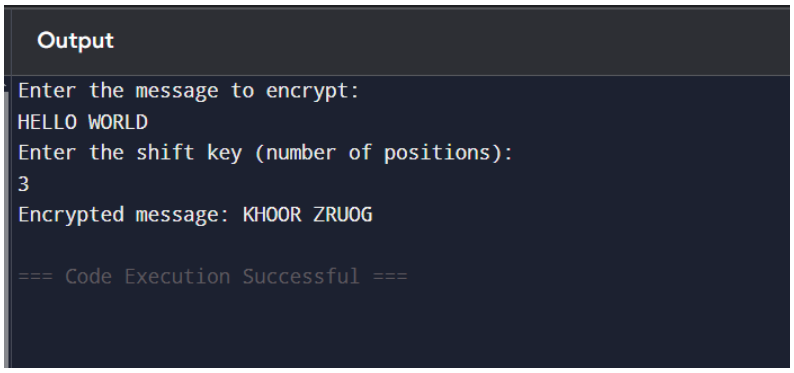
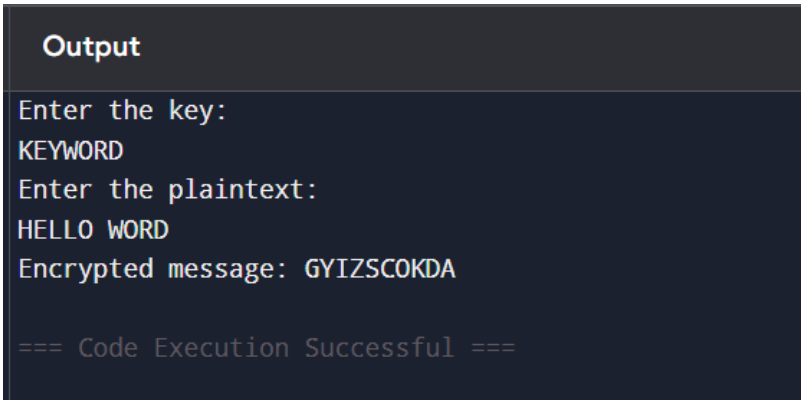
**OUTPUT**

1. **Ceaser cipher**




2. **Playfair cipher**



**INPUT**

1. **Ceaser cipher**

```
import java.util.Scanner;
class Main {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter the message to encrypt:");
    String plaintext = scanner.nextLine();
    System.out.println("Enter the shift key (number of positions):");
    int shift = scanner.nextInt();
    scanner.nextLine();
    System.out.println("Encrypted message: " + encrypt(plaintext, shift));
  }
  public static String encrypt(String text, int shift) {
    StringBuilder result = new StringBuilder();
    for (char c : text.toCharArray()) {
      if (Character.isUpperCase(c)) result.append((char) ((c - 'A' + shift) % 26 + 'A'));
      else if (Character.isLowerCase(c)) result.append((char) ((c - 'a' + shift) % 26 +
'a'));
      else result.append(c);
    }
    return result.toString();
  }
}
```

2. **Playfair cipher**

```
import java.util.*;
class Main {
  private static char[][] matrix = new char[5][5];
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter the key:");
    String key = scanner.nextLine().toUpperCase().replaceAll("[^A-Z]", "").replace("J",
"I");
    System.out.println("Enter the plaintext:");
    String plaintext = scanner.nextLine().toUpperCase().replaceAll("[^A-Z]",
"").replace("J", "I");
    generateMatrix(key);
    System.out.println("Encrypted message: " + encrypt(plaintext));
```

```
    }

    private static void generateMatrix(String key) {
        StringBuilder k = new StringBuilder(key);
        for (char c = 'A'; c <= 'Z'; c++) if (c != 'J' && k.indexOf(String.valueOf(c)) == -1)
k.append(c);
        int idx = 0;
        for (int i = 0; i < 5; i++) for (int j = 0; j < 5; j++) matrix[i][j] = k.charAt(idx++);
    }
    private static String encrypt(String text) {
        StringBuilder p = new StringBuilder(), r = new StringBuilder();
        for (int i = 0; i < text.length(); i++) {
            p.append(text.charAt(i));
            if (i + 1 < text.length() && text.charAt(i) == text.charAt(i + 1)) p.append('X');
        }
        if (p.length() % 2 != 0) p.append('X');
        for (int i = 0; i < p.length(); i += 2) {
            char a = p.charAt(i), b = p.charAt(i + 1);
            int[] p1 = findPos(a), p2 = findPos(b);
            if (p1[0] == p2[0]) {
                r.append(matrix[p1[0]][(p1[1] + 1) % 5]).append(matrix[p2[0]][(p2[1] + 1) %
5]);
            } else if (p1[1] == p2[1]) {
                r.append(matrix[(p1[0] + 1) % 5][p1[1]]).append(matrix[(p2[0] + 1) %
5][p2[1]]);
            } else {
                r.append(matrix[p1[0]][p2[1]]).append(matrix[p2[0]][p1[1]]);
            }
        }
        return r.toString();
    }
    private static int[] findPos(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 null;
    }
}
```

**RESULT**

**OUTPUT**

1. **Rail fence**

```
Output
Encrypted Message:
HOLELWRDLO

Decrypted Message:
HELLOWORLD

=== Code Execution Successful ===
```

2. **Row & Column Transformation**

```
Output
Enter the plain text:
HELLO
Enter the rows:
2
Enter the columns:
3
Encrypted Message: LEHXOL
```

1. **Rail fence**

```
import java.util.Arrays;
class Main {
   public static String encryptRailFence(String text, int key) {
      char[][] rail = new char[key][text.length()];
      for (int i = 0; i < key; i++) Arrays.fill(rail[i], '\n');
      boolean dirDown = false;
      int row = 0, col = 0;
      for (int i = 0; i < text.length(); i++) {
         if (row == 0 || row == key - 1) dirDown = !dirDown;
         rail[row][col++] = text.charAt(i);
         row = dirDown ? row + 1 : row - 1;
      }
      StringBuilder result = new StringBuilder();
      for (int i = 0; i < key; i++)
         for (int j = 0; j < text.length(); j++)
            if (rail[i][j] != '\n') result.append(rail[i][j]);
      return result.toString();
   }
   public static String decryptRailFence(String cipher, int key) {
      char[][] rail = new char[key][cipher.length()];
      for (int i = 0; i < key; i++) Arrays.fill(rail[i], '\n');
      boolean dirDown = true;
      int row = 0, col = 0;
      for (int i = 0; i < cipher.length(); i++) {
         if (row == 0) dirDown = true;
         if (row == key - 1) dirDown = false;
         rail[row][col++] = '*';
         row = dirDown ? row + 1 : row - 1;
```

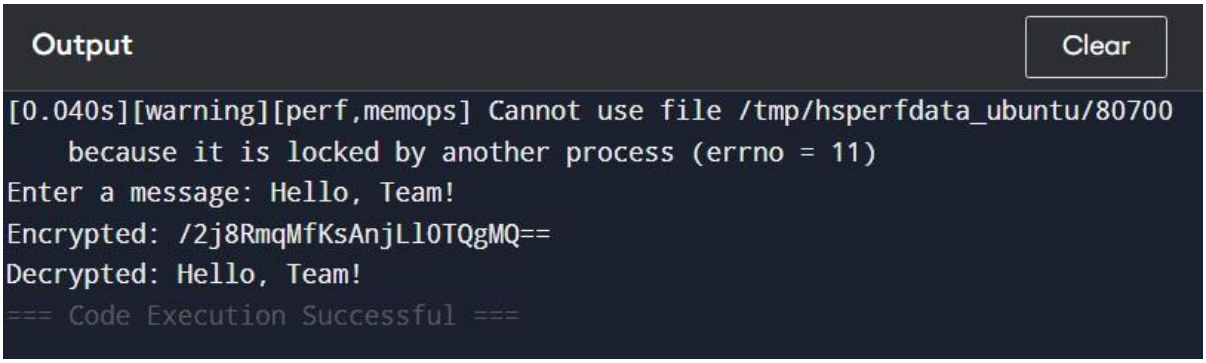
```
        }
        int index = 0;
        for (int i = 0; i < key; i++)
            for (int j = 0; j < cipher.length(); j++)
                if (rail[i][j] == '*' && index < cipher.length())
                    rail[i][j] = cipher.charAt(index++);
        StringBuilder result = new StringBuilder();
        row = 0; col = 0;
        for (int i = 0; i < cipher.length(); i++) {
            if (row == 0) dirDown = true;
            if (row == key - 1) dirDown = false;
            if (rail[row][col] != '*') result.append(rail[row][col++]);
            row = dirDown ? row + 1 : row - 1;
        }
        return result.toString();
    }
    public static void main(String[] args) {
        System.out.println("Encrypted Message: ");
        System.out.println(encryptRailFence("HELLOWORLD", 3));
        System.out.println("\nDecrypted Message: ");
        System.out.println(decryptRailFence(encryptRailFence("HELLOWORLD", 3), 3));
    }
}
```

2. **Row & Column Transformation**

```
import java.util.Scanner;
class Main {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.println("Enter the plain text:");
      String plaintext = scanner.nextLine().toUpperCase().replaceAll("[^A-Z]", "");
      System.out.println("Enter the rows:");
      int rows = scanner.nextInt();
      System.out.println("Enter the columns:");
      int cols = scanner.nextInt();
      System.out.println(encrypt(plaintext, rows, cols));
   }
   public static String encrypt(String text, int rows, int cols) {
      char[][] matrix = new char[rows][cols];
      int index = 0;
      for (int i = 0; i < rows; i++)
         for (int j = 0; j < cols; j++)
            matrix[i][j] = index < text.length() ? text.charAt(index++) : 'X';
      StringBuilder ciphertext = new StringBuilder();
      for (int i = 0; i < rows; i++)
         for (int j = cols - 1; j >= 0; j--)
            ciphertext.append(matrix[i][j]);
      return ciphertext.toString();
   }}
```

**RESULT**

**OUTPUT**

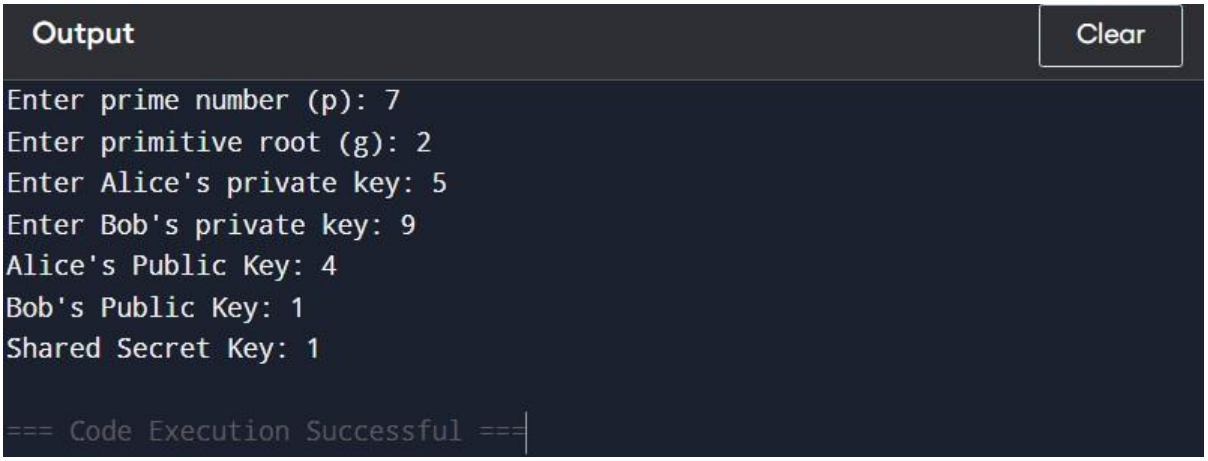

Output
Clear
[0.040s][warning][perf,memops] Cannot use file /tmp/hsperfdata_ubuntu/80700
because it is locked by another process (errno = 11)
Enter a message: Hello, Team!
Encrypted: /2j8RmqMfKsAnjLl0TQgMQ==
Decrypted: Hello, Team!
=== Code Execution Successful ===

**INPUT**

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import java.util.Base64;
import java.util.Scanner;
class Main {
   public static void main(String[] args) throws Exception {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter a message: ");
      String message = sc.nextLine();
      SecretKey key = KeyGenerator.getInstance("DES").generateKey();
      Cipher cipher = Cipher.getInstance("DES");
      cipher.init(Cipher.ENCRYPT_MODE, key);
      String encrypted =
Base64.getEncoder().encodeToString(cipher.doFinal(message.getBytes()));
      System.out.println("Encrypted: " + encrypted);
      cipher.init(Cipher.DECRYPT_MODE,  key);
      String decrypted = new String(cipher.doFinal(Base64.getDecoder().decode(encrypted)));
      System.out.println("Decrypted: " + decrypted);
   }
}
```

**RESULT**

**OUTPUT**

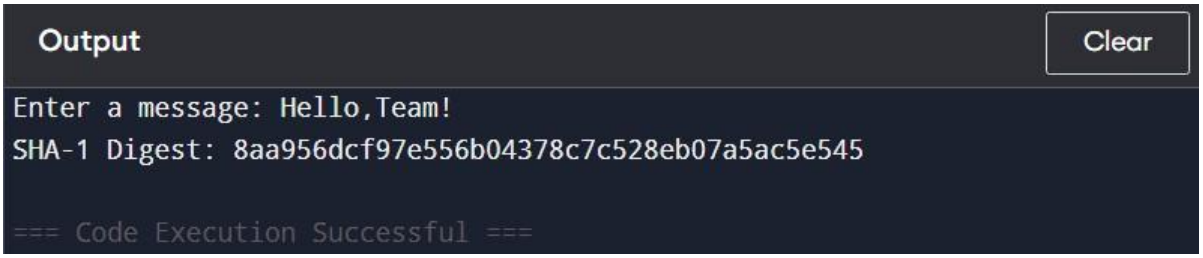

Output
Clear
Enter prime number (p): 7
Enter primitive root (g): 2
Enter Alice's private key: 5
Enter Bob's private key: 9
Alice's Public Key: 4
Bob's Public Key: 1
Shared Secret Key: 1
=== Code Execution Successful ===

**INPUT**

```
import java.util.Scanner;
class Main {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter prime number (p): ");
      int p = sc.nextInt();
      System.out.print("Enter primitive root (g): ");
      int g = sc.nextInt();
      System.out.print("Enter Alice's private key: ");
      int a = sc.nextInt();
      System.out.print("Enter Bob's private key: ");
      int b = sc.nextInt();
      int A = (int) Math.pow(g, a) % p; // Alice's public key
      int B = (int) Math.pow(g, b) % p; // Bob's public key
      int sharedKey = (int) Math.pow(B, a) % p; // Shared key calculated by Alice (or Bob)
      System.out.println("Alice's Public Key: " + A);
      System.out.println("Bob's Public Key: " + B);
      System.out.println("Shared Secret Key: " + sharedKey);
   }
}
```

**RESULT**

**OUTPUT**

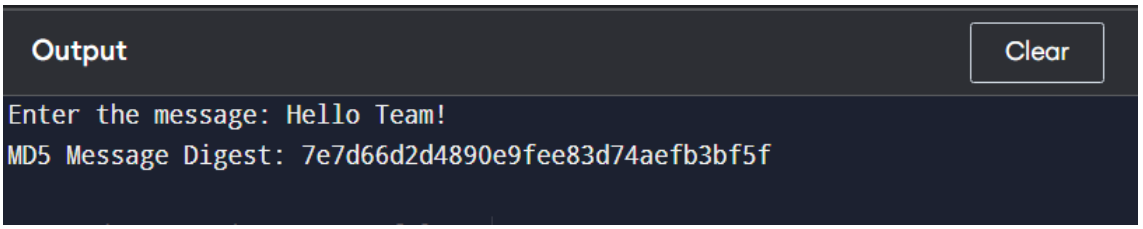

Output
Clear
Enter a message: Hello,Team!
SHA-1 Digest: 8aa956dcf97e556b04378c7c528eb07a5ac5e545
=== Code Execution Successful ===

**INPUT**

```
import java.security.MessageDigest;
import java.util.Scanner;
class Main {
   public static void main(String[] args) throws Exception {
      Scanner sc = new Scanner(System.in);

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

      byte[] digest = MessageDigest.getInstance("SHA-1").digest(message.getBytes());
      StringBuilder hexDigest = new StringBuilder();

      for (byte b : digest) hexDigest.append(String.format("%02x", b));

      System.out.println("SHA-1 Digest: " + hexDigest);
   }
}
```

**RESULT**

**OUTPUT**


Output
Clear
Enter the message: Hello Team!
MD5 Message Digest: 7e7d66d2d4890e9fee83d74aefb3bf5f

**INPUT**

```
import java.security.MessageDigest;
import java.util.Scanner;

 class Main {
   public static void main(String[] args) throws Exception {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter the message: ");
      String message = scanner.nextLine();

      MessageDigest md = MessageDigest.getInstance("MD5");

      byte[] digest = md.digest(message.getBytes());

      StringBuilder hexString = new StringBuilder();
      for (byte b : digest) {
         hexString.append(String.format("%02x",  b));
      }
      System.out.println("MD5 Message Digest: " + hexString.toString());
   }
}
```

**RESULT**

**OUTPUT**


Output
Clear
Enter the message: Hello team!
Digital Signature: MC0CFBpDq
/hWlpbGui3mXN3RZ42laWGxAhUAiglgVXF4NfyB8LCf7ygQ3cmytWw=
Signature verification result: true

**INPUT**

```
import java.security.*;
import java.util.Base64;
import java.util.Scanner;
class Main {
   public static void main(String[] args) throws Exception {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter the message: ");
      String message = scanner.nextLine();
      KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("DSA");
      keyPairGenerator.initialize(1024);
      KeyPair keyPair = keyPairGenerator.generateKeyPair();
      Signature signature = Signature.getInstance("SHA256withDSA");
      signature.initSign(keyPair.getPrivate());
      signature.update(message.getBytes());
      byte[] signedMessage = signature.sign();
      String signatureBase64 = Base64.getEncoder().encodeToString(signedMessage);
      System.out.println("Digital Signature: " + signatureBase64);
      Signature verifier = Signature.getInstance("SHA256withDSA");
      verifier.initVerify(keyPair.getPublic());
      verifier.update(message.getBytes());
      boolean isVerified = verifier.verify(signedMessage);
      System.out.println("Signature verification result: " + isVerified);
   }
}
```

**RESULT:**