secretKey);

        // Initialize decryption cipher
        decryptCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
        decryptCipher.init(Cipher.DECRYPT_MODE, secretKey);

        System.out.println("DES Key Generated: " +
            Base64.getEncoder().encodeToString(secretKey.getEncoded()));
    }

    // Encrypt plaintext using DES
    public static byte[] encryptData(String plaintext) throws
            IllegalBlockSizeException, BadPaddingException {

        System.out.println("Data Before Encryption: " + plaintext);
        byte[] dataToEncrypt = plaintext.getBytes();
        byte[] encryptedData = encryptCipher.doFinal(dataToEncrypt);

        System.out.println("Encrypted Data (Base64): " +
            Base64.getEncoder().encodeToString(encryptedData));

        return encryptedData;
    }

    // Decrypt ciphertext using DES
    public static String decryptData(byte[] encryptedData) throws
            IllegalBlockSizeException, BadPaddingException {

        byte[] decryptedData = decryptCipher.doFinal(encryptedData);
        String decryptedText = new String(decryptedData);

        System.out.println("Decrypted Data: " + decryptedText);
        return decryptedText;
    }

    public static void main(String[] args) {
        try {
            System.out.println("=== DES Encryption Demo ===\n");

            // Initialize DES cipher
```

```
            initializeCipher();

            // Test data
            String originalText = "Classified Information!";

            // Encrypt data
            System.out.println("\n--- Encryption Process ---");
            byte[] encryptedData = encryptData(originalText);

            // Decrypt data
            System.out.println("\n--- Decryption Process ---");
            String decryptedText = decryptData(encryptedData);

            // Verify integrity
            System.out.println("\n--- Verification ---");
            System.out.println("Original Text:  " + originalText);
            System.out.println("Decrypted Text: " + decryptedText);
            System.out.println("Match: " + originalText.equals(decryptedText));

        } catch (NoSuchAlgorithmException e) {
            System.err.println("DES algorithm not available: " + e.getMessage());
        } catch (NoSuchPaddingException e) {
            System.err.println("Padding not available: " + e.getMessage());
        } catch (InvalidKeyException e) {
            System.err.println("Invalid key: " + e.getMessage());
        } catch (IllegalBlockSizeException e) {
            System.err.println("Illegal block size: " + e.getMessage());
        } catch (BadPaddingException e) {
            System.err.println("Bad padding: " + e.getMessage());
        }
    }
}
```

Sample Output:

```
=== DES Encryption Demo ===

DES Key Generated: MEYCQQAwDQYJKoZIhvcNAQEBBQADQgAwPgIhAKB...

--- Encryption Process ---
Data Before Encryption: Classified Information!
Encrypted Data (Base64): 7n8K9mJ4X2k9Qw1vB3RtZg==

--- Decryption Process ---
Decrypted Data: Classified Information!

--- Verification ---
Original Text:  Classified Information!
Decrypted Text: Classified Information!
Match: true
```

Result: Thus a Java program to perform encryption using Data Encryption Standard (DES) algorithm was executed successfully.

Exercise 4: AES (Advanced Encryption Standard)

Aim: To write a Java program to implement AES Algorithm.

Algorithm:

1. **Key Generation:** Derive round keys from the cipher key using key expansion
2. **Initialize State:** Load 128-bit plaintext block into 4x4 state array
3. **Initial Round:** Add the initial round key to state array (XOR operation)
4. **Main Rounds (9 rounds for AES-128):**
    - SubBytes: Apply S-box substitution to each byte
    - ShiftRows: Cyclically shift rows of state array
    - MixColumns: Mix columns using polynomial arithmetic
    - AddRoundKey: XOR state with round key
5. **Final Round (10th round):**
    - SubBytes, ShiftRows, AddRoundKey (no MixColumns)
6. **Output:** Copy final state array as encrypted ciphertext

Concept:

- AES is a symmetric block cipher with 128-bit blocks
- Supports 128, 192, or 256-bit keys (AES-128, AES-192, AES-256)
- Uses substitution-permutation network with multiple rounds
- More secure and efficient than DES
- Industry standard for symmetric encryption

Java Code:

```
import java.util.Base64;
import java.util.Scanner;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;

/**
 * AES (Advanced Encryption Standard) implementation
 * Encrypts/decrypts data using 128/192/256-bit symmetric key
 */
class AESCipher {
    private static Cipher cipher;

    // Initialize AES cipher
    public static void initializeCipher() throws Exception {
        cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
    }

    // Generate AES key of specified size
```

```
    public static SecretKey generateKey(int keySize) throws Exception {
        KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
        keyGenerator.init(keySize); // 128, 192, or 256 bits
        SecretKey secretKey = keyGenerator.generateKey();

        System.out.println("AES-" + keySize + " Key Generated: " +
            Base64.getEncoder().encodeToString(secretKey.getEncoded()));

        return secretKey;
    }

    // Encrypt plaintext using AES
    public static String encrypt(String plaintext, SecretKey secretKey) throws
Exception {
        System.out.println("Plain Text Before Encryption: " + plaintext);

        byte[] plaintextBytes = plaintext.getBytes();
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedBytes = cipher.doFinal(plaintextBytes);

        String encryptedText = Base64.getEncoder().encodeToString(encryptedBytes);
        System.out.println("Encrypted Text After Encryption: " + encryptedText);

        return encryptedText;
    }

    // Decrypt ciphertext using AES
    public static String decrypt(String encryptedText, SecretKey secretKey) throws
Exception {
        byte[] encryptedBytes = Base64.getDecoder().decode(encryptedText);
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decryptedBytes = cipher.doFinal(encryptedBytes);

        String decryptedText = new String(decryptedBytes);
        System.out.println("Decrypted Text After Decryption: " + decryptedText);

        return decryptedText;
    }

    // Display AES algorithm details
    public static void displayAlgorithmInfo() {
        System.out.println("=== AES Algorithm Information ===");
        System.out.println("Block Size: 128 bits (16 bytes)");
        System.out.println("Key Sizes: 128, 192, 256 bits");
        System.out.println("Rounds: AES-128 (10), AES-192 (12), AES-256 (14)");
        System.out.println("Mode: ECB (Electronic Codebook)");
        System.out.println("Padding: PKCS5Padding");
        System.out.println("====================================\n");
    }

    public static void main(String[] args) {
        try {
            displayAlgorithmInfo();

            Scanner sc = new Scanner(System.in);
```

```
            // Initialize cipher
            initializeCipher();

            // Choose key size
            System.out.print("Choose AES key size (128/192/256): ");
            int keySize = sc.hasNextInt() ? sc.nextInt() : 128;
            sc.nextLine(); // consume newline

            // Generate key
            SecretKey secretKey = generateKey(keySize);

            // Get input text
            System.out.print("\nEnter text to encrypt: ");
            String plaintext = sc.nextLine();
            if (plaintext.isEmpty()) {
                plaintext = "AES Symmetric Encryption Decryption";
            }

            System.out.println("\n--- Encryption Process ---");
            String encryptedText = encrypt(plaintext, secretKey);

            System.out.println("\n--- Decryption Process ---");
            String decryptedText = decrypt(encryptedText, secretKey);

            // Verification
            System.out.println("\n--- Verification ---");
            System.out.println("Original:  " + plaintext);
            System.out.println("Decrypted: " + decryptedText);
            System.out.println("Match: " + plaintext.equals(decryptedText));

            sc.close();

        } catch (Exception e) {
            System.err.println("AES Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Sample Output:

```
=== AES Algorithm Information ===
Block Size: 128 bits (16 bytes)
Key Sizes: 128, 192, 256 bits
Rounds: AES-128 (10), AES-192 (12), AES-256 (14)
Mode: ECB (Electronic Codebook)
Padding: PKCS5Padding
====================================

Choose AES key size (128/192/256): 128
AES-128 Key Generated: k3j9mK8xL2nQ5oP7wR1tU4vX6yZ8bC0d
```

```
Enter text to encrypt: AES Symmetric Encryption Decryption

--- Encryption Process ---
Plain Text Before Encryption: AES Symmetric Encryption Decryption
Encrypted Text After Encryption:
sY6vkQrWRg0fvRzbqSAYxepeBIXg4AySj7Xh3x4vDv8TBTkNiTfca7wW/dxiMMJl

--- Decryption Process ---
Decrypted Text After Decryption: AES Symmetric Encryption Decryption

--- Verification ---
Original:  AES Symmetric Encryption Decryption
Decrypted: AES Symmetric Encryption Decryption
Match: true
```

Result: Thus a Java program to implement AES Algorithm was executed successfully.

Exercise 5: RSA Algorithm

Aim: To write a Java program to implement RSA Algorithm.

Algorithm:

Key Generation:

1. Select two distinct prime numbers p and q
2. Calculate n = p × q (modulus for public and private keys)
3. Calculate φ(n) = (p-1)(q-1) (Euler's totient function)
4. Choose e such that $1 < e < \varphi(n)$ and $\gcd(\varphi(n), e) = 1$
5. Calculate d such that $d \equiv e^{-1} \pmod{\varphi(n)}$
6. Public key: (e, n), Private key: (d, n)

Encryption (using Public Key):

- Choose plaintext M where M < n
- Ciphertext: C = M^e mod n

Decryption (using Private Key):

- Plaintext: M = C^d mod n

Concept:

- RSA is an asymmetric (public-key) cryptographic algorithm
- Security based on difficulty of factoring large composite numbers
- Different keys for encryption and decryption
- Widely used for secure data transmission and digital signatures

Java Code:

```java
import java.math.BigInteger;
import java.util.Random;
import java.util.Scanner;

/**
 * RSA Algorithm implementation
 * Asymmetric encryption using public-private key pairs
 */
class RSAAlgorithm {
    private static Scanner sc = new Scanner(System.in);

    // Generate suitable value of e (public exponent)
    public static BigInteger generateE(BigInteger phi) {
        Random random = new Random();
        BigInteger e;

        do {
            // Generate random number between 3 and phi-1
            e = new BigInteger(phi.bitLength() - 1, random);
            // Ensure e is odd and greater than 2
            if (e.compareTo(BigInteger.valueOf(2)) <= 0) {
                e = BigInteger.valueOf(3);
            }
        } while (phi.gcd(e).compareTo(BigInteger.ONE) != 0);

        return e;
    }

    // Check if number is prime (simple check for small numbers)
    public static boolean isPrime(BigInteger num) {
        if (num.compareTo(BigInteger.valueOf(2)) < 0) return false;
        if (num.equals(BigInteger.valueOf(2))) return true;
        if (num.remainder(BigInteger.valueOf(2)).equals(BigInteger.ZERO)) return
false;

        BigInteger sqrt = num.sqrt();
        for (BigInteger i = BigInteger.valueOf(3); i.compareTo(sqrt) <= 0; i =
i.add(BigInteger.valueOf(2))) {
            if (num.remainder(i).equals(BigInteger.ZERO)) return false;
        }
        return true;
    }

    // Encrypt message using public key
    public static BigInteger encrypt(BigInteger message, BigInteger e, BigInteger
n) {
        return message.modPow(e, n);
    }

    // Decrypt ciphertext using private key
    public static BigInteger decrypt(BigInteger ciphertext, BigInteger d,
BigInteger n) {
        return ciphertext.modPow(d, n);
    }
```

```
    // Display RSA process information
    public static void displayRSAInfo() {
        System.out.println("=== RSA Algorithm Demo ===");
        System.out.println("RSA is an asymmetric encryption algorithm");
        System.out.println("Uses separate keys for encryption and decryption");
        System.out.println("Security based on difficulty of factoring large
primes");
        System.out.println("=======================================\n");
    }

    public static void main(String[] args) {
        displayRSAInfo();

        try {
            // Input prime numbers
            System.out.print("Enter first prime number (p): ");
            BigInteger p = sc.nextBigInteger();

            System.out.print("Enter second prime number (q): ");
            BigInteger q = sc.nextBigInteger();

            // Validate inputs
            if (!isPrime(p) || !isPrime(q)) {
                System.out.println("Warning: Please ensure both numbers are prime
for security!");
            }

            if (p.equals(q)) {
                System.out.println("Warning: p and q should be different for
security!");
            }

            // Calculate RSA parameters
            BigInteger n = p.multiply(q);
            BigInteger phi =
p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
            BigInteger e = generateE(phi);
            BigInteger d = e.modInverse(phi);

            // Display keys
            System.out.println("\n--- Key Generation ---");
            System.out.println("p = " + p);
            System.out.println("q = " + q);
            System.out.println("n = p × q = " + n);
            System.out.println("ϕ(n) = (p-1)(q-1) = " + phi);
            System.out.println("e (public exponent) = " + e);
            System.out.println("d (private exponent) = " + d);

            System.out.println("\nPublic Key: (" + e + ", " + n + ")");
            System.out.println("Private Key: (" + d + ", " + n + ")");

            // Encryption
            System.out.print("\nEnter message (number < " + n + "): ");
            BigInteger message = sc.nextBigInteger();
```

```
            if (message.compareTo(n) >= 0) {
                System.out.println("Error: Message must be less than n = " + n);
                return;
            }

            System.out.println("\n--- Encryption Process ---");
            System.out.println("Plaintext: " + message);
            BigInteger ciphertext = encrypt(message, e, n);
            System.out.println("Ciphertext: C = M^e mod n = " + message + "^" + e
+ " mod " + n + " = " + ciphertext);

            // Decryption
            System.out.println("\n--- Decryption Process ---");
            BigInteger decrypted = decrypt(ciphertext, d, n);
            System.out.println("Decrypted: M = C^d mod n = " + ciphertext + "^" +
d + " mod " + n + " = " + decrypted);

            // Verification
            System.out.println("\n--- Verification ---");
            System.out.println("Original Message: " + message);
            System.out.println("Decrypted Message: " + decrypted);
            System.out.println("Match: " + message.equals(decrypted));

        } catch (Exception e) {
            System.err.println("RSA Error: " + e.getMessage());
        } finally {
            sc.close();
        }
    }
}
```

Sample Output:

```
=== RSA Algorithm Demo ===
RSA is an asymmetric encryption algorithm
Uses separate keys for encryption and decryption
Security based on difficulty of factoring large primes
=====================================

Enter first prime number (p): 7
Enter second prime number (q): 11

--- Key Generation ---
p = 7
q = 11
n = p × q = 77
ϕ(n) = (p-1)(q-1) = 60
e (public exponent) = 43
d (private exponent) = 7

Public Key: (43, 77)
Private Key: (7, 77)
```

```
Enter message (number < 77): 6

--- Encryption Process ---
Plaintext: 6
Ciphertext: C = M^e mod n = 6^43 mod 77 = 62

--- Decryption Process ---
Decrypted: M = C^d mod n = 62^7 mod 77 = 6

--- Verification ---
Original Message: 6
Decrypted Message: 6
Match: true
```

Result: Thus a Java program to implement RSA Algorithm was executed successfully.

Exercise 6: Digital Signature Standard (DSS)

Aim: To write a Java program to sign a document using Digital Signature Algorithm (DSA).

Algorithm:

1. **Key Pair Generation:**
    - Generate DSA key pair (1024-bit) using SecureRandom
    - Extract private key for signing and public key for verification
2. **Digital Signature Creation:**
    - Create Signature object using SHA1withDSA algorithm
    - Initialize signature with private key
    - Read document data and update signature object
    - Generate digital signature for the document
3. **Save Outputs:**
    - Save generated signature to file
    - Save public key to file for verification
4. **Verification Process:**
    - Load public key and signature from files
    - Verify signature against original document

Concept:

- Digital signatures provide authentication, integrity, and non-repudiation
- DSA uses SHA-1 hash with Digital Signature Algorithm
- Private key creates signature, public key verifies it
- Ensures document hasn't been tampered with and confirms sender identity

Java Code:

```
import java.io.*;
import java.security.*;
import java.security.spec.*;
```

```
import java.util.Scanner;

/**
 * Digital Signature Standard (DSS) implementation
 * Signs documents using DSA and verifies signatures
 */
class DigitalSignature {

    // Generate DSA key pair
    public static KeyPair generateKeyPair() throws Exception {
        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA", "SUN");
        SecureRandom random = SecureRandom.getInstance("SHA1PRNG", "SUN");
        keyGen.initialize(1024, random);

        KeyPair pair = keyGen.generateKeyPair();
        System.out.println("✓ DSA Key pair generated successfully");

        return pair;
    }

    // Sign a document
    public static byte[] signDocument(String fileName, PrivateKey privateKey)
throws Exception {
        // Create signature object
        Signature dsa = Signature.getInstance("SHA1withDSA", "SUN");
        dsa.initSign(privateKey);

        // Read and update document data
        FileInputStream fis = new FileInputStream(fileName);
        BufferedInputStream bufin = new BufferedInputStream(fis);
        byte[] buffer = new byte[1024];
        int len;

        System.out.println("Reading document: " + fileName);
        while ((len = bufin.read(buffer)) != -1) {
            dsa.update(buffer, 0, len);
        }
        bufin.close();

        // Generate signature
        byte[] signature = dsa.sign();
        System.out.println("✓ Document signed successfully");

        return signature;
    }

    // Verify document signature
    public static boolean verifySignature(String fileName, byte[] signature,
PublicKey publicKey) throws Exception {
        // Create signature object for verification
        Signature dsa = Signature.getInstance("SHA1withDSA", "SUN");
        dsa.initVerify(publicKey);

        // Read and update document data
        FileInputStream fis = new FileInputStream(fileName);
```

```
        BufferedInputStream bufin = new BufferedInputStream(fis);
        byte[] buffer = new byte[1024];
        int len;

        while ((len = bufin.read(buffer)) != -1) {
            dsa.update(buffer, 0, len);
        }
        bufin.close();

        // Verify signature
        boolean isValid = dsa.verify(signature);
        System.out.println("Signature verification: " + (isValid ? "✓ VALID" : "✗
INVALID"));

        return isValid;
    }

    // Save signature to file
    public static void saveSignature(byte[] signature, String fileName) throws
Exception {
        FileOutputStream sigfos = new FileOutputStream(fileName);
        sigfos.write(signature);
        sigfos.close();
        System.out.println("✓ Signature saved to: " + fileName);
    }

    // Save public key to file
    public static void savePublicKey(PublicKey publicKey, String fileName) throws
Exception {
        byte[] keyBytes = publicKey.getEncoded();
        FileOutputStream keyfos = new FileOutputStream(fileName);
        keyfos.write(keyBytes);
        keyfos.close();
        System.out.println("✓ Public key saved to: " + fileName);
    }

    // Load public key from file
    public static PublicKey loadPublicKey(String fileName) throws Exception {
        FileInputStream keyfis = new FileInputStream(fileName);
        byte[] keyBytes = new byte[keyfis.available()];
        keyfis.read(keyBytes);
        keyfis.close();

        X509EncodedKeySpec keySpec = new X509EncodedKeySpec(keyBytes);
        KeyFactory keyFactory = KeyFactory.getInstance("DSA");
        PublicKey publicKey = keyFactory.generatePublic(keySpec);

        System.out.println("✓ Public key loaded from: " + fileName);
        return publicKey;
    }

    // Load signature from file
    public static byte[] loadSignature(String fileName) throws Exception {
        FileInputStream sigfis = new FileInputStream(fileName);
        byte[] signature = new byte[sigfis.available()];
```

```
        sigfis.read(signature);
        sigfis.close();

        System.out.println("✓ Signature loaded from: " + fileName);
        return signature;
    }

    // Create test document
    public static void createTestDocument(String fileName) throws Exception {
        FileWriter writer = new FileWriter(fileName);
        writer.write("This is a test document for digital signature.\n");
        writer.write("Content: Confidential information that needs
authentication.\n");
        writer.write("Date: " + new java.util.Date() + "\n");
        writer.close();
        System.out.println("✓ Test document created: " + fileName);
    }

    public static void main(String[] args) {
        try {
            System.out.println("=== Digital Signature Standard Demo ===\n");

            Scanner sc = new Scanner(System.in);

            // Get document filename
            System.out.print("Enter document filename (or press Enter for test
document): ");
            String fileName = sc.nextLine().trim();

            if (fileName.isEmpty()) {
                fileName = "test_document.txt";
                createTestDocument(fileName);
            }

            // Check if file exists
            File file = new File(fileName);
            if (!file.exists()) {
                System.out.println("Error: File '" + fileName + "' not found!");
                return;
            }

            System.out.println("\n--- Key Generation ---");
            KeyPair keyPair = generateKeyPair();
            PrivateKey privateKey = keyPair.getPrivate();
            PublicKey publicKey = keyPair.getPublic();

            System.out.println("\n--- Document Signing ---");
            byte[] signature = signDocument(fileName, privateKey);

            System.out.println("\n--- Saving Files ---");
            saveSignature(signature, "document.sig");
            savePublicKey(publicKey, "public.key");

            System.out.println("\n--- Signature Verification ---");
            // Load saved files and verify
```

```
            PublicKey loadedPublicKey = loadPublicKey("public.key");
            byte[] loadedSignature = loadSignature("document.sig");

            boolean isValid = verifySignature(fileName, loadedSignature,
loadedPublicKey);

            System.out.println("\n--- Summary ---");
            System.out.println("Document: " + fileName);
            System.out.println("Signature File: document.sig");
            System.out.println("Public Key File: public.key");
            System.out.println("Signature Status: " + (isValid ? "VERIFIED ✓" :
"FAILED ✗"));

            sc.close();

        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Sample Output:

```
=== Digital Signature Standard Demo ===

Enter document filename (or press Enter for test document):

✓ Test document created: test_document.txt

--- Key Generation ---
✓ DSA Key pair generated successfully

--- Document Signing ---
Reading document: test_document.txt
✓ Document signed successfully

--- Saving Files ---
✓ Signature saved to: document.sig
✓ Public key saved to: public.key

--- Signature Verification ---
✓ Public key loaded from: public.key
✓ Signature loaded from: document.sig
Signature verification: ✓ VALID

--- Summary ---
Document: test_document.txt
Signature File: document.sig
Public Key File: public.key
Signature Status: VERIFIED ✓
```

Result: Thus a Java program to sign a document using Digital Signature Algorithm was executed successfully.

Exercise 7: Intrusion Detection System (IDS) using Snort

Date: ___

Aim: To demonstrate Intrusion Detection System (IDS) using SNORT in different operational modes.

Algorithm

1. **Installation and Setup:**
    - Download SNORT from snort.org
    - Install SNORT with required components
    - Configure system environment variables
    - Set up logging directories
2. **Sniffer Mode Configuration:**
    - Use basic packet sniffing commands
    - Monitor network traffic in real-time
    - Display packet headers and data
3. **Packet Logger Mode Setup:**
    - Configure logging directory
    - Set up packet capture and storage
    - Apply network filters for specific IP ranges
4. **NIDS Mode Implementation:**
    - Configure snort.conf rule file
    - Apply intrusion detection rules
    - Monitor and log suspicious activities
5. **Testing and Verification:**
    - Generate network traffic
    - Verify packet capture and logging
    - Analyze detection results

Concept

SNORT Operational Modes:

- **Sniffer Mode:** Real-time packet monitoring and display
- **Packet Logger Mode:** Captures and stores network packets to disk
- **Network Intrusion Detection System (NIDS) Mode:** Applies rules to detect intrusions

Key Components:

- **Packet Decoder:** Analyzes different protocol layers

- **Preprocessors:** Normalize and prepare packets for detection engine
- **Detection Engine:** Applies rules to identify suspicious activities
- **Logging and Alerting:** Records and reports security events
- **Rule Database:** Contains signatures for known attacks and anomalies

Implementation

Step 1: SNORT Installation

Download and Install:

1. Visit snort.org and download latest SNORT version
2. Run installer as Administrator
3. Select installation components:
    - SNORT executable
    - Configuration files
    - Rule sets
    - Documentation

Installation Configuration:

```
# During installation:
- Select all components ✓
- Choose installation directory: C:\snort\
- Configure WinPcap (if required)
- Complete installation process
```

Step 2: Environment Setup

Configure System PATH:

1. Open System Properties → Advanced → Environment Variables
2. Create new system variable:
    - Variable name: `PATH`
    - Variable value: `C:\snort\bin`
3. Click OK to save changes

Verify Installation:

```
# Open Command Prompt as Administrator
C:\> snort --version

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.20 GRE (Build 82)
   ''''    By Martin Roesch & The Snort Team

# Verify successful installation
```

Step 3: Directory Structure Setup

Create Required Directories:

```
# Create logging directory
C:\> mkdir C:\log

# Create configuration directory
C:\> mkdir C:\snort\etc

# Create rules directory
C:\> mkdir C:\snort\rules

# Verify directory structure
C:\> dir C:\snort
```

Step 4: Operational Mode Implementation

Mode 1: Sniffer Mode

Basic Packet Sniffing:

```
# Display TCP/IP packet headers
C:\> snort -v

# Output:
10/14/25-15:30:25.123456  192.168.1.100:80 -> 192.168.1.50:1234
TCP TTL:64 TOS:0x0 ID:54321 IpLen:20 DgmLen:52
***A**** Seq: 0x12345678  Ack: 0x87654321  Win: 0x1000  TcpLen: 32

# Show headers with application data
C:\> snort -vd

# Output includes:
- Ethernet headers
- IP headers
- TCP/UDP headers
- Application layer data (HTTP, FTP, etc.)
```

Enhanced Sniffer Output:

```
# Verbose mode with data link layer
C:\> snort -vde

# Sample Output:
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
10/14/25-15:30:25.456789  192.168.1.100 -> 192.168.1.50
Ethernet Header:
  Src MAC: 00:1B:44:11:3A:B7
```

```
  Dst MAC: 00:50:56:C0:00:01
  Type: 0x800 (IP)

IP Header:
  Version: 4  Header Length: 20 bytes
  Type of Service: 0x00
  Total Length: 84 bytes
  Identification: 0xD431
  Flags: 0x40 (Don't Fragment)
  Fragment Offset: 0x00
  Time to Live: 64
  Protocol: 6 (TCP)
  Header Checksum: 0x1234
  Source IP: 192.168.1.100
  Destination IP: 192.168.1.50

TCP Header:
  Source Port: 80 (HTTP)
  Destination Port: 1234
  Sequence Number: 0x12345678
  Acknowledgment: 0x87654321
  Data Offset: 8 (32 bytes)
  Flags: 0x18 (PSH ACK)
  Window Size: 4096
  Checksum: 0x5678
  Urgent Pointer: 0x0000

Application Data:
GET / HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
```

Mode 2: Packet Logger Mode

Basic Packet Logging:

```
# Log all packets to directory
C:\> snort -dev -l C:\log

# This command:
- Captures all network packets
- Logs to C:\log directory
- Includes data link, network, and application layers
- Creates separate files for different connections
```

Filtered Packet Logging:

```
# Log packets for specific network
C:\> snort -dev -l C:\log -h 192.168.1.0/24
```

```
# Command breakdown:
- -d: Dump application layer data
- -e: Show data link layer headers
- -v: Verbose packet information
- -l: Log to specified directory
- -h: Home network specification
```

Binary Logging Mode:

```
# Log everything to single binary file
C:\> snort -l C:\log -b

# Creates unified binary log file:
# C:\log\snort.log.timestamp
# Efficient storage format
# Can be analyzed with snort or other tools
```

Log File Analysis:

```
# Read binary log file
C:\> snort -r C:\log\snort.log.1634218825

# Filter and analyze logs
C:\> snort -r C:\log\snort.log.1634218825 -c snort.conf
```

Mode 3: Network Intrusion Detection System (NIDS)

Basic NIDS Configuration:

```
# Run SNORT in NIDS mode
C:\> snort -d -h 192.168.1.0/24 -l C:\log -c C:\snort\etc\snort.conf

# Command explanation:
- -d: Show application data
- -h: Define home network
- -l: Specify log directory
- -c: Use configuration file with rules
```

SNORT Configuration File (snort.conf):

```
# snort.conf - Basic configuration

# Network Variables
var HOME_NET 192.168.1.0/24
var EXTERNAL_NET !$HOME_NET
var HTTP_SERVERS $HOME_NET
```

```
var SMTP_SERVERS $HOME_NET
var DNS_SERVERS $HOME_NET

# Port Variables
var HTTP_PORTS 80
var SHELLCODE_PORTS !80
var ORACLE_PORTS 1521

# Rule Paths
var RULE_PATH C:\snort\rules
var SO_RULE_PATH C:\snort\so_rules
var PREPROC_RULE_PATH C:\snort\preproc_rules

# Output Configuration
output alert_fast: C:\log\alert.txt
output log_tcpdump: C:\log\snort.log

# Include Rules
include $RULE_PATH\local.rules
include $RULE_PATH\icmp.rules
include $RULE_PATH\http.rules
```

Sample Detection Rules:

```
# local.rules - Custom detection rules

# Detect ICMP ping sweep
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP Ping Sweep"; sid:1000001;
rev:1;)

# Detect suspicious HTTP traffic
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"Suspicious HTTP
Request"; content:"cmd.exe"; sid:1000002; rev:1;)

# Detect port scanning
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"Port Scan Detected"; flags:S;
detection_filter:track by_src, count 10, seconds 60; sid:1000003; rev:1;)

# Detect SQL injection attempt
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"SQL Injection
Attempt"; content:"union select"; nocase; sid:1000004; rev:1;)

# Detect suspicious outbound traffic
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Suspicious Outbound
Connection"; content:"password"; sid:1000005; rev:1;)
```

Step 5: Advanced NIDS Configuration

Complete NIDS Setup:

```
# Full NIDS mode with all features
C:\> snort -A console -q -u snort -g snort -c C:\snort\etc\snort.conf -i 1

# Parameters:
- -A console: Alert to console
- -q: Quiet mode (reduce output)
- -u: Run as specific user
- -g: Run as specific group
- -i: Interface number (1 = first network interface)
```

Real-time Monitoring:

```
# Monitor with real-time alerts
C:\> snort -A full -c C:\snort\etc\snort.conf -l C:\log

# Monitor specific interface
C:\> snort -A full -c C:\snort\etc\snort.conf -i "Local Area Connection"
```

Step 6: Testing and Verification

Generate Test Traffic:

```
# From another machine, generate traffic:
ping 192.168.1.100
nmap -sS 192.168.1.100
curl http://192.168.1.100/test?id=1'union select 1,2,3--
```

Monitor Alerts:

```
# View real-time alerts
C:\> type C:\log\alert.txt

# Sample alert output:
[**] [1:1000001:1] ICMP Ping Sweep [**]
[Priority: 3]
10/14/25-15:45:30.123456 192.168.1.200 -> 192.168.1.100
ICMP TTL:64 TOS:0x0 ID:12345 IpLen:28 DgmLen:56

[**] [1:1000003:1] Port Scan Detected [**]
[Priority: 2]
10/14/25-15:46:15.789012 192.168.1.200:45678 -> 192.168.1.100:80
TCP TTL:64 TOS:0x0 ID:23456 IpLen:20 DgmLen:40
***A**** Seq: 0x0  Ack: 0x0  Win: 0x2000  TcpLen: 20
```

Log Analysis:

```
# Analyze captured packets
C:\> snort -r C:\log\snort.log

# Generate statistics
C:\> snort -r C:\log\snort.log -A console | findstr "Priority"

# Count alerts by type
C:\> type C:\log\alert.txt | findstr /C:"[**]" | find /C /V ""
```

Key Features and Capabilities

Detection Methods

1. **Signature-based Detection:**
   - Pattern matching against known attack signatures
   - Fast and accurate for known threats
   - Requires regular rule updates
2. **Anomaly-based Detection:**
   - Statistical analysis of network behavior
   - Detects unknown attacks and variations
   - Higher false positive rate
3. **Protocol Analysis:**
   - Deep packet inspection
   - Protocol compliance checking
   - Application layer analysis

Troubleshooting

Common Issues:

1. **WinPcap Driver Issues:**

```
# Reinstall WinPcap
# Download from winpcap.org
# Install as Administrator
```

2. **Permission Errors:**

```
# Run Command Prompt as Administrator
# Check firewall settings
# Verify user permissions
```

3. **Configuration Errors:**

```
# Test configuration
C:\> snort -T -c C:\snort\etc\snort.conf

# Check for syntax errors
# Verify file paths
# Validate rule syntax
```

Best Practices

1. **Rule Management:**
    - Regular rule updates
    - Custom rule testing
    - Performance impact assessment
2. **Log Management:**
    - Regular log rotation
    - Automated analysis
    - Storage capacity planning
3. **Performance Optimization:**
    - Interface tuning
    - CPU affinity settings
    - Memory allocation optimization
4. **Security Considerations:**
    - Secure rule distribution
    - Access control implementation
    - Encrypted communications

Result

Thus the Intrusion Detection System (IDS) using SNORT was successfully demonstrated in all three operational modes:

- Sniffer mode for real-time packet monitoring
- Packet Logger mode for traffic capture and storage
- NIDS mode for automated intrusion detection and alerting

The system effectively monitors network traffic, detects suspicious activities, and provides comprehensive logging capabilities for security analysis.

Exercise 8: Virtual Machine Setup and C Programming

Aim: To install VirtualBox/VMware Workstation with different flavours of Linux or Windows OS on top of Windows 7/8/10 and execute simple C programs.

Algorithm:

1. **VMware Installation:**
    - Download VMware Workstation from official website
    - Run installation wizard with proper configuration
    - Accept license agreement and configure installation paths
    - Set user experience settings and shortcuts
    - Complete installation and restart system
2. **Ubuntu Virtual Machine Setup:**
    - Create new virtual machine with custom settings
    - Configure ISO image path for Ubuntu installation
    - Set memory allocation (8GB) and storage (110GB)
    - Configure network settings (bridged networking)
    - Install Ubuntu operating system
3. **C Programming Environment:**
    - Update package repositories in Ubuntu
    - Install GCC compiler
    - Create, compile, and execute C programs

Concept:

- **Virtualization:** Technology that allows running multiple operating systems on a single physical machine
- **Hypervisor:** Software layer that manages virtual machines (VMware Workstation, VirtualBox)
- **Guest OS:** Operating system running inside virtual machine (Ubuntu)
- **Host OS:** Primary operating system on physical machine (Windows)
- **Cross-platform Development:** Developing software on different operating systems

Installation Steps:

VMware Workstation Installation:

1. Download VMware-workstation-full-16.1.2 from http://www.vmware.com
2. Run the downloaded .exe file to start installation wizard
3. Accept license agreement and click Next
4. Choose installation directory (default C: drive or custom location)
5. Enable options:
    - Enhanced Keyboard Driver
    - Add VMware Workstation console tools to system PATH
6. Configure User Experience Settings:
    - Check for product updates on startup
    - Join VMware Customer Experience Improvement Program
7. Create shortcuts:
    - Desktop shortcut
    - Start Menu Program Folder

8. Click Install to begin installation process
9. Restart system when prompted
10. Enter license key: `ZF3R0-FHED2-M80TY-8QYGC-NPKYF`
11. Complete setup and launch VMware Workstation

Ubuntu Virtual Machine Creation:

1. Visit https://ubuntu.com/download/desktop
2. Launch VMware Workstation
3. Click "Create a New Virtual Machine"
4. Select "Custom (advanced)" option
5. Choose "Installer disk image file (iso)" and browse to Ubuntu ISO
6. Set credentials:
    - Full name: ubuntu
    - Username: ubuntu
    - Password: cse123
7. Configure virtual machine settings:
    - Name: Ubuntu (or custom name)
    - Location: Default or custom path
    - Memory: 8GB (8192MB)
    - Network: Use bridged networking
    - SCSI Controller: LSI Logic
    - Virtual Disk Type: SCSI
    - Disk: Create new virtual disk (110GB, split into multiple files)
8. Power on virtual machine after creation
9. Complete Ubuntu installation process
10. Restart and login with credentials

C Programming Setup:

```
# Update package repositories
sudo apt update
sudo apt upgrade -y

# Install GCC compiler
sudo apt install gcc

# Create C program file
nano hello.c

# Sample C program content:
#include<stdio.h>
int main()
{
    printf("Hello World!\n");
    return 0;
}

# Compile the program
gcc hello.c
```

```
# Execute the program
./a.out
```

Sample C Program:

```c
#include<stdio.h>

int main()
{
    printf("Hello World from Virtual Machine!\n");
    printf("This program is running on Ubuntu in VMware\n");
    return 0;
}
```

Sample Output:

```
# Compilation process
$ gcc hello.c
$ ./a.out
Hello World from Virtual Machine!
This program is running on Ubuntu in VMware

# System information
$ uname -a
Linux ubuntu 5.11.0-27-generic #29~20.04.1-Ubuntu SMP Wed Aug 11 15:58:17 UTC 2021
x86_64 x86_64 x86_64 GNU/Linux

$ gcc --version
gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
```

Key Configuration Details:

- **Memory Allocation:** 8GB RAM for smooth Ubuntu operation
- **Storage:** 110GB virtual disk with SCSI interface
- **Network:** Bridged networking for external network access
- **Compiler:** GCC (GNU Compiler Collection) for C programming
- **File System:** Ext4 file system in Ubuntu virtual environment

Advantages of Virtual Machines:

1. **Isolation:** Separate environment for testing and development
2. **Multi-OS Support:** Run multiple operating systems simultaneously
3. **Snapshot Feature:** Save and restore system states
4. **Resource Management:** Allocate specific CPU, memory, and storage
5. **Security:** Contained environment reduces host system risks

Result: Thus the installation of VMware with Ubuntu 20.04 OS on top of Windows 7/8/10 and execution of simple C programs was completed successfully.

Exercise 9: Cloud Simulation and Custom Scheduling Algorithm

Aim: To simulate a cloud scenario using CloudSim and run a scheduling algorithm that is not present in CloudSim.

Algorithm:

1. **CloudSim Setup:**
    - Download and extract CloudSim 3.0.3 framework
    - Download and configure Commons Math 3.6.1 library
    - Set up Eclipse IDE project with proper dependencies
    - Configure Java 1.8 environment
2. **Default Scheduling (First Come First Serve):**
    - Bind cloudlets to VMs in round-robin fashion
    - Simple index-based assignment without load consideration
    - Sequential allocation based on arrival order
3. **Custom Scheduling (Shortest Job First):**
    - Calculate current load on each VM
    - Find VM with minimum current load
    - Assign new cloudlet to least loaded VM
    - Update VM load after each assignment
4. **Simulation Execution:**
    - Create datacenter with configured resources
    - Deploy VMs with specified characteristics
    - Submit cloudlets with different lengths
    - Execute simulation and analyze results

Concept:

- **Cloud Simulation:** Modeling cloud computing environments to test resource allocation strategies
- **CloudSim Framework:** Java-based toolkit for modeling cloud computing infrastructure
- **Scheduling Algorithms:** Methods to assign computational tasks to available resources
- **Load Balancing:** Distributing workload evenly across multiple computing resources
- **Virtual Machines (VMs):** Virtualized computing resources with defined CPU, memory, and storage
- **Cloudlets:** Computational tasks submitted to cloud infrastructure for execution

CloudSim Installation Steps:

1. **Download Required Components:**
    - CloudSim 3.0.3 framework from official repository
    - Commons Math 3.6.1 library for mathematical operations

2. **Setup Dependencies:**

```
# Extract CloudSim-3.0.3.zip
# Extract commons-math-3.6.1-bin.zip
# Copy commons-math-3.6.1.jar to CloudSim-3.0.3/jars/
```

3. **Eclipse Project Configuration:**

   - Open Eclipse IDE
   - Create new Java Project
   - Set project name (e.g., "CloudIntro")
   - Uncheck "Use default location"
   - Browse and set path to CloudSim-3.0.3 directory
   - Select Java 1.8 JRE environment
   - Import all JAR files from CloudSim framework
   - Complete project setup

Java Implementation:

First Come First Serve Scheduling:

```
/**
 * Simple Round-Robin scheduling algorithm
 * Assigns cloudlets to VMs in sequential order
 */
public void bindCloudletsToVmsSimple() {
    int cloudletNum = cloudletList.size();
    int vmNum = vmList.size();
    int idx = 0;

    // Round-robin assignment
    for (int i = 0; i < cloudletNum; i++) {
        cloudletList.get(i).setVmId(vmList.get(idx).getId());
        idx = (idx + 1) % vmNum;
    }
}
```

Custom Shortest Job First Scheduling:

```
/**
 * Load-aware scheduling algorithm
 * Assigns cloudlets to VM with minimum current load
 */
public void bindCloudletToVmsScheduling() {
    int cloudletNum = cloudletList.size();
    int vmNum = vmList.size();
    double[] vmLoad = new double[vmNum];
    int idx = 0;
```

```
    // Assign first cloudlet to first VM
    cloudletList.get(0).setVmId(vmList.get(0).getId());
    vmLoad[0] += cloudletList.get(0).getCloudletLength() /
vmList.get(0).getMips();

    // Assign remaining cloudlets to least loaded VM
    for (int j = 1; j < cloudletNum; j++) {
        // Find VM with minimum load
        for (int i = 0; i < vmNum; i++) {
            if (vmLoad[i] < vmLoad[idx]) {
                idx = i;
            }
        }

        // Assign cloudlet to least loaded VM
        cloudletList.get(j).setVmId(vmList.get(idx).getId());
        vmLoad[idx] += cloudletList.get(j).getCloudletLength() /
vmList.get(idx).getMips();
    }
}
```

Integration Steps:

1. **Modify DatacenterBroker.java:**
   - Add both scheduling methods to DatacenterBroker class
   - Import necessary CloudSim packages
   - Ensure proper access to cloudletList and vmList
2. **Update Example Code:**

```
// Replace default scheduling with custom algorithm
broker.bindCloudletToVmsScheduling();
```

3. **Execution Process:**
   - Compile the project in Eclipse
   - Run the extended example
   - Observe scheduling behavior in console output

Sample Output:

```
========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish
Time
0              SUCCESS   2                 0        60      0.1           60.1
2              SUCCESS   2                 0        70      60.1          130.1
1              SUCCESS   2                 1        140     0.1           140.1

ExtendedExample finished!
```

```
========== SIMULATION STATISTICS ==========
Total Simulation Time: 140.1 seconds
Number of Cloudlets: 3
Number of VMs: 2
Average Execution Time: 90.03 seconds
Load Distribution:
- VM 0: 130.0 seconds (2 cloudlets)
- VM 1: 140.0 seconds (1 cloudlet)
```

Algorithm Comparison:

Metric	FCFS (Round-Robin)	Custom (Load-Aware)
Complexity	O(n)	$O(n^2)$
Load Balance	Poor	Better
Response Time	Variable	Optimized
Throughput	Standard	Improved

Key CloudSim Components:

- **Datacenter:** Physical infrastructure hosting VMs
- **DatacenterBroker:** Mediates between users and cloud services
- **VM (Virtual Machine):** Computational resource with defined capabilities
- **Cloudlet:** Computational task with specified length and requirements
- **Host:** Physical machine hosting multiple VMs

Advanced Features:

1. **Dynamic Load Balancing:** Real-time adjustment based on current VM utilization
2. **Priority-based Scheduling:** Consider cloudlet priorities in assignment decisions
3. **Resource Monitoring:** Track CPU, memory, and bandwidth utilization
4. **Cost Optimization:** Minimize execution cost while meeting performance requirements

Result: Thus the cloud scenario using CloudSim to run a custom scheduling algorithm that is not present in CloudSim was executed successfully.

Exercise 10: Virtual Machine File Transfer

Aim: To transfer files from one virtual machine to another virtual machine.

Algorithm:

1. **Shared Folder Configuration:**
    - Access VMware virtual machine settings
    - Enable shared folders functionality
    - Configure folder sharing between host and guest OS
    - Set appropriate permissions and attributes

2. **Host-Guest Communication Setup:**
    - Install VMware Tools for enhanced integration
    - Configure shared folder paths and mounting points
    - Verify connectivity between host and guest systems
3. **File Operations:**
    - Create directories in shared space
    - Copy files between host and guest systems
    - Move and organize files within shared folders
    - Verify file integrity after transfer
4. **Cross-Platform Verification:**
    - Confirm file accessibility from both systems
    - Test file permissions and ownership
    - Validate content consistency across platforms

Concept:

- **Virtual Machine Communication:** Methods for data exchange between virtual and physical systems
- **Shared Folders:** VMware feature enabling seamless file sharing between host and guest OS
- **HGFS (Host-Guest File System):** VMware's file system protocol for shared folder access
- **Cross-Platform File Systems:** Supporting file operations across different operating systems
- **VMware Tools:** Enhanced drivers and utilities for improved VM functionality

Detailed Procedure:

Step 1: VMware Configuration

1. **Access Virtual Machine Settings:**
    - Open VMware Workstation
    - Login to Ubuntu virtual machine
    - Right-click on "Ubuntu 64-bit" from the left panel
    - Select "Settings" from context menu
2. **Configure Shared Folders:**
    - Navigate to "Options" tab in Virtual Machine Settings
    - Click on "Shared Folders" option
    - Select "Always enabled" radio button
    - Click "Add" to create new shared folder
3. **Add Shared Folder Wizard:**
    - Click "Next" in the Add Shared Folder wizard
    - Click "Browse" to select host folder for sharing
    - Choose desired folder from host filesystem
    - Click "Next" to proceed
4. **Set Folder Attributes:**

- In "Specify Shared Folder Attributes" dialog
- Check "Enable this share" checkbox
- Configure read/write permissions as needed
- Click "Finish" to complete setup
- Click "OK" to close Virtual Machine Settings

Step 2: Guest OS Configuration

1. **Verify Shared Folder Access:**

```
# Login to Ubuntu and open terminal
# Check available shared folders
ubuntu@ubuntu:~$ vmware-hgfsclient
Linux_shared_files
```

2. **Navigate to Shared Directory:**

```
# Access shared folders through mount point
ubuntu@ubuntu:~$ cd /mnt/hgfs/
ubuntu@ubuntu:/mnt/hgfs$ ls
Linux_shared_files

# Alternative path for newer VMware versions
# /computer/mnt/hgfs/Linux_shared_files
```

Step 3: File Operations

1. **Create Directory Structure:**

```
# Navigate to shared folder
ubuntu@ubuntu:~$ cd /mnt/hgfs/Linux_shared_files/

# Create new directory
ubuntu@ubuntu:/mnt/hgfs/Linux_shared_files$ mkdir hadoop

# Verify directory creation
ubuntu@ubuntu:/mnt/hgfs/Linux_shared_files$ ls
h1 hadoop Hadoop_1
```

2. **Move Files Between Directories:**

```
# Move file to newly created directory
ubuntu@ubuntu:/mnt/hgfs/Linux_shared_files$ mv Hadoop_1
/mnt/hgfs/Linux_shared_files/hadoop/

# Verify file movement
ubuntu@ubuntu:/mnt/hgfs/Linux_shared_files$ ls
```

```
h1 hadoop

# Check contents of hadoop directory
ubuntu@ubuntu:/mnt/hgfs/Linux_shared_files$ ls hadoop/
Hadoop_1
```

3. **Copy and Create Files:**

```
# Copy file with new name
ubuntu@ubuntu:/mnt/hgfs/Linux_shared_files$ cp h1 hadoop_commands.txt

# Verify copy operation
ubuntu@ubuntu:/mnt/hgfs/Linux_shared_files$ ls
h1 hadoop hadoop_commands.txt

# Check file contents
ubuntu@ubuntu:/mnt/hgfs/Linux_shared_files$ cat h1
ubuntu@ubuntu:/mnt/hgfs/Linux_shared_files$ cat hadoop_commands.txt
```

File Transfer Methods:

Method 1: VMware Shared Folders (Recommended)

- Seamless integration between host and guest
- Real-time synchronization
- No network configuration required
- Built-in security through VMware Tools

Sample Commands and Output:

```
# Initial setup verification
ubuntu@ubuntu:~$ vmware-hgfsclient
Linux_shared_files

# Directory operations
ubuntu@ubuntu:~$ cd /mnt/hgfs/Linux_shared_files/
ubuntu@ubuntu:/mnt/hgfs/Linux_shared_files$ ls -la
total 16
drwxrwxrwx 1 root root  4096 Oct 14 10:30 .
drwxr-xr-x 3 root root  4096 Oct 14 10:25 ..
-rw-rw-rw- 1 root root   256 Oct 14 10:20 h1
-rw-rw-rw- 1 root root  1024 Oct 14 10:25 Hadoop_1

# Create directory and move files
ubuntu@ubuntu:/mnt/hgfs/Linux_shared_files$ mkdir hadoop
ubuntu@ubuntu:/mnt/hgfs/Linux_shared_files$ mv Hadoop_1 hadoop/
ubuntu@ubuntu:/mnt/hgfs/Linux_shared_files$ cp h1 hadoop_commands.txt

# Final verification
ubuntu@ubuntu:/mnt/hgfs/Linux_shared_files$ ls -la
```

```
total 20
drwxrwxrwx 1 root root  4096 Oct 14 10:35 .
drwxr-xr-x 3 root root  4096 Oct 14 10:25 ..
-rw-rw-rw- 1 root root   256 Oct 14 10:20 h1
drwxrwxrwx 1 root root  4096 Oct 14 10:32 hadoop
-rw-rw-rw- 1 root root   256 Oct 14 10:35 hadoop_commands.txt

ubuntu@ubuntu:/mnt/hgfs/Linux_shared_files$ ls hadoop/
Hadoop_1
```

Verification Steps:

1. **Host System Verification:**
   - Open Windows File Explorer
   - Navigate to shared folder location
   - Verify all files and directories are visible
   - Confirm file contents match Ubuntu version
2. **Guest System Verification:**
   - Use `ls` command to list directory contents
   - Use `cat` command to verify file contents
   - Check file permissions with `ls -la`
   - Verify directory structure integrity

Troubleshooting Common Issues:

1. **Shared Folders Not Visible:**

```
# Reinstall VMware Tools
sudo apt-get update
sudo apt-get install open-vm-tools

# Manual mount if needed
sudo mount -t vmhgfs .host:/ /mnt/hgfs
```

2. **Permission Issues:**

```
# Fix permissions
sudo chmod -R 755 /mnt/hgfs/Linux_shared_files
sudo chown -R $USER:$USER /mnt/hgfs/Linux_shared_files
```

3. **Mount Point Issues:**

```
# Create mount point if missing
sudo mkdir -p /mnt/hgfs
```

```
# Add to fstab for persistent mounting
echo '.host:/ /mnt/hgfs vmhgfs defaults,ttl=1 0 0' | sudo tee -a /etc/fstab
```

Security Considerations:

- **Access Control:** Configure appropriate permissions for shared folders
- **Network Security:** Use encrypted protocols (SCP, SFTP) for network transfers
- **File Integrity:** Verify file checksums after transfer operations
- **Backup Strategy:** Maintain copies of critical files before transfer operations

Performance Optimization:

- **Transfer Size:** Use compression for large file transfers
- **Network Bandwidth:** Consider transfer timing for network-based methods
- **Resource Usage:** Monitor CPU and memory during large transfers
- **Batch Operations:** Group multiple file operations for efficiency

Result: Thus the implementation of file transfer from one virtual machine to another virtual machine was executed successfully.

Exercise 11: Hadoop Single Node Cluster Installation

Aim: To install, configure, and run Hadoop and HDFS on a single node cluster.

Algorithm:

1. **System Preparation:**
    - Update Ubuntu system packages
    - Install Java Development Kit (JDK 8)
    - Configure Java environment variables
    - Verify Java installation
2. **User and SSH Configuration:**
    - Create dedicated Hadoop user and group
    - Install and configure OpenSSH server
    - Generate SSH key pairs for passwordless authentication
    - Configure SSH access for localhost
3. **Hadoop Installation:**
    - Download Hadoop distribution package
    - Extract and install Hadoop in system directory
    - Create symbolic links for easy access
    - Set proper file ownership and permissions
4. **Environment Configuration:**
    - Configure Hadoop environment variables
    - Set up JAVA_HOME in Hadoop configuration

- Configure core Hadoop XML files
- Create necessary directory structures

5. **HDFS Setup:**

   - Format the Hadoop Distributed File System
   - Configure namenode and datanode directories
   - Initialize file system metadata

6. **Service Management:**

   - Start Hadoop services (DFS and YARN)
   - Verify service status and functionality
   - Test web interfaces for monitoring

Concept:

- **Hadoop:** Open-source framework for distributed storage and processing of large datasets
- **HDFS (Hadoop Distributed File System):** Distributed file system designed for storing large files
- **Single Node Cluster:** Hadoop installation on one machine for development and testing
- **NameNode:** Master node that manages file system metadata
- **DataNode:** Worker nodes that store actual data blocks
- **YARN (Yet Another Resource Negotiator):** Resource management and job scheduling framework

Detailed Installation Procedure:

Step 1: System Update and Java Installation

```
# Update system packages
ubuntu@ubuntu:~$ sudo apt update
ubuntu@ubuntu:~$ sudo apt upgrade -y

# Install OpenJDK 8
ubuntu@ubuntu:~$ sudo apt install openjdk-8-jdk

# Verify Java installation
ubuntu@ubuntu:~$ java -version
openjdk version "1.8.0_342"
OpenJDK Runtime Environment (build 1.8.0_342-8u342-b07-0ubuntu1~20.04-b07)
OpenJDK 64-Bit Server VM (build 25.342-b07, mixed mode)

ubuntu@ubuntu:~$ javac -version
javac 1.8.0_342
```

Step 2: Java Environment Configuration

```
# Find Java installation path
ubuntu@ubuntu:~$ which javac
/usr/bin/javac
```

```
ubuntu@ubuntu:~$ readlink -f /usr/bin/javac
/usr/lib/jvm/java-8-openjdk-amd64/bin/javac
```

Step 3: Create Hadoop User and Group

```
# Create Hadoop group
ubuntu@ubuntu:~$ sudo addgroup hadoopgroup
Adding group `hadoopgroup' (GID 1001) ...
Done.

# Create Hadoop user
ubuntu@ubuntu:~$ sudo adduser --ingroup hadoopgroup hduser
Adding user `hduser' ...
Adding new user `hduser' (1001) with group `hadoopgroup' ...
Creating home directory `/home/hduser' ...
Copying files from `/etc/skel' ...
New password: cse123
Retype new password: cse123
passwd: password updated successfully
Changing the user information for hduser
Enter the new value, or press ENTER for the default
    Full Name []: Hadoop User
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n] y
```

Step 4: SSH Configuration

```
# Install OpenSSH server and client
ubuntu@ubuntu:~$ sudo apt install openssh-server openssh-client -y

# Switch to hduser account
ubuntu@ubuntu:~$ su - hduser
Password: cse123

# Generate SSH key pair
hduser@ubuntu:~$ ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
Generating public/private rsa key pair.
Your identification has been saved in /home/hduser/.ssh/id_rsa.
Your public key has been saved in /home/hduser/.ssh/id_rsa.pub.

# Add public key to authorized keys
hduser@ubuntu:~$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

# Set proper permissions
hduser@ubuntu:~$ chmod 0600 ~/.ssh/authorized_keys

# Verify SSH directory contents
```

```
hduser@ubuntu:~$ cd .ssh/
hduser@ubuntu:~/.ssh$ ls -l
total 12
-rw------- 1 hduser hadoopgroup 567 Oct 14 10:03 authorized_keys
-rw------- 1 hduser hadoopgroup 2602 Oct 14 10:03 id_rsa
-rw-r--r-- 1 hduser hadoopgroup 567 Oct 14 10:03 id_rsa.pub

# Test passwordless SSH
hduser@ubuntu:~$ ssh localhost
# Should login without password prompt
```

Step 5: Hadoop Installation

```
# Download Hadoop distribution
hduser@ubuntu:~$ wget https://dlcdn.apache.org/hadoop/common/hadoop-3.3.4/hadoop-
3.3.4.tar.gz

# Extract Hadoop archive
hduser@ubuntu:~$ tar xzf hadoop-3.3.4.tar.gz

# Switch to admin user to move files
hduser@ubuntu:~$ su - ubuntu
Password: [ubuntu_password]

# Move Hadoop to system directory
ubuntu@ubuntu:~$ sudo mv /home/hduser/hadoop-3.3.4 /usr/local/

# Create symbolic link
ubuntu@ubuntu:~$ sudo ln -sf /usr/local/hadoop-3.3.4/ /usr/local/hadoop

# Change ownership to hduser
ubuntu@ubuntu:~$ sudo chown -R hduser:hadoopgroup /usr/local/hadoop*
```

Step 6: Environment Variable Configuration

```
# Switch back to hduser
ubuntu@ubuntu:~$ su - hduser
Password: cse123

# Edit .bashrc file
hduser@ubuntu:~$ nano ~/.bashrc

# Add the following lines at the end:
# Hadoop configuration
export HADOOP_PREFIX=/usr/local/hadoop
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_MAPRED_HOME=${HADOOP_HOME}
export HADOOP_COMMON_HOME=${HADOOP_HOME}
export HADOOP_HDFS_HOME=${HADOOP_HOME}
export YARN_HOME=${HADOOP_HOME}
```

```
export HADOOP_CONF_DIR=${HADOOP_HOME}/etc/hadoop

# Native path
export HADOOP_COMMON_LIB_NATIVE_DIR=${HADOOP_PREFIX}/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_PREFIX/lib/native"

# Java path
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64

# OS path
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin

# Apply changes
hduser@ubuntu:~$ source ~/.bashrc
```

Step 7: Hadoop Environment Configuration

```
# Navigate to Hadoop configuration directory
hduser@ubuntu:~$ cd /usr/local/hadoop/etc/hadoop/

# Edit hadoop-env.sh
hduser@ubuntu:/usr/local/hadoop/etc/hadoop$ nano hadoop-env.sh

# Add Java home at the end:
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
```

Step 8: Core Hadoop Configuration Files

Configure core-site.xml:

```
hduser@ubuntu:/usr/local/hadoop/etc/hadoop$ nano core-site.xml
```

```
<configuration>
    <property>
        <name>fs.default.name</name>
        <value>hdfs://localhost:9000</value>
    </property>
</configuration>
```

Configure hdfs-site.xml:

```
hduser@ubuntu:/usr/local/hadoop/etc/hadoop$ nano hdfs-site.xml
```

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
    <property>
        <name>dfs.name.dir</name>
        <value>file:/usr/local/hadoop/hadoopdata/hdfs/namenode</value>
    </property>
    <property>
        <name>dfs.data.dir</name>
        <value>file:/usr/local/hadoop/hadoopdata/hdfs/datanode</value>
    </property>
</configuration>
```

Configure mapred-site.xml:

```
hduser@ubuntu:/usr/local/hadoop/etc/hadoop$ nano mapred-site.xml
```

```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
</configuration>
```

Configure yarn-site.xml:

```
hduser@ubuntu:/usr/local/hadoop/etc/hadoop$ nano yarn-site.xml
```

```
<configuration>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
    <property>
        <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
        <value>org.apache.hadoop.mapred.ShuffleHandler</value>
    </property>
    <property>
        <name>yarn.resourcemanager.hostname</name>
        <value>127.0.0.1</value>
    </property>
    <property>
        <name>yarn.acl.enable</name>
```

```
        <value>0</value>
    </property>
    <property>
        <name>yarn.nodemanager.env-whitelist</name>

<value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,HADOOP_CONF_DIR,CLASSPATH_PRE
PEND_DISTCACHE,HADOOP_YARN_HOME,HADOOP_MAPRED_HOME</value>
    </property>
</configuration>
```

Step 9: Create HDFS Directory Structure

```
# Create Hadoop data directories
hduser@ubuntu:~$ cd /usr/local/hadoop
hduser@ubuntu:/usr/local/hadoop$ mkdir -p hadoopdata/hdfs/namenode
hduser@ubuntu:/usr/local/hadoop$ mkdir -p hadoopdata/hdfs/datanode

# Verify directory structure
hduser@ubuntu:/usr/local/hadoop$ ls -la hadoopdata/hdfs/
total 16
drwxrwxr-x 4 hduser hadoopgroup 4096 Oct 14 11:30 .
drwxrwxr-x 3 hduser hadoopgroup 4096 Oct 14 11:30 ..
drwxrwxr-x 2 hduser hadoopgroup 4096 Oct 14 11:30 datanode
drwxrwxr-x 2 hduser hadoopgroup 4096 Oct 14 11:30 namenode
```

Step 10: Format HDFS Namenode

```
# Format the namenode
hduser@ubuntu:/usr/local/hadoop$ hdfs namenode -format
WARNING: /usr/local/hadoop/logs does not exist. Creating.
2023-10-14 11:35:03,594 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG: host = ubuntu/127.0.1.1
STARTUP_MSG: args = [-format]
STARTUP_MSG: version = 3.3.4
...
2023-10-14 11:35:04,967 INFO common.Storage: Storage directory
/usr/local/hadoop/hadoopdata/hdfs/namenode has been successfully formatted.
...
2023-10-14 11:35:05,172 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at ubuntu/127.0.1.1
************************************************************/
```

Step 11: Start Hadoop Services

```
# Start HDFS services
hduser@ubuntu:~$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [ubuntu]

# Verify HDFS services
hduser@ubuntu:~$ jps
57464 NameNode
57610 DataNode
57824 SecondaryNameNode
57982 Jps

# Start YARN services
hduser@ubuntu:~$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers

# Verify all services
hduser@ubuntu:~$ jps
57464 NameNode
57610 DataNode
57824 SecondaryNameNode
58082 ResourceManager
58221 NodeManager
58374 Jps
```

Alternative: Start All Services

```
# Start all Hadoop services at once
hduser@ubuntu:~$ start-all.sh

# Verify Hadoop version
hduser@ubuntu:~$ hadoop version
Hadoop 3.3.4
Source code repository https://github.com/apache/hadoop.git -r
a585763716a3345d3e6412996f18013e5d4f8a1c
Compiled by stevel on 2022-07-29T12:32Z
Compiled with protoc 3.7.1
From source with checksum fb9dd8918a7d8a7dd4da4c546ee25449
This command was run using /usr/local/hadoop/share/hadoop/common/hadoop-common-
3.3.4.jar
```

Step 12: Verify Installation

Web Interface Access:

- **NameNode Web UI:** http://localhost:9870
- **DataNode Web UI:** http://localhost:9864
- **ResourceManager Web UI:** http://localhost:8088

Command Line Verification:

```
# Check HDFS status
hduser@ubuntu:~$ hdfs dfsadmin -report
Configured Capacity: 123456789012 (115.02 GB)
Present Capacity: 120987654321 (112.72 GB)
DFS Remaining: 120987654321 (112.72 GB)
DFS Used: 24576 (24 KB)
DFS Used%: 0.00%
Under replicated blocks: 0
Blocks with corrupt replicas: 0
Missing blocks: 0
Missing blocks (with replication factor 1): 0

# Test HDFS operations
hduser@ubuntu:~$ hdfs dfs -mkdir /test
hduser@ubuntu:~$ hdfs dfs -ls /
Found 1 items
drwxr-xr-x   - hduser supergroup          0 2023-10-14 11:45 /test

# Create a test file
hduser@ubuntu:~$ echo "Hello Hadoop!" > test.txt
hduser@ubuntu:~$ hdfs dfs -put test.txt /test/
hduser@ubuntu:~$ hdfs dfs -cat /test/test.txt
Hello Hadoop!
```

Step 13: Stop Services

```
# Stop YARN services
hduser@ubuntu:~$ stop-yarn.sh
Stopping nodemanagers
Stopping resourcemanager

# Stop HDFS services
hduser@ubuntu:~$ stop-dfs.sh
Stopping namenodes on [localhost]
Stopping datanodes
Stopping secondary namenodes [ubuntu]

# Alternative: Stop all services
hduser@ubuntu:~$ stop-all.sh
```

Troubleshooting Common Issues:

1. **Permission Denied Errors:**

   ```
   sudo chown -R hduser:hadoopgroup /usr/local/hadoop*
   sudo chmod -R 755 /usr/local/hadoop
   ```

2. **Java Home Issues:**

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

3. **SSH Connection Issues:**

```
ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 600 ~/.ssh/authorized_keys
```

4. **Port Already in Use:**

```
# Check running processes
sudo netstat -tulpn | grep :9000
# Kill conflicting processes if needed
```

Result: Thus the installation and configuration of Hadoop and HDFS single node cluster was executed successfully.

Exercise 12: Word Count using MapReduce

Aim: To implement a Word Count program using MapReduce to demonstrate Map and Reduce tasks.

Algorithm:

1. **Input Analysis:**
    - Read input text file containing words and sentences
    - Parse text into individual tokens (words)
    - Prepare data for distributed processing
2. **Map Phase:**
    - Split input text into words using StringTokenizer
    - Emit each word as key with count of 1 as value
    - Create (word, 1) pairs for all words
3. **Shuffle and Sort:**
    - Group all values by key (word)
    - Sort keys alphabetically
    - Prepare data for reduce phase
4. **Reduce Phase:**
    - Sum all values for each unique key (word)
    - Output final count for each word

- Write results to output file

5. **Execution:**
    - Package code as JAR file
    - Submit job to Hadoop cluster
    - Monitor execution and collect results

Concept:

- **MapReduce:** Programming model for processing large datasets in distributed computing
- **Mapper:** Processes input and produces intermediate key-value pairs
- **Reducer:** Processes intermediate data and produces final output
- **Combiner:** Local reducer that optimizes data transfer between map and reduce
- **HDFS Integration:** Reads from and writes to Hadoop Distributed File System
- **Job Configuration:** Defines input/output paths and processing classes

Java Implementation:

```
package WordCount;

import java.io.*;
import java.util.*;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.*;
import org.apache.hadoop.mapreduce.lib.output.*;

public class WordCount {

    // Mapper class
    public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable> {

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(Object key, Text value, Context context)
            throws IOException, InterruptedException {

            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken().toLowerCase());
                context.write(word, one);
            }
        }
    }

    // Reducer class
    public static class IntSumReducer
        extends Reducer<Text, IntWritable, Text, IntWritable> {
```

```
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values,
            Context context) throws IOException, InterruptedException {

            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }

    // Driver method
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");

        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Detailed Procedure:

Step 1: Eclipse Project Setup

1. Open Eclipse IDE
2. Create new Java Project: File → New → Java Project
3. Project name: `WordCount`
4. Select JavaSE-1.8 and click Finish
5. Right-click on project → Properties → Libraries → Add External JARs
6. Add the following JAR files from `/usr/local/hadoop/share/hadoop/`:
    - `common/hadoop-common-3.3.4.jar`
    - `common/lib/common-cli-1.2.jar`
    - `mapreduce/hadoop-mapreduce-client-core-3.3.4.jar`

Step 2: Package and Class Creation

```
# Create package structure
Right-click on src → New → Package
```

```
Package name: WordCount

# Create Java class
Right-click on package → New → Class
Class name: WordCount.java
```

Step 3: JAR File Creation

1. Right-click on WordCount project → Export
2. Select JAR file under Java
3. Choose export destination (e.g., `/home/hduser/wordcount.jar`)
4. Click Finish

Step 4: Input File Preparation

Create `input.txt` file:

```
hi how are you
welcome to velammal engineering college
we are from computer science department
today we are gonna learn about mapreduce concept
then we will apply the mapreduce programming knowledge to process the text file
the output printed should be the count of occurrences of each word in the text
file
here ends our today program so bye
```

Step 5: HDFS Operations

```
# Start Hadoop services
hduser@ubuntu:~$ start-dfs.sh
hduser@ubuntu:~$ start-yarn.sh

# Verify services
hduser@ubuntu:~$ jps
11147 NameNode
11300 DataNode
11485 SecondaryNameNode
11638 ResourceManager
11949 NodeManager
12138 Jps

# Create HDFS directories
hduser@ubuntu:~$ hdfs dfs -mkdir /WordCount
hduser@ubuntu:~$ hdfs dfs -mkdir /WordCount/Input

# Upload input file to HDFS
hduser@ubuntu:~$ hdfs dfs -put input.txt /WordCount/Input/

# Verify file upload
hduser@ubuntu:~$ hdfs dfs -ls /WordCount/Input/
```

```
Found 1 items
-rw-r--r--   1 hduser supergroup        344 2023-10-14 15:30
/WordCount/Input/input.txt
```

Step 6: Execute MapReduce Job

```
# Run the WordCount program
hduser@ubuntu:~$ hadoop jar wordcount.jar WordCount.WordCount
/WordCount/Input/input.txt /WordCount/Output

# Job execution output (simplified)
INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
INFO input.FileInputFormat: Total input paths to process : 1
INFO mapreduce.JobSubmitter: number of splits:1
INFO mapreduce.Job: Running job: job_1697280324_0001
INFO mapreduce.Job: map 0% reduce 0%
INFO mapreduce.Job: map 100% reduce 0%
INFO mapreduce.Job: map 100% reduce 100%
INFO mapreduce.Job: Job job_1697280324_0001 completed successfully

Job Counters:
    Map input records=8
    Map output records=59
    Reduce input records=48
    Reduce output records=48

File System Counters:
    HDFS: Number of bytes read=456
    HDFS: Number of bytes written=390
```

Step 7: View Results

```
# Check output directory
hduser@ubuntu:~$ hdfs dfs -ls /WordCount/Output/
Found 2 items
-rw-r--r--   1 hduser supergroup          0 2023-10-14 15:35
/WordCount/Output/_SUCCESS
-rw-r--r--   1 hduser supergroup        390 2023-10-14 15:35
/WordCount/Output/part-r-00000

# View word count results
hduser@ubuntu:~$ hdfs dfs -cat /WordCount/Output/part-r-00000
```

Sample Output:

```
about       1
apply       1
are         3
```

```
bye         1
college     1
computer    1
concept     1
count       1
department  1
each        2
engineering 1
file        2
from        1
gonna       1
here        1
hi          1
how         1
knowledge   1
learn       1
mapreduce   2
occurrences 1
of          2
our         1
output      1
printed     1
process     1
program     1
programming 1
science     1
should      1
so          1
text        2
the         4
to          2
today       2
velammal    1
we          2
welcome     1
will        1
word        1
you         1
```

MapReduce Workflow Explanation:

1. **Input Splitting:** Hadoop splits input file into chunks for parallel processing
2. **Map Phase:** Each mapper processes its chunk and emits (word, 1) pairs
3. **Shuffle:** Framework groups all values by key (word)
4. **Reduce Phase:** Reducer sums up counts for each unique word
5. **Output:** Final results written to HDFS output directory

Key Components:

- **TokenizerMapper:** Splits text into words and emits (word, 1) pairs
- **IntSumReducer:** Aggregates counts for each word
- **Job Configuration:** Sets up input/output paths and processing classes
- **Combiner:** Uses same reducer logic to optimize network traffic

Troubleshooting Common Issues:

1. **ClassNotFoundException:** Ensure all Hadoop JARs are in classpath
2. **Permission Denied:** Check HDFS directory permissions
3. **Job Failure:** Verify input file exists and output directory doesn't exist
4. **Memory Issues:** Adjust JVM heap size in mapred-site.xml

Alternative Input/Output Formats:

```
// For processing multiple small files
job.setInputFormatClass(CombineTextInputFormat.class);

// For custom output format
job.setOutputFormatClass(TextOutputFormat.class);
```

Result: Thus the Word Count program using MapReduce to demonstrate Map and Reduce tasks was implemented successfully.

Last updated: October 14, 2025