**ENGIENEERING METHOD PROYECT REPORT**

1. **Problem identification**

Since the company will improved its security system, cryptography will have a fundamental role in the process, since it is a very secure way to protect data.

In order to implement the cryptography-based security system, they need a program that generates (n) prime numbers (being ‘n’ the desired amount). Since currently they don’t have any function or program within their system that can perform this task. Prime numbers are needed in order to implement effective encryption algorithms.

The prime numbers must be displayed on the screen, arranged in a matrix.

Functional Requirements

|  |  |
| --- | --- |
| **Name:** | R. #1 Generate prime numbers. |
| **Description:** | The program must be able to generate (n) prime numbers. It must have three algorithms that can perform this task. |
| Input: | Amount (n) of prime numbers |
|  |  |
| Output: | A table bidimensional with de first (n) prime numbers |
|  |  |

|  |  |
| --- | --- |
| **Name:** | R. #2 Get input |
| **Description:** | The program must be able to receive the input from the user via a GUI.  This input must be an integer (n) >0 and will be used to generate the prime numbers and create a matrix containing them in order to display them on the screen. |
| Input: | An input (n) that represents the maximum amount of prime numbers that must be generated. |
|  |  |
| Output: | <None> |
|  |  |

|  |  |
| --- | --- |
| **Name:** | R. #3 Generate Matrix |
| **Description:** | The program must generate a matrix containing all the integers from 0 to (n) where (n) is an input given by the user. |
| Input: | Input (n) |
|  |  |
| Output: | A matrix containing all the numbers from 0 to (n) |
|  |  |

|  |  |
| --- | --- |
| **Name:** | R. #4 Differeance the primes numbers |
| **Description:** | as the algorithm finds that the number is or is not a prime, that is, that allows to show in real time the process performed by the algorithm to find these prime numbers. |
| Input: | <None> |
|  |  |
| Output: | Green: Prime numbers |
|  | Red: Numbers not prime |

1. **Relevant information compilation**

Fuente:

<https://whatis.techtarget.com/definition/prime-number>

<https://crypto.stackexchange.com/questions/20867/why-are-primes-important-for-encryption>

Prime number:

A prime number is a whole number greater than 1 whose only factors are 1 and itself. A factor is a whole number that can be divided evenly into another number. Numbers that have more than two factors are called composite numbers. The number 1 is neither prime nor composite.

Imagen que contiene captura de pantalla

Descripción generada automáticamente

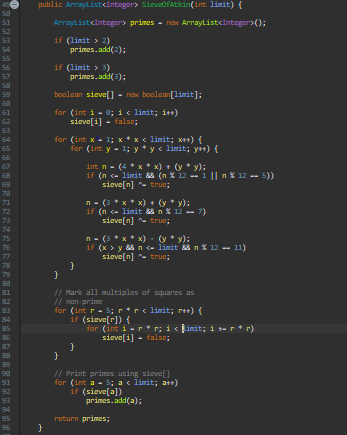
Prime numbers and cryptography:

Encryption always follows a fundamental rule: the algorithm or the actual procedure being used doesn’t need to be kept secret, but the key does. Even the most sophisticated hacker in the world will be unable to decrypt data as long as the key remains secret and prime numbers are very useful for creating keys. For example, the strength of public/private key encryption lies in the fact that it's easy to calculate the product of two randomly chosen prime numbers, but it can be very difficult and time consuming to determine which two prime numbers were used to create an extremely large product, when only the product is known. This problem is called prime factorization and finding an algorithm which does it fast is one of the unsolved problems of computer science.

1. **Creative Solutions search**

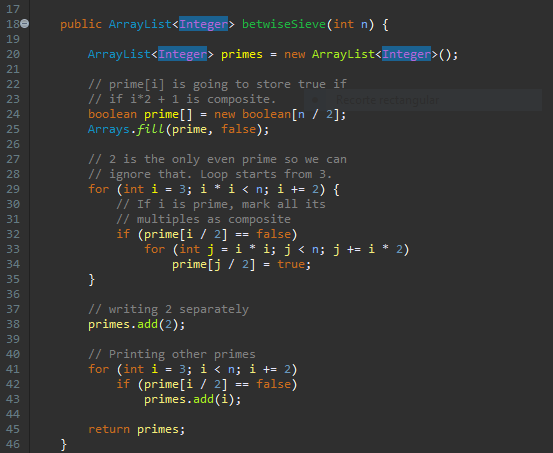
We will tackle the problem of generating prime numbers using different approaches.

**-Sieve of Atkin:**



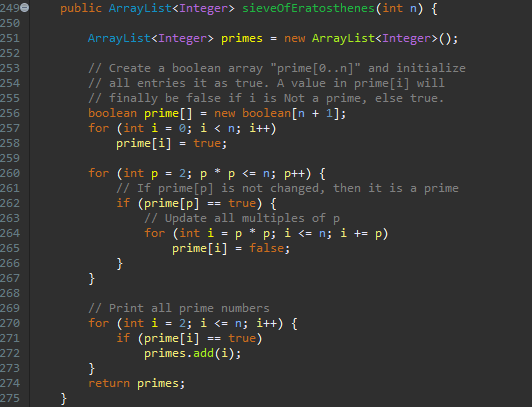
* This algorithm adds numbers 2 and 3 to the array list if the input is bigger than them.
* Then creates an array of Boolean and put some indexes like true.
* In the square indices, there are indicated as false
* All indexes indicated as true are added to the array list

**-Sieve of Betwise:**



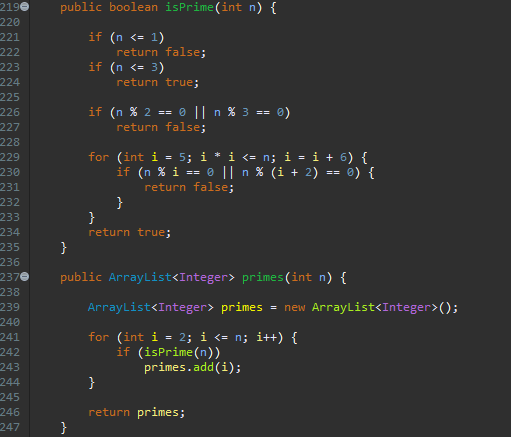
* This method fills an array of Boolean with false
* Add the number 2 to the array
* If find a prime number mark all its multiples as false
* Add primes to an array list

**-Sieve of Eratosthenes:**



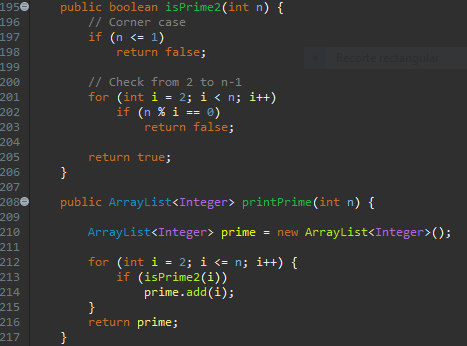
* This method fills an array of Boolean with true
* Indicates squares of prime numbers as false
* Add all primes to an array list

**-Simple methods to find prime numbers: 1.**



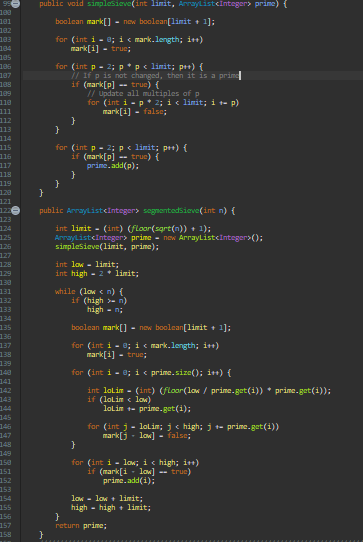
* This algorithm search primes looking each number since 2 to n
* To know if a number is a prime first look if is divisible by 2 and 3 and if it’s the square of another prime number
* If the number is prime its added to an array list

**2.**



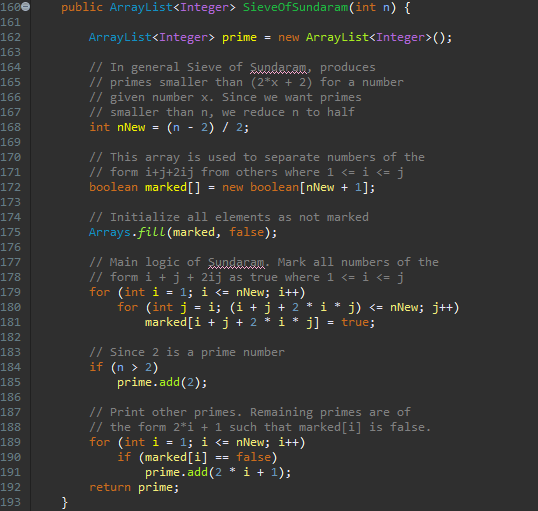
* This algorithm search primes looking each number since 2 to n
* To know if a number is a prime look if the number is divisible by another number since 2 to n
* If the number is prime its added to an array list

**-Segmented sieve:**



* This algorithm takes the square root of the input and find all primes since 2 to that number with the simple sieve and add the numbers to an array list
* Then find more prime numbers with the primes found before with an array of Boolean

**-Sieve of Sundaram:**



* This algorithm reduces the input by 2, then divides that number by 2 and creates an array of Boolean filled with false.
* Then marks some indexes that are not prime with true.
* If find an index with false, add the resulting number of (2 \* index + 1) to an array list

1. **Transition of ideas formulation to preliminary designs**

In this phase we’re going to discard some solutions from the previous phase. Solutions that will be discard are the following:

* Sieve of Atkin
* Simple method to find prime numbers (1 and 2)
* Segmented sieve

The reason why we’re going to discard these solutions is because are not very efficient and a bit unstable.

1. **Evaluation and selection of the best solution**

**Evaluation criteria #1:** Efficiency, that is, the number of lines required to reach the solution.

1. Give a solution by executing a ridiculous amount of lines.
2. Give a solution by running many lines.
3. Give a solution by running few lines

**Evaluation criteria #2:** Code decoupling.

1. Very coupled, hard to reuse
2. Uncoupled, can be reused in other solutions

|  |  |  |  |
| --- | --- | --- | --- |
|  | Evaluation criteria #1 | Evaluation criteria #2 | Total |
| Sieve of Atkin | 1 | 1 | 2 |
| Sieve of Betwise | 3 | 2 | 5 |
| Sieve of Eratosthenes | 3 | 2 | 5 |
| Simple method #1 | 2 | 1 | 3 |
| Simple method #2 | 1 | 1 | 2 |
| Segmented sieve | 1 | 1 | 2 |
| Sieve of Sundaram | 3 | 2 | 5 |