Task 1: Encryption and Decryption

Main Class

Algorithm 1: Main Class

Input: None
Output: Encrypted and decrypted numbers

```
1  Print the program header;
2  userNumber ← Ask the user to input a 4-digit number;
3  encryptedNumber ← EncryptuserNumber;
4  Print the encryptedNumber;
5  decryptedNumber ← DecryptencryptedNumber;
6  Print the decryptedNumber;
7  Print the program footer;
```

Algorithm 2: Encrypt Utility Function

Input: String input
Output: Encrypted string

```
1  if input is null OR input length is not 4 OR input is not a number then
2  |  Throw an IllegalArgumentException;
3  end
4  digits ← convert input to an array of characters;
5  for each character in digits do
6  |  digit ← convert character to an integer;
7  |  digit ← (digit + 7) mod 10;
8  |  character ← convert digit back to a character;
9  |  replace character in digits;
10 end
11 Swap the first and third characters in digits;
12 Swap the second and fourth characters in digits;
13 encryptedNumber ← convert digits back to a string;
14 return encryptedNumber;
```

Algorithm 3: Decrypt Utility Function

Input: String input
Output: Decrypted string

if *input is null OR input length is not 4 OR input is not a number* **then**
    Throw an IllegalArgumentException;
end
digits ← convert input to an array of characters;
Swap the first and third characters in digits;
Swap the second and fourth characters in digits;
for *each character in digits* **do**
    digit ← convert character to an integer;
    digit ← (digit - 7 + 10) mod 10;
    character ← convert digit back to a character;
    replace character in digits;
end
decryptedNumber ← convert digits back to a string;
return decryptedNumber;

Sieve Utility Function

Algorithm 4: Sieve Utility Function

Input: Integer limit
Output: Array of prime numbers

Create a boolean array `isPrime` of size `limit` and initialize all elements to true;
Create an integer array `prime` of size `limit`;
for *i from 2 to* $\sqrt{limit}$ **do**
    if *isPrime[i] is true* **then**
        for *j from* $i \times 2$ *to limit step i* **do**
            `isPrime[`j`]` ← false;
        end
    end
end
for *i from 2 to limit* **do**
    if *isPrime[i] is true* **then**
        Append i to `prime`;
    end
end
return *prime*;