Cifrado Asimétrico

Variables privadas

```
private static KeyPair keyPair;  3 usages
private static KeyPairGenerator key;  5 usages
private static PublicKey publicKey;  3 usages
private static PrivateKey privateKey;  3 usages
private static byte[] msgEncriptar;  3 usages
private static byte[] msgDesencriptar;  2 usages
```

Declaro las variables para **Keys** y los bytes para los mensajes.

Menú

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

    while (!exit) {
        System.out.println("\n--- MenU RSA ---");
        System.out.println("1. Generar par de claves");
        System.out.println("2. Cifrar mensaje");
        System.out.println("3. Descifrar mensaje");
        System.out.println("4. Salir");
        System.out.print("Eligir opción: ");
        int option = 0;

        try {
            option = sc.nextInt();
            if (option>4 || option<1){
                System.out.println("\nOpcion incorrecta, 1-4");
            }
        }
        catch (Exception ex){
            System.out.println("\nOpcion incorrecta, solo enteros");
        }
        sc.nextLine();

        String msg;
```

Se repite el menú infinitamente y se dan las opciones y un control de errores para la opción.

Case1

```
switch (option){
    case(1):
        try {
            key = KeyPairGenerator.getInstance( algorithm: "RSA");
            key.initialize( keysize: 2048);
            keyPair = key.generateKeyPair();
            publicKey = keyPair.getPublic();
            privateKey = keyPair.getPrivate();
            System.out.println("\nKeys Generadas");
        }
        catch (Exception ex){
            System.out.println("\nKey no generada");
        }
        break;
```

Switch con el primer **case**, con un try género todas las llaves necesarias.

Case2

```
case(2):
    if (publicKey == null){
        System.out.println("\nGenera las claves primero.");
    }
    else{
        System.out.println("Introduce el mensaje: ");
        msg = sc.nextLine();
        msgEncriptar = encriptar(msg.getBytes(), publicKey, key.getAlgorithm());
        System.out.println("Texto encriptado:\n"+Base64.getEncoder().encodeToString(msgEncriptar));
    }
    break;
```

Case2 compruebo si las claves han sido generadas y pido **input** del mensaje para pasarlo a la función **encriptar()** para luego mostrar en **Base64** el mensaje encriptado.

Case3

```
case(3):
    if (privateKey == null){
        System.out.println("\nGenera las claves primero.");
    }
    else {
        System.out.println("Introduce el mensaje: ");
        msg = sc.nextLine();
        msgDesencriptar = desencriptar(msgEncriptar, privateKey, key.getAlgorithm());
        System.out.println("Texto desencriptado:\n" + new String(msgDesencriptar));
    }
    break;
```

Tercer **case** donde vuelvo a comprobar si se han generado las claves y pido **input** del mensaje encriptado. Llamo a la función **desencriptar()** con el mensaje encriptado y las claves. Muestro el mensaje desencriptado utilizando **new String**.

Case4

```
case(4):
    System.out.println("Cerrando...");
    exit = true;
    break;
```

Ultimo **case** simplemente para cambiar el **bool** y salir del bucle del menú y cerrar programa.

encriptar()

```
public static byte[] encriptar(byte[] inputBytes, PublicKey publicKey, String algorithm){
    try{
        Cipher cipher = Cipher.getInstance(algorithm);
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);
        return cipher.doFinal(inputBytes);
    }
    catch (Exception ex){
        System.out.println("Error: "+ex);
    }
    return null;
}
```

Función para encriptar, pasándole los atributos: **bytes de mensaje**, **publicKey** y **algoritmo.**

desencriptar()

```java
public static byte[] desencriptar(byte[] inputBytes, PrivateKey privateKey, String algorithm){
    try {
        Cipher cipher = Cipher.getInstance(algorithm);
        cipher.init(Cipher.DECRYPT_MODE, privateKey);
        return cipher.doFinal(inputBytes);
    }
    catch (Exception ex){
        System.out.println("Error: "+ex);
    }
    return null;
}
```

Función para desencriptar, pasándole los atributos: **bytes de mensaje**, **privateKey** y **algoritmo.**