CRYPTO LAB-6

NAME: ABISHEK K G **REG NO: 23BCE1739**

DATE: 27/02/2026 **SLOT: L29+L30**

Code:

```
public class RSAHomomorphic {

    // ---------- GCD ----------
    static long gcd(long a, long b) {
        while (b != 0) {
            long t = b;
            b = a % b;
            a = t;
        }
        return a;
    }

    // -------- Extended Euclidean ----------
    static long[] extended_euclidean(long a, long b) {
        if (b == 0)
            return new long[]{a, 1, 0};

        long[] val = extended_euclidean(b, a % b);
        long g = val[0];
        long x1 = val[2];
        long y1 = val[1] - (a / b) * val[2];
```

```
    return new long[]{g, x1, y1};
}

// -------- Modular Inverse ----------
static long mod_inverse(long e, long phi) {
    long[] val = extended_euclidean(e, phi);
    long g = val[0];
    long x = val[1];

    if (g != 1) {
        System.out.println("Inverse does not exist");
        return -1;
    }

    return (x % phi + phi) % phi;
}

// ---- Modular Exponentiation ----------
static long mod_exp(long base, long exp, long mod) {
    long res = 1;
    base = base % mod;

    while (exp > 0) {
        if (exp % 2 == 1)
            res = (res * base) % mod;
```

```
        base = (base * base) % mod;
        exp = exp / 2;
    }

    return res;
}

// ------------- Encrypt ----------------
static long encrypt(long m, long e, long n) {
    return mod_exp(m, e, n);
}

// ------------- Decrypt ----------------
static long decrypt(long c, long d, long n) {
    return mod_exp(c, d, n);
}

// ================= MAIN =================
public static void main(String[] args) {

    // ===== Stage 1: Key Generation =====
    long p = 11;     // small prime
    long q = 17;     // small prime

    long n = p * q;
```

```
long phi = (p - 1) * (q - 1);

long e = 7;      // choose e such that gcd(e, phi) = 1

System.out.println("---- RSA KEY GENERATION ----");
System.out.println("p = " + p);
System.out.println("q = " + q);
System.out.println("n = p*q = " + n);
System.out.println("phi(n) = " + phi);
System.out.println("gcd(e, phi) = " + gcd(e, phi));

long d = mod_inverse(e, phi);

System.out.println("Public Key (e, n) = (" + e + ", " + n + ")");
System.out.println("Private Key (d, n) = (" + d + ", " + n + ")");

// ===== Stage 2: Encryption/Decryption =====
long M = 5;

System.out.println("\n---- ENCRYPTION / DECRYPTION ----");
System.out.println("Plaintext M = " + M);

long C = encrypt(M, e, n);
System.out.println("Ciphertext C = M^e mod n = " + C);

long M_dec = decrypt(C, d, n);
```

```
System.out.println("Decrypted M = C^d mod n = " + M_dec);

if (M == M_dec)
   System.out.println("Decryption Verified");
else
   System.out.println("Decryption Failed");

// ===== Stage 3: Homomorphic Property =====
long m1 = 4;
long m2 = 3;

System.out.println("\n---- HOMOMORPHIC PROPERTY TEST ----");
System.out.println("m1 = " + m1);
System.out.println("m2 = " + m2);

long c1 = encrypt(m1, e, n);
long c2 = encrypt(m2, e, n);

System.out.println("E(m1) = " + c1);
System.out.println("E(m2) = " + c2);

long c_mul = (c1 * c2) % n;
System.out.println("C_mul = E(m1)*E(m2) mod n = " + c_mul);

long dec_mul = decrypt(c_mul, d, n);
System.out.println("D(C_mul) = " + dec_mul);
```

```
        long expected = (m1 * m2) % n;
        System.out.println("(m1*m2) mod n = " + expected);

        if (dec_mul == expected)
            System.out.println("Multiplicative Homomorphic Property Verified");
        else
            System.out.println("Homomorphic Property Not Verified");
    }
}
```

OUTPUT: