1. Implement GCD Calculator program using Eucledian Algorithm

```
import java.util.Scanner;
public class GCDUsingEuclidean
 {
   public static int gcd(int a, int b)
   {
      int r1 = a;
      int r2 = b;
      int q, r;
      while (r2 > 0)
      {
         q = r1 / r2;
         r = r1 - q * r2;
         r1 = r2;
         r2 = r;
      }
      return r1; // GCD is stored in r1
   }
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter first number: ");
      int num1 = scanner.nextInt();
      System.out.print("Enter second number: ");
      int num2 = scanner.nextInt();
      int result = gcd(num1, num2);
      System.out.println("GCD of " + num1 + " and " + num2 + " is: " + result);
      scanner.close();
   }
}
```

2. Implement a Java program to find the values of s and t for the given linear equation ax+by=gcd(a,b) using Extended Eucledian Algorithm.

```
import java.util.Scanner;
public class ExtendedEuclidean
 {
    public static int extendedGCD(int a, int b, int[] coefficients)
    {
      int r1 = a, r2 = b;
      int s1 = 1, s2 = 0, t1 = 0, t2 = 1;
      int q, r, s, t;
      while (r2 > 0)
     {
         q = r1 / r2;
         r = r1 - q * r2;
         r1 = r2;
         r2 = r;
         s = s1 - q * s2;
         s1 = s2;
         s2 = s;
         t = t1 - q * t2;
         t1 = t2;
         t2 = t;
      }
      coefficients[0] = s1;
      coefficients[1] = t1;
      return r1;
   }
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter value for a: ");
      int a = scanner.nextInt();
      System.out.print("Enter value for b: ");
      int b = scanner.nextInt();
      int[] coefficients = new int[2]; // To store s and t
      int gcd = extendedGCD(a, b, coefficients);
      System.out.println("GCD(" + a + ", " + b + ") = " + gcd);
      System.out.println("Values of s and t:");
      System.out.println("s = " + coefficients[0]);
      System.out.println("t = " + coefficients[1]);
      scanner.close();
   }
}
```

3. Write a program to find the multiplicative inverse of a given number using Extended Eucledian Algorithm

```
import java.util.Scanner;
public class MultiplicativeInverse
 {
 public static int findMultiplicativeInverse(int b, int n)
 {
       int r1 = n, r2 = b;
       int t1 = 0, t2 = 1;
       int q, r, t;
       while (r2 > 0)
        {
          q = r1 / r2;
          r = r1 - q * r2;
          r1 = r2;
          r2 = r;
          t = t1 - q * t2;
          t1 = t2;
          t2 = t;
       }
       if (r1 != 1)
       {
          System.out.println("Multiplicative inverse does not exist.");
          return -1;
       }
       if (t1 < 0)
       {
          t1 += n;
       }
       return t1;
   }
   public static void main(String[] args)
   {
       Scanner scanner = new Scanner(System.in);
       System.out.print("Enter value for b: ");
       int b = scanner.nextInt();
       System.out.print("Enter value for n (modulus): ");
       int n = scanner.nextInt();
       int inverse = findMultiplicativeInverse(b, n);
       if (inverse != -1)
       {
          System.out.println("Multiplicative Inverse of " + b + " modulo " + n + " is: " +
inverse);
       }
       scanner.close();
   }
}
```

4. write a java program to solve a linear diophantine equation ax+by=c to find a particular and general solutions using Extended Eucledian algorithm

```
import java.util.Scanner;
public class LinearDiophantineSolver
 {
 public static int extendedGCD(int a, int b, int[] coefficients)
 {
      int r1 = a, r2 = b;
      int s1 = 1, s2 = 0, t1 = 0, t2 = 1;
      int q, r, s, t;
      while (r2 > 0)
      {
         q = r1 / r2;
         r = r1 - q * r2;
         r1 = r2;
         r2 = r;
         s = s1 - q * s2;
         s1 = s2;
         s2 = s;
         t = t1 - q * t2;
         t1 = t2;
         t2 = t;
      }
      coefficients[0] = s1;
      coefficients[1] = t1;
      return r1;
   }
   public static void solveDiophantine(int a, int b, int c)
 {
      int[] coefficients = new int[2]; // To store x0 and y0
      int gcd = extendedGCD(a, b, coefficients);
      if (c % gcd != 0)
     {
         System.out.println("No integer solution exists.");
         return;
      }
      int x0 = coefficients[0] * (c / gcd);
      int y0 = coefficients[1] * (c / gcd);
      System.out.println("Particular Solution: x0 = " + x0 + ", y0 = " + y0);
      System.out.println("General Solution: x = " + x0 + " + (" + (b / gcd) + ")t");
      System.out.println("                  y = " + y0 + " - (" + (a / gcd) + ")t");
      System.out.println("(where t is any integer)");
   }
   public static void main(String[] args)
    {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter coefficient a: ");
      int a = scanner.nextInt();
      System.out.print("Enter coefficient b: ");
      int b = scanner.nextInt();
      System.out.print("Enter constant c: ");
      int c = scanner.nextInt();
      solveDiophantine(a, b, c);
      scanner.close();
   }}
```

5. Write a Java program to solve a single variable linear congruent equation ax= b mod n

```
import java.util.Scanner;
public class LinearCongruenceSolver
 {
   public static int extendedGCD(int a, int b, int[] coefficients)
 {
      int r1 = a, r2 = b;
      int s1 = 1, s2 = 0, t1 = 0, t2 = 1;
      int q, r, s, t;
      while (r2 > 0) {
       q = r1 / r2;
         r = r1 - q * r2;
         r1 = r2;
         r2 = r;
         s = s1 - q * s2;
         s1 = s2;
         s2 = s;
         t = t1 - q * t2;
         t1 = t2;
         t2 = t;
      }
      coefficients[0] = s1;
      coefficients[1] = t1;
      return r1;
   }
   public static void solveLinearCongruence(int a, int b, int n)
   {
      int[] coefficients = new int[2];
      int gcd = extendedGCD(a, n, coefficients);
      if (b % gcd != 0)
      {
         System.out.println("No solution exists.");
         return;
      }
      int x0 = (coefficients[0] * (b / gcd)) % n;
      if (x0 < 0)
      {
         x0 += n;
      }
      System.out.println("Particular Solution: x0 = " + x0);
    System.out.println("General Solution: x = " + x0 + " + " + (n / gcd) + "t (mod " + n + ")");
      System.out.println("(where t is any integer)");
   }
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter coefficient a: ");
      int a = scanner.nextInt();
      System.out.print("Enter constant b: ");
      int b = scanner.nextInt();
      System.out.print("Enter modulus n: ");
      int n = scanner.nextInt();
      solveLinearCongruence(a, b, n);
      scanner.close();
   } }
```

6. Write a Java program to solve a set of multi variable linear congruent equations

```
import java.util.Scanner;
public class ModularInverseSolver
{
   public static int modInverse(int a, int m)
 {
      int[] coefficients = new int[2];
      int gcd = extendedGCD(a, m, coefficients);
      if (gcd != 1)
      {
         throw new ArithmeticException("Modular inverse does not exist");
      }
      return (coefficients[0] % m + m) % m;
   }
   public static int extendedGCD(int a, int b, int[] coefficients)
 {
      int r1 = a, r2 = b, s1 = 1, s2 = 0, t1 = 0, t2 = 1;
      int q, r, s, t;
      while (r2 > 0) {
         q = r1 / r2;
         r = r1 - q * r2;
         r1 = r2;
         r2 = r;
         s = s1 - q * s2;
         s1 = s2;
         s2 = s;
         t = t1 - q * t2;
         t1 = t2;
         t2 = t;
      }
      coefficients[0] = s1;
      coefficients[1] = t1;
      return r1;
   }
   public static int determinantModulo(int[][] matrix, int mod)
 {
      int det = (matrix[0][0] * (matrix[1][1] * matrix[2][2] - matrix[1][2] * matrix[2][1])
            - matrix[0][1] * (matrix[1][0] * matrix[2][2] - matrix[1][2] * matrix[2][0])
            + matrix[0][2] * (matrix[1][0] * matrix[2][1] - matrix[1][1] * matrix[2][0])) %
mod;
      return (det + mod) % mod;
   }
   public static int[][] adjugateMatrix(int[][] matrix, int mod)
{
      int[][] adj = new int[3][3];
      adj[0][0] = (matrix[1][1] * matrix[2][2] - matrix[1][2] * matrix[2][1]) % mod;
      adj[0][1] = (matrix[0][2] * matrix[2][1] - matrix[0][1] * matrix[2][2]) % mod;
      adj[0][2] = (matrix[0][1] * matrix[1][2] - matrix[0][2] * matrix[1][1]) % mod;
      adj[1][0] = (matrix[1][2] * matrix[2][0] - matrix[1][0] * matrix[2][2]) % mod;
      adj[1][1] = (matrix[0][0] * matrix[2][2] - matrix[0][2] * matrix[2][0]) % mod;
      adj[1][2] = (matrix[0][2] * matrix[1][0] - matrix[0][0] * matrix[1][2]) % mod;
      adj[2][0] = (matrix[1][0] * matrix[2][1] - matrix[1][1] * matrix[2][0]) % mod;
      adj[2][1] = (matrix[0][1] * matrix[2][0] - matrix[0][0] * matrix[2][1]) % mod;
      adj[2][2] = (matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]) % mod;
```

```
        for (int i = 0; i < 3; i++)
            for (int j = 0; j < 3; j++)
                adj[i][j] = (adj[i][j] + mod) % mod;
        return adj;
    }
    public static int[] solveModularSystem(int[][] matrix, int[] constants, int mod)
    {
        int det = determinantModulo(matrix, mod);
        int detInverse = modInverse(det, mod);
        int[][] adj = adjugateMatrix(matrix, mod);
        int[] solution = new int[3];
        for (int i = 0; i < 3; i++) {
            solution[i] = 0;
            for (int j = 0; j < 3; j++) {
                solution[i] = (solution[i] + adj[i][j] * constants[j]) % mod;
            }
            solution[i] = (solution[i] * detInverse) % mod;
        }
        return solution;
    }
    public static void main(String[] args)
 {
        Scanner scanner = new Scanner(System.in);
        int[][] matrix = new int[3][3];
        int[] constants = new int[3];
        System.out.println("Enter the modulo value:");
        int mod = scanner.nextInt();
        System.out.println("Enter the coefficients of the equations (row-wise):");
        for (int i = 0; i < 3; i++)
            for (int j = 0; j < 3; j++)
                matrix[i][j] = scanner.nextInt() % mod;
        System.out.println("Enter the constant terms:");
        for (int i = 0; i < 3; i++)
            constants[i] = scanner.nextInt() % mod;
        try {
            int[] solution = solveModularSystem(matrix, constants, mod);
            System.out.println("Solution:");
            for (int i = 0; i < 3; i++)
                System.out.printf("x%d ≡ %d (mod %d)\n", i + 1, solution[i], mod);
        }
 catch (ArithmeticException e)
  {
            System.out.println("No unique solution exists.");
        }

        scanner.close();
    }
}
```

7. Write a Java program to implement Caeser Cipher/Additive Cipher/Shift Cipher

```
import java.util.Scanner;
public class caeserCipher
{
    public static String encrypt(String plaintext, int key)
    {
      String plaintext1=plaintext.toLowerCase();
      String ciphertext = new String();
      for (int i = 0; i < plaintext1.length(); i++)
      {
         int pascii = (int)plaintext1.charAt(i);
         int result = (pascii+key)%256;
         char encryptedChar = (char) (result);
         ciphertext=ciphertext+encryptedChar;
      }
      return ciphertext.toUpperCase();
   }
     public static String decrypt(String ciphertext, int key)
      {
      String retrievedplaintext = new String();
      for (int i = 0; i <ciphertext.length(); i++) {
         int cascii = (int)ciphertext.charAt(i);
         int result = (cascii-key)%256;
         char decryptedChar = (char) (result);
         retrievedplaintext=retrievedplaintext+decryptedChar;
      }
      return retrievedplaintext.toLowerCase();
   }
      public static void main(String[] args)
      {
      Scanner scanner = new Scanner(System.in);
       System.out.println("Enter plaintext: ");
      String plaintext = scanner.nextLine();
      System.out.println("Enter key: ");
      int key = scanner.nextInt();
      String encryptedText = encrypt(plaintext, key);
      System.out.println("Encrypted Text: " + encryptedText);
      String decryptedText = decrypt(encryptedText, key);
      System.out.println("Decrypted Text: " + decryptedText);
      scanner.close();
   }
}
```

8. Write a Java program to implement Multiplicative Cipher

```
import java.util.Scanner;
public class multipliactiveCipher
{
      public static String encrypt(String plaintext, int key)
      {
      String plaintext1=plaintext.toLowerCase();
      StringBuffer ciphertext = new StringBuffer();
      for (int i = 0; i < plaintext1.length(); i++) {
         int pascii = (int)plaintext1.charAt(i);
         int result = (pascii*key)%256;
         char encryptedChar = (char) (result);
         ciphertext=ciphertext.append(encryptedChar);
      }
      return ciphertext.toString();
   }
   public static String decrypt(String ciphertext, int key)
   {
      StringBuffer retrievedplaintext = new StringBuffer();
      for (int i = 0; i <ciphertext.length(); i++)
       {
         int cascii = (int)ciphertext.charAt(i);
         int inversekey=multiplicativeInverse(key);
         int result = (cascii*inversekey)%256;
         char decryptedChar = (char) (result);
         retrievedplaintext=retrievedplaintext.append(decryptedChar);
      }
      return retrievedplaintext.toString();
   }
   public static int multiplicativeInverse(int key)
   {
      int m=256;
      int m0=m;
      int y = 0, x = 1;
      while (key > 1) {
         int q = key / m;
         int t = m;
         m = key % m;
         key = t;
         t = y;
         y = x - q * y;
         x = t;
      }
     if (x < 0)
         x += m0;
      return x;
   }
  public static void main(String[] args)
 {
      Scanner scanner = new Scanner(System.in);
      System.out.println("Enter plaintext: ");
      String plaintext = scanner.nextLine();
      System.out.println("Enter key: ");
      int key = scanner.nextInt();
```

```
        String encryptedText = encrypt(plaintext, key);
        System.out.println("Encrypted Text: " + encryptedText);
        String decryptedText = decrypt(encryptedText, key);
        System.out.println("Decrypted Text: " + decryptedText);
        scanner.close();
    }
}
```

9. Write a Java program to implement Affine Cipher

```
import java.util.Scanner;
public class affineCipher
{
      public static String encrypt(String plaintext, int key1,int key2)
      {
      String plaintext1=plaintext.toLowerCase();
      StringBuffer ciphertext = new StringBuffer();
      for (int i = 0; i < plaintext1.length(); i++) {
         int pascii = (int)plaintext1.charAt(i);
         int result = (pascii*key1+key2)%256;
         char encryptedChar = (char) (result);
         ciphertext=ciphertext.append(encryptedChar);
      }
      return ciphertext.toString();
   }
   public static String decrypt(String ciphertext, int key1, int key2)
    {
      StringBuffer retrievedplaintext = new StringBuffer();
      for (int i = 0; i <ciphertext.length(); i++)
       {
         int cascii = (int)ciphertext.charAt(i);
         int inversekey=multiplicativeInverse(key1);
         int result = ((cascii-key2)*inversekey)%256;
         char decryptedChar = (char) (result);
         retrievedplaintext=retrievedplaintext.append(decryptedChar);
      }
      return retrievedplaintext.toString();
   }
   public static int multiplicativeInverse(int key)
    {
      int m=256;
      int m0=m;
      int y = 0, x = 1;
     while (key > 1)
      {
         int q = key / m;
         int t = m;
         m = key % m;
         key = t;
         t = y;
         y = x - q * y;
         x = t;
      }
      if (x < 0)
         x += m0;
      return x;
   }
    public static void main(String[] args)
    {
      Scanner scanner = new Scanner(System.in);
      System.out.println("Enter plaintext: ");
      String plaintext = scanner.nextLine();
      System.out.println("Enter the first key: ");
```

```
        int key1 = scanner.nextInt();
        System.out.println("Enter the second key: ");
        int key2 = scanner.nextInt();
        String encryptedText = encrypt(plaintext, key1, key2);
        System.out.println("Encrypted Text: " + encryptedText);
        String decryptedText = decrypt(encryptedText, key1, key2);
        System.out.println("Decrypted Text: " + decryptedText);
        scanner.close();
    }
}
```

10. Write a Java program to implement Vernam Cipher

```
import java.util.Scanner;
public class VernamCipher
 {
   public static String encrypt(String plaintext, String key)
    {
      StringBuilder ciphertext = new StringBuilder();
      for (int i = 0; i < plaintext.length(); i++)
       {
        char encryptedChar = (char) (plaintext.charAt(i) ^ key.charAt(i % key.length()));
         ciphertext.append(encryptedChar);
      }
      return ciphertext.toString();
   }
    public static String decrypt(String ciphertext, String key)
    {
      StringBuilder plaintext = new StringBuilder();
      for (int i = 0; i < ciphertext.length(); i++)
      {
         char decryptedChar = (char) (ciphertext.charAt(i) ^ key.charAt(i % key.length()));
         plaintext.append(decryptedChar);
      }
      return plaintext.toString();
   }
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter plaintext: ");
      String plaintext = scanner.nextLine();
      System.out.print("Enter key: ");
      String key = scanner.nextLine();
      String encryptedText = encrypt(plaintext, key);
      System.out.println("Encrypted Text: " + encryptedText);
      String decryptedText = decrypt(encryptedText, key);
      System.out.println("Decrypted Text: " + decryptedText);
      scanner.close();
   }
}
```

11. Write a Java Program to implement Hill Cipher

```
import java.util.Scanner;
public class HillCipher
 {
   public static String encrypt(String plaintext, int[][] key)
   {
      StringBuilder ciphertext = new StringBuilder();
      while (plaintext.length() % 3 != 0)
       {
         plaintext += 'X';
      }
      plaintext = plaintext.toUpperCase();
      for (int i = 0; i < plaintext.length(); i += 3)
      {
         int[] block = new int[3];
         for (int j = 0; j < 3; j++)
        {
            block[j] = plaintext.charAt(i + j) - 'A';
         }
         int[] result = multiplyMatrix(key, block);
         for (int j = 0; j < 3; j++)
         {
            ciphertext.append((char) (result[j] % 26 + 'A'));
         }
      }
      return ciphertext.toString();
   }
   private static int[] multiplyMatrix(int[][] matrix, int[] vector)
    {
      int[] result = new int[3];
      for (int i = 0; i < 3; i++)
       {
         result[i] = 0;
         for (int j = 0; j < 3; j++)
       {
            result[i] += matrix[i][j] * vector[j];
         }
      }
      return result;
   }
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter plaintext: ");
      String plaintext = scanner.nextLine();
      int[][] key = new int[3][3];
      System.out.println("Enter key matrix (3x3):");
      for (int i = 0; i < 3; i++)
      {
         for (int j = 0; j < 3; j++)
        {
            key[i][j] = scanner.nextInt();
         }
      }
```

```
        String encryptedText = encrypt(plaintext, key);
        System.out.println("Encrypted Text: " + encryptedText);
        scanner.close();
    }
}
```

12. Write a Java Program to implement RSA Algorithm

```
import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.*;
public class RSAAlgorithm
{
   private BigInteger privateKey;
   private BigInteger publicKey;
   private BigInteger n;
   public RSAAlgorithm(int bitLength)
   {
      SecureRandom random = new SecureRandom();
      BigInteger p = BigInteger.probablePrime(bitLength / 2, random);
      System.out.println("First Prime Number p="+p);
      BigInteger q = BigInteger.probablePrime(bitLength / 2, random);
      System.out.println("Second Prime Number q="+q);
      n = p.multiply(q);
      BigInteger phi = (p.subtract(BigInteger.ONE)).multiply(q.subtract(BigInteger.ONE));
      publicKey = generatePublicKey(phi, random);
      privateKey = publicKey.modInverse(phi);
   }
   private BigInteger generatePublicKey(BigInteger phi, SecureRandom random)
     {
      BigInteger e;
      do {
         e = new BigInteger(phi.bitLength(), random);
      } while (e.compareTo(BigInteger.ONE) <= 0 || e.compareTo(phi) >= 0 ||
!e.gcd(phi).equals(BigInteger.ONE));
      return e;
   }
   public BigInteger[] encrypt(String message)
   {
      byte[] bytes = message.getBytes();
      BigInteger[] encrypted = new BigInteger[bytes.length];
      for (int i = 0; i < bytes.length; i++) {
         encrypted[i] = BigInteger.valueOf(bytes[i]).modPow(publicKey, n);
      }
      return encrypted;
   }
   public String decrypt(BigInteger[] ciphertext)
   {
      byte[] decryptedBytes = new byte[ciphertext.length];
      for (int i = 0; i < ciphertext.length; i++)
      {
         decryptedBytes[i] = ciphertext[i].modPow(privateKey, n).byteValue();
      }
      return new String(decryptedBytes);
   }
   public static void main(String[] args)
   {
      int bitLength = 1024;
      RSAAlgorithm rsa = new RSAAlgorithm(bitLength);
      Scanner s=new Scanner(System.in);
      System.out.println(“Enter the Plaintext”);
```

```
        String message =s.nextLine();
        BigInteger[] encryptedMessage = rsa.encrypt(message);
        System.out.println("Encrypted Message: " + encryptedMessage);
        String decryptedMessage = rsa.decrypt(encryptedMessage);
        System.out.println("Decrypted Message: " + decryptedMessage);
    }
}
```

13. Write a Java program to implement DES Algortihm

```
import javax.crypto.*;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
import java.util.Scanner;
public class DESExample
{
   public static String encrypt(String plaintext, String key)
   {
      try
      {
         Cipher ci = Cipher.getInstance("DES");
         SecretKeySpec ks = new SecretKeySpec(key.getBytes(), "DES");
         ci.init(Cipher.ENCRYPT_MODE, ks);
         byte[] encryptedBytes = ci.doFinal(plaintext.getBytes());
         return Base64.getEncoder().encodeToString(encryptedBytes);
      }
     catch (Exception e)
     {
         e.printStackTrace();
         return null;
      }
   }
     public static String decrypt(String ciphertext, String key)
    {
      try
      {
         Cipher cipher = Cipher.getInstance("DES");
         SecretKeySpec ks = new SecretKeySpec(key.getBytes(), "DES");
         cipher.init(Cipher.DECRYPT_MODE, ks);
         byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(ciphertext));
         return new String(decryptedBytes);
      }
     catch (Exception e)
     {
         e.printStackTrace();
         return null;
      }
   }
   public static void main(String[] args)
    {
      Scanner scanner = new Scanner(System.in);
      try
      {
        System.out.print("Enter text: ");
         String text = scanner.nextLine();
         System.out.print("Enter key (8 characters): ");
         String key = scanner.nextLine();
         String encryptedText = encrypt(text, key);
         System.out.println("Encrypted Text: " + encryptedText);
         String decryptedText = decrypt(encryptedText, key);
         System.out.println("Decrypted Text: " + decryptedText);
      }
     finally
```

```
    {
        scanner.close();
    }
  }
}
```

14. Write a Java Program to generate Message Digest using MD5 Algorithm

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.*;
public class MD5Example
 {
  public static String generateMD5(String input)
{
       try
       {
         MessageDigest md = MessageDigest.getInstance("MD5");
           md.update(input.getBytes());
           byte[] digest = md.digest();
           StringBuilder result = new StringBuilder();
           for (byte b : digest)
           {
              result.append(String.format("%02x", b & 0xff));
           }
           return result.toString();
       }
      catch (NoSuchAlgorithmException e)
       {
           e.printStackTrace();
           return null;
       }
   }
   public static void main(String[] args)
    {
      Scanner s=new Scanner(System.in);
      System.out.println("Enter the Message");
      String input =s.nextLine();
      String md5Hash = generateMD5(input);
      System.out.println("MD5 Hash: " + md5Hash);
   }
}
```

15. Write a Java program to generate the hash code using SHA256 Algorithm

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Scanner;
public class SHA256Example
{
 public static String generateSHA256(String input)
  {
      try
      {
         MessageDigest digest = MessageDigest.getInstance("SHA-256");
         byte[] hash = digest.digest(input.getBytes());
         StringBuilder hexString = new StringBuilder();
         for (byte b : hash)
         {
            String hex = Integer.toHexString(0xff & b);
            if (hex.length() == 1)
            {
               hexString.append('0');
            }
            hexString.append(hex);
         }
         return hexString.toString();
      }
     catch (NoSuchAlgorithmException e)
     {
         e.printStackTrace();
         return null;
      }
   }
   public static void main(String[] args)
    {
      Scanner scanner = new Scanner(System.in);
      System.out.println("Enter the Message:");
      String input = scanner.nextLine();
      String sha256Hash = generateSHA256(input);
      System.out.println("SHA-256 Hash: " + sha256Hash);
      scanner.close();
   }
}
```