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:

```java
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()) {
```

```java
            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);
```

```java
            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:

```java
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();
```

```java
        }

        // 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) {
```

```java
            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:

```java
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
 */
```

```java
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
```

```java
            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
```

```java
    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.

Last updated: July 27, 2025