***Identificación del problema***

Una parte fundamental de la mayoría del software producido actualmente es la seguridad de los datos, esto es, mantener la triada CID: Confidencialidad, Integridad y Disponibilidad de la información. Para esto, hay una rama de la informática que juega un papel muy importante llamada Criptografía, la cual se ocupa, entre otras cosas, de generar técnicas de cifrado que permitan mantener segura la información. Por ello una empresa ha decidido tomarse en serio la seguridad de sus sistemas y necesita un programa que genere números primos dado que estos presentan una singularidad y es que son únicos respecto a que solo son divisibles por 1 y por ellos mismos.

***Requerimientos funcionales***

RF1 Verificar si el número es primo dividiendo el número ingresado (n) por números desde 2 hasta el número ingresado menos 1 (n-1).

RF2 Verificar si un número es par dividiéndolo por 2, en caso de que no lo sea se divide por todos los números desde 3 al número ingresando sin contar los números pares.

RF3 Verificar si un número es primo dividiéndolo por 2 en caso de que no lo sea se divide por todos los números desde el numero ingresado hasta el 3 sin contar los números pares.

RF4 Hacer una interfaz gráfica de usuario

|  |  |
| --- | --- |
| RF1 | Método 1 Verificación de numero primo |
| Resumen | Verificar si el número es primo dividiendo el número ingresado (n) por números desde 2 hasta el número ingresado menos 1 (n-1) |
| Entrada | Numero entero hasta el cual se van a buscar números primos |
| Salida | Los números primos hasta el valor ingresado |

|  |  |
| --- | --- |
| RF2 | Método 2 Verificación de numero primo |
| Resumen | Verificar si un número es par dividiéndolo por 2, en caso de que no lo sea se divide por todos los números desde 3 al número ingresando sin contar los números pares. |
| Entrada | Número entero hasta el cual se van a buscar números primos |
| Salida | Los números primos hasta el valor ingresado |

|  |  |
| --- | --- |
| RF3 | Método 3 Verificación de numero primo |
| Resumen | Verificar si un número es primo dividiéndolo por 2 en caso de que no lo sea se divide por todos los números desde el número ingresado hasta el 3 sin contar los números pares. |
| Entrada | Numero entero hasta el cual se van a buscar números primos |
| Salida | Los números primos hasta el valor ingresado |

|  |  |
| --- | --- |
| RF4 | Hacer interfaz grafica |
| Resumen | Hacer una interfaz gráfica intuitiva para el usuario |
| Entrada | N/A |
| Salida | La interfaz de usuario es mostrada |

***Recopilación información necesaria***

Para darle comienzo al proyecto se consideraron los siguientes puntos

1. ¿Qué es un número primo?: Un numero primo es un natural entero positivo aquel que solo es divisible entre el mismo y uno, dando como resultado de la división otro número entero.
2. ¿Cómo se podía generar una matriz lo más cuadrada posible?: En primer lugar para todos los números se les puede sacar la raíz cuadrada sin embargo no todos tienen una raíz cuadrada exacta, por ejemplo, la raíz cuadrada de 100 es 10 lo que nos daría una matriz 10 x 10 pero la raíz de 133 es 11.53, la solución a este problema es tomar la parte entera de la raíz del número ingresado que no da una raíz exacta (En este caso 133 con raíz 11.53) y sumarle 1 (en este caso sumarle 1 a 11) de esta manera la matriz quedara lo más cuadrada posible
3. ¿Todos los números pares no son primos? Todos los números pares no son primos excepto el 2 porque es divisible solo entre 1 y el mismo, y todos los demás pares serán divisibles entre 2, 1 y ellos mismos. Esto nos introduce al concepto de la Criba de Eratóstenes.
4. ¿Qué es la Criba de Eratóstenes?: La Criba de Eratóstenes es un método para generar números primos y dice que todos los múltiplos de un numero primo no pueden ser números primos porque estos números ya serian divisibles por este, por ejemplo, 3 x 4 = 12, 3 x 5 = 15, 3 x 6 = 18 … todos los resultados de estas multiplicaciones serán divisibles entre 3, 12/4 = 3, 15/3 = 5, 18/3 = 6 …
5. ¿Cómo pintar la matriz de manera sucesiva? Para poder pintar cada casilla de la matriz en tiempo real la manera más viable es hacer el uso de un hilo de trabajo que se encargue de controlar la funcionalidad de pintado de casilla mientras que el resto del programa se encargaría del resto que sería generar la matriz e ir descartando los números primos o no.
6. ¿Qué errores podrían hacer que el programa se detenga? Uno de los errores que se pueden generar es que el usuario ingresara un numero negativo, para que diera los numero primos hasta ese número ingresado, esto generaba dos subproblemas, el primero es que no hay ningún número negativo que sea número primo ya que estos números por lo menos serán divisables por cuatro valores, por ejemplo -5, este número es divisible por 1, -1, 5, -5, además de que esto violenta la regla de que el numero debe ser natural positivo. Y el segundo problema es que no es posible sacarle una raíz cuadrada a un número negativo. Otro error era que se ingresara letras en vez de números.
7. ¿Qué es un numero natural?: Numero natural es aquel que no es negativo y es entero.

Fuentes: Ver Anexos.

***Búsqueda de soluciones creativas***

La técnica usada para la generación de ideas fue “Lluvia de ideas”, básicamente consiste en dar diferentes tipos de soluciones sin discriminarlas y sin importar que tan inconveniente sea, para después seleccionar las más viables.

***Ideas:***

1. Tomar el valor ingresado por el usuario y comenzar a dividir este por cada uno de sus números predecesores desde el 2 hasta el n ingresado, si el único numero con el que no hubo residuo fue con el mismo es un numero primo.
2. Tomar el valor ingresado por el usuario y comenzar a dividir este por cada uno de sus números predecesores desde 2 hasta la mitad (ascendentemente) y desde el n hasta la mitad (descendentemente), si el único número con el que no hubo residuo fue con el mismo es un numero primo.
3. Tomar el valor ingresado por el usuario y comenzar a dividir este por cada uno de sus números predecesores desde el n ingresado hasta el 2 (descendentemente), si el único número con el que no hubo residuo fue con el mismo es un numero primo.
4. Validar antes si el numero es diferente de 2 y si es par o no, después tomar el valor ingresado por el usuario y comenzar a dividir este por cada uno de sus números predecesores impares desde el 2 hasta el n ingresado, si el único número con el que no hubo residuo fue con el mismo es un numero primo.
5. Tomar el valor ingresado por el usuario y comenzar a dividir este por cada uno de sus números predecesores desde el 2 hasta el n-1 ingresado (ascendentemente), este procedimiento hasta que encuentre un número con el cual el residuo es cero, si este numero existe entonces el número no es primo.
6. Tomar el concepto de la criba de Eratóstenes que consiste en que todos los múltiplos de un numero primo no pueden ser números primos
7. Crear un archivo plano con los números primos del 1 al 100000 y tomar estos números para usarlos en el programa.

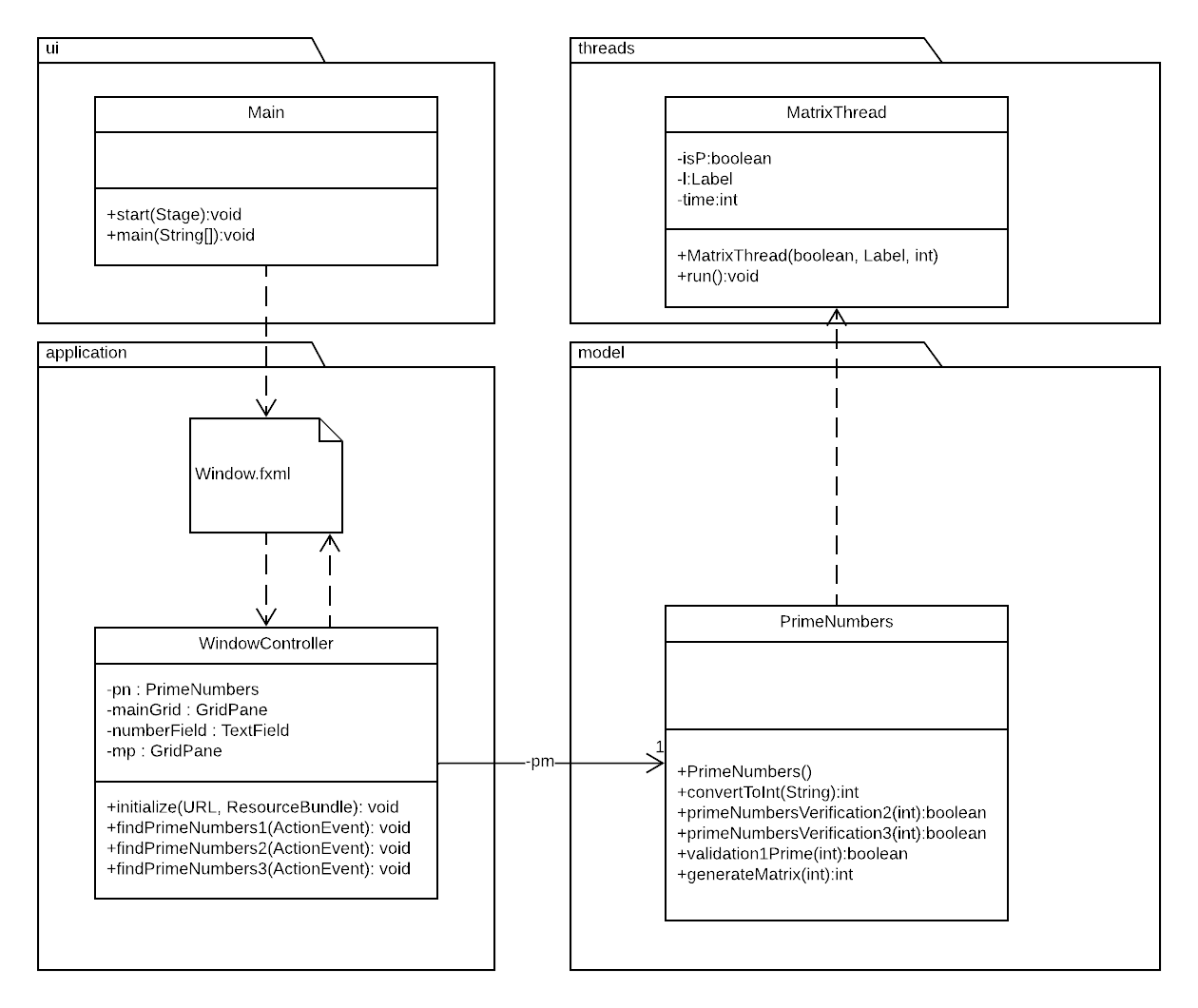
***Transición de la formulación de ideas de diseños preliminares***

En esta sección se explicará los motivos por los cuales se descartaron las 4 ideas menos viables para el programa.

Después de analizar cada una de las ideas creativas, se ha decidido que la idea número 7 es la primera en descartarse, puesto que solo puede calcular una limitada cantidad de números primos y además ineficiente y costoso por los archivos innecesarios que utiliza.   
(RESPETA MI IDEA JAJAJAJA)

***Evaluación y selección de la mejor solución***

***Diseño del diagrama de clases de la solución***



***Pseudocódigo de los algoritmos más relevantes***

***Diseño de Casos de Pruebas Unitarias***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Clase | Metodo | Escenario | Valores de entrada | Salida |
| PrimeNumbers | validation1Prime() | Al método se le darán números y contara la cantidad de números primos encontrados | Al método a testear se le pasan números del 1 al 100 | 25 que es la cantidad de números primos del 1 al 100 sin contar el 1 |
| PrimeNumbers | primeNumbersVerification2() | Al método se le darán números y contara la cantidad de números primos encontrados | Al método a testear se le pasan números del 1 al 100 | 25 que es la cantidad de números primos del 1 al 100 contando el 1 |
| PrimeNumbers | primeNumbersVerification2() | Al método se le darán números y contara la cantidad de números primos encontrados | Al método a testear se le pasan números del 1 al 100 | 25 que es la cantidad de números primos del 1 al 100 contando el 1 |
| PrimeNumbers | generateMatrix() | Al método se le da un numero con raíz cuadrada inexacta | Al método a testar se le da el numero 133 | 12, que es el siguiente número entero más cercano al del resultado de la raíz cuadrada |
| PrimeNumbers | generateMatrix() | Al método se le da un numero con raíz exacta | Al método a testear se le da el numero 100 | 10, que es el resultado de la raíz cuadrada |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Clase | Metodo | Escenario | Valores de entrada | Salida |
| PrimeNumbers | validation1Prime() | Al método se le darán números y verificara si es primo o no este valor de verdad booleano será guardado en un ArrayList | Al método a testear se le pasan números del 1 al 7 | Un ArrayList que contiene 4 valores de verdad positivos ya que hay 4 números primos sin contar el 1 (2,3,5,7) |
| PrimeNumbers | primeNumbersVerification2() | Al método se le darán números y verificara si es primo o no este valor de verdad booleano será guardado en un ArrayList | Al método a testear se le pasan números del 1 al 7 | Un ArrayList que contiene 5 valores de verdad positivos ya que hay 5 numeros primos (1,2,3,5,7) |
| PrimeNumbers | primeNumbersVerification2() | Al método se le darán números y verificara si es primo o no este valor de verdad booleano será guardado en un ArrayList | Al método a testear se le pasan números del 1 al 7 | Un ArrayList que contiene 5 valores de verdad positivos ya que hay 5 números primos (1,2,3,5,7) |
| PrimeNumbers | convertToInt() | Al método se le dará una cadena de texto que se pueda convertir a entero | Al método a testear se le pasara la cadena “22” | El numero 22 |
| PrimeNumbers | convertToInt() | Al método se le dará una cadena de texto que no se pueda convertir a entero | Al método a testear se le pasara la cadena “bb” | NumberFormat  Exception |
| PrimeNumbers | convertToInt() | Al método se le dará una cadena de texto que se pueda convertir a entero | Al método a testear se le pasara la cadena “-100” | NumberFormat  Exception |

***Análisis de complejidad temporal***

***Análisis de complejidad temporal***

**public** **boolean** primeNumbersVerification2(**int** n){

1 **boolean** aux = **true**;

2 **if**(n%2 == 0 && n!=2) {

3 aux = **false**;

4 }**else** {

5 **for**(**int** i = 3; i < n && aux; i+=2) {

6 **if**(n%i == 0) {

7 aux = **false**;

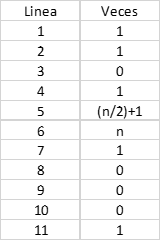
8 }

9 }

10 }

11 **return** aux;

}



n = 9

T (n) = (3/2)n+6

Analisis linea 5

i = 3, i = 5, i = 7, i = 9, i = 11 (Sale)

Sumando las veces i se le asigna un valor nos da 5

Tomando el entero 9 / 2 = 4.5 = 4

Tenemos 4 + 1 = 5

**public** **boolean** primeNumbersVerification3(**int** n){

1 **boolean** aux = **true**;

2 **for**(**int** i = n - 1; i > 1 && aux; i--) {

3 **if**(n%i == 0) {

4 aux = **false**;

5 }

6 }

7 **return** aux;

}



n = 5

T (n) = 2n+4

Analisis linea 2

i = 4, i = 3, i = 2, i = 1 (Sale)

Sumando las veces i se le asigna un valor nos da 4

n – 1 = 4 (Veces que se le asigno vaolor a i)

Tenemos 5 - 1 = 5

**public** **boolean** Validation1Prime(**int** numberToValidate) {

1 **int** counter = 0;

2 **boolean** ok = **false**;

3 **for**(**int** i = 2; i <= numberToValidate; i++) {

4 **if**(numberToValidate % i == 0) {

5 counter++;

6 }

7 }

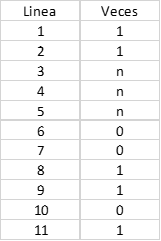
8 **if**(counter == 1) {

9 ok = **true**;

10 }

11 **return** ok;

}



n = 5

T (n) = 3n+6

Analisis linea 3

i = 2, i = 3, i = 4, i = 5, i = 6 (Sale)

Sumando las veces i se le asigna un valor nos da 5

n = 5 (Veces que se le asigno vaolor a i)

***Análisis de complejidad espacial***

primeNumbersVerification2(**int** n)

|  |  |  |
| --- | --- | --- |
| Tipo | Variable | Veces |
| Entrada | n | 1 |
| Auxiliar | i | 1 |
| Salida | aux | 1 |

Complejidad: 3

primeNumbersVerification3(**int** n)

|  |  |  |
| --- | --- | --- |
| Tipo | Variable | Veces |
| Entrada | n | 1 |
| Auxiliar | i | 1 |
| Salida | aux | 1 |

Complejidad: 3

Validation1Prime(**int** numberToValidate)

|  |  |  |
| --- | --- | --- |
| Tipo | Variable | Veces |
| Entrada | numberToValidate | 1 |
| Auxiliar | counter | 1 |
| Auxiliar | i | 1 |
| Salida | ok | 1 |

Complejidad: 4

***Anexos***

Fuentes de recopilación:

<https://es.wikipedia.org/wiki/N%C3%BAmero_natural>

<http://geogebra.es/gauss/materiales_didacticos/primaria/actividades/aritmetica/naturales_y_enteros/criba_de_eratostenes/actividad.html>

<https://www.youtube.com/watch?v=GST7EhThqpQ>

<https://www.educapeques.com/recursos-para-el-aula/fichas-de-matematicas-y-numeros/numeros-primos.html>

<https://brainly.lat/tarea/3450780>