**Max-Mean Dispersion Problem**

Diseño y Análisis de Algoritmos, Universidad de La Laguna

**INDICE**

1. Introducción ……………………………………………………………………………….. [Página 2](#Introduccion)
2. Estructura de clases ……………………………………………………………………. [Página 3](#EstructuraClases)
   1. Class FrameWork ………………………………………………………….  [Página 4](#FrameWork)
   2. Class Vertex ………………………………………………………………….. [Página 6](#Vertex)
   3. Class Edge …………………………………………………………………….. [Página 7](#Edge)
   4. Class Graph …………………………………………………………………... [Página 9](#Graph)
   5. Class Chrono …………………………………………………………………. [Página 11](#Chrono)
   6. Class Algorithm ……………………………………………………………… Página --
   7. Class GreedyAlgorithm ………………………………………………….. Página --
   8. Class AnotherGreedyAlgorithm ……………………………………… Página --
   9. Class GraspAlgorithm …………………………………………………….. Página --
   10. Class MultiBootAlgorithm ………………………………………………. Página --
   11. Class VNSAlgorithm ………………………………………………………… Página --
3. **Introducción**

**¿En qué consiste el Max-Mean Dispersion Problem?**

El Max-Mean Dispersion Problem es una variante de un modelo de optimización clásico en el contexto de maximizar la diversidad de un conjunto de elementos. En particular este problema del cálculo de la dispersión media máxima modela varios problemas reales, como el control de la contaminación o la clasificación de una página web.

**Objetivo de la práctica**

Proponer, implementar y evaluar algoritmos constructivos y búsquedas por entornos para el Max-Mean Dispersion Problem.

**Max-mean Dispersion Problem**

Sea dado un grafo completo G = (V, E), donde V es el conjunto de vértices (|V| = n) y E es el conjunto de aristas (|E| = n(n − 1)/2. Cada arista (i, j) ∈ E tiene asociada una distancia o afinidad d(i, j). En el max-mean dispersion problem se desea encontrar el subconjunto de vértices S ⊆ V que maximiza la dispersión media dada por

1. **Estructura de clases:**

En este apartado se definirán las clases empleadas para la realización de la práctica, en ellas se mencionarán y definirán brevemente los métodos y los atributos de las mismas. Para ello emplearemos la notación de UML:

+ : Expresa que el atributo/método es público.

- : Expresa que el atributo/método es privado.

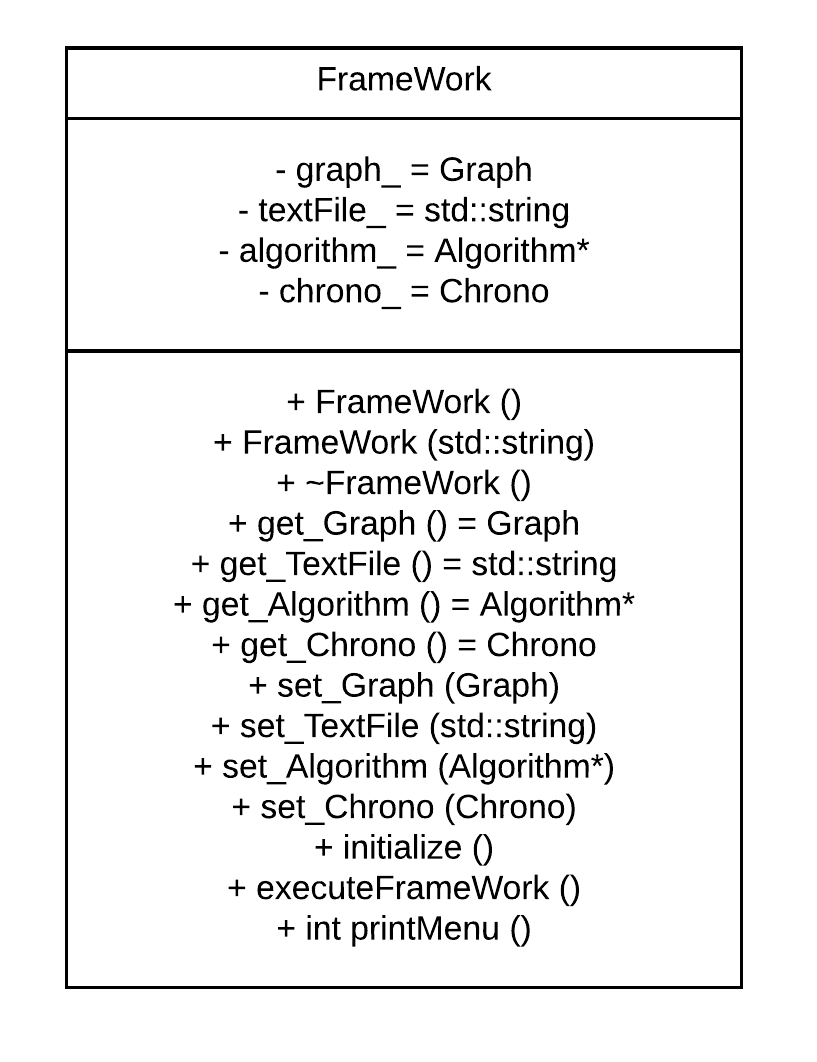
# : Expresa que el atributo/método es protegido.

Para cada clase se hará una pequeña introducción de la misma seguida de una imagen con su respectivo diagrama en UML. A continuación, se definirán sus atributos y métodos. En el caso de las clases relacionadas con los tipos de algoritmos que buscan la solución (Greedy, Grasp, MultiBoost y VNP), además, al final de la misma estará el pseudocódigo de dicho algoritmo.

**2.1 Class FrameWork**

En esta clase almacenamos el grafo generado por los datos del fichero de entrada y posteriormente se le pedirá al usuario que elija que algoritmo desea implementar para la resolución del mismo.

**Diagrama UML de la clase FrameWork**



**Atributos de la clase FrameWork**

- graph\_: Objeto de la clase *Graph* en donde almacenaremos los datos de entrada.

-textFile\_: Cadena de caracteres que contendrá el nombre del fichero de entrada con los datos iniciales.

- algorithm\_: Puntero a un objeto de la clase Algorithm que será el encargado de ejecutar el algoritmo para calcular la solución al problema.

- chorno\_: Objeto de la clase *Chrono* que se encargará de medir el tiempo que tarda la ejecución del algoritmo.

**Métodos de la clase FrameWork**

+ Framework (): Constructor por defecto de la clase.

+ Framework (string textfile): Constructor de la clase al que recibe y añade el nombre del fichero de entrada a los atributos del grafo.

+ ~Framework (): Destruye el objeto framework.

+ Graph get\_Graph (): Método que devuelve el atributo *graph\_.*

+ get\_TextFile (): Método que devuelve el atributo *textFile*\_.

+ get\_Algorithm (): Método que devuelve el atributo *algorithm*\_.

+ get\_Chrono (): Método que devuelve el atributo *chrono*\_.

+ set\_Graph (Graph graph): Método que establece el atributo *graph*\_.

+ set\_TextFile (string textFile): Método que establece el atributo *textFile*\_.

+ set\_Algorithm (Algorithm\* algorithm): Método que establece el atributo *algorithm*\_.

+ set\_Chrono (Chrono chrono): Método que establece el atributo *chorno*\_.

+ initialize (): Método que inicializa el objeto FrameWork, inicializa el grafo y pide al usuario que indique que clase de algoritmo desea ejecutar para la resolución del problema.

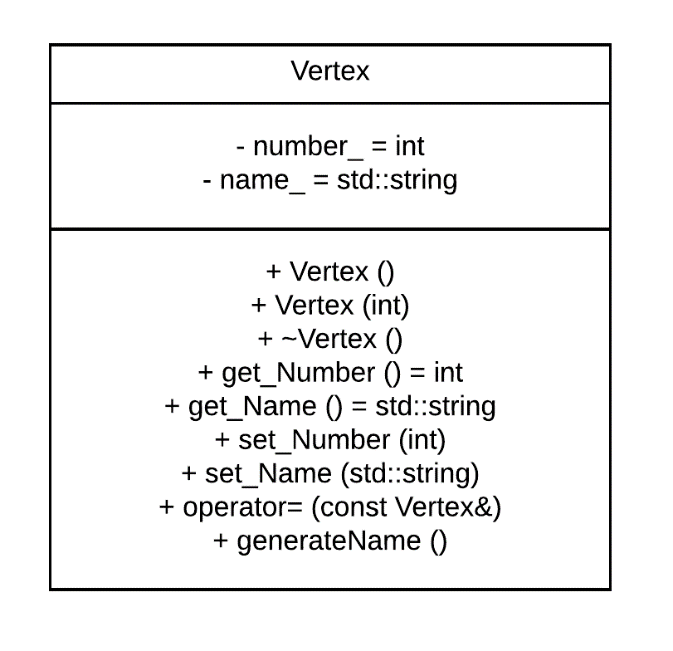
+ executeFrameWork (): Método que se encarga de llamar a la ejecución del algoritmo, contabilizar el tiempo y llamar a la impresión del tiempo y los resultados del algoritmo

+ printMenu (): Método que imprime un menú en el que se le solicita al usuario la selección del algoritmo que se desea emplear.

**2.2 Class Vertex**

Esta clase se emplea para almacenar un vértice con su nombre y numeración con la que será tratado en los algoritmos. En el caso concreto de nuestra práctica se creará un vector de vértices donde se almacenarán todos los vértices correspondientes a un determinado grafo.

**Diagrama UML de la clase Vertex**



**Atributos de la clase Vertex**

- number\_: Número que se le asigna al vértice, su identificador en el programa.

- name\_: Nombre del vértice en caso de tenerlo, en caso de no tener nombre se le asignaría como "Vertex X" donde X es el número identificatorio del vértice.

**Métodos de la clase Vertex**

+ Vertex (): Constructor por defecto del vértice.

+ Vertex (int number): Constructor al que se le pasa como parámetro el número que identifica al vértice.

+ ~Vertex (): Destruye el objeto Vertex.

+ get\_Number (): Método que devuelve el atributo *number*\_.

+ get\_Name (): Método que devuelve el atributo *name*\_.

+ set\_Number (int number): Método que establece el atributo *number*\_.

+ set\_Name (string name): Método que establece el atributo *name*\_.

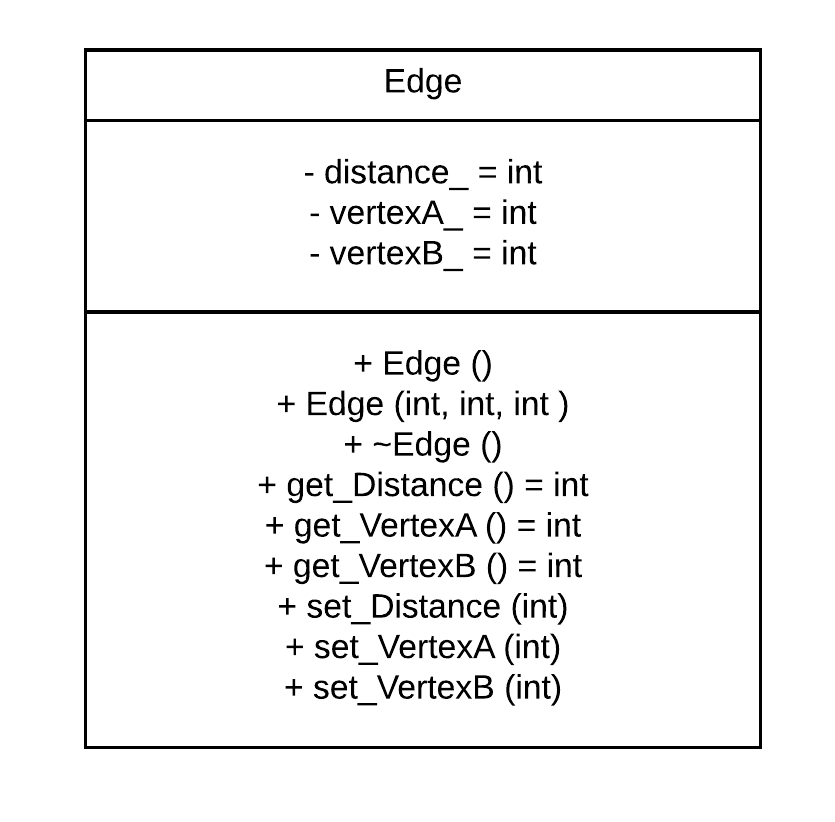
+ operator= (const Vertex& vertex): Sobrecarga del operador de asignación.

+ generateName (): Genera el nombre por defecto del vértice.

**2.3 Class Edge**

Esta clase se emplea para almacenar una arista, una arista es la línea que une un vértice con otro y tiene un valor como puede ser la distancia, coste, etc... En el caso particular de nuestra práctica emplearemos esta clase para almacenar todas las aristas pertenecientes a un determinado grafo.

**Diagrama UML de la clase Edge**



**Atributos de la clase Edge**

- distance\_: Numero que almacena el coste en trasladarse del vértice A al vértice B.

- vertexA\_: Número identificatorio del vértice de partida de la arista.

- vertexB\_: Número identificatorio del vértice final de la arista.

**Métodos de la clase Edge**

+ Edge (): Constructor por defecto de la clase arista.

+ Edge (int vertexA, int vertexB, int distance): Constructor de la clase arista a la que se le pasa el vértice de partida, el final y el coste de la arista.

+ ~Edge (): Destructor del objeto arista.

+ get\_Distance (): Método que devuelve el atributo *distance*\_.

+ get\_VertexA (): Método que devuelve el atributo *vertexA*\_.

+ get\_VertexB (): Método que devuelve el atributo *vertexB\_*.

+ set\_Distance (int distance): Método que establece el atributo *distance*\_.

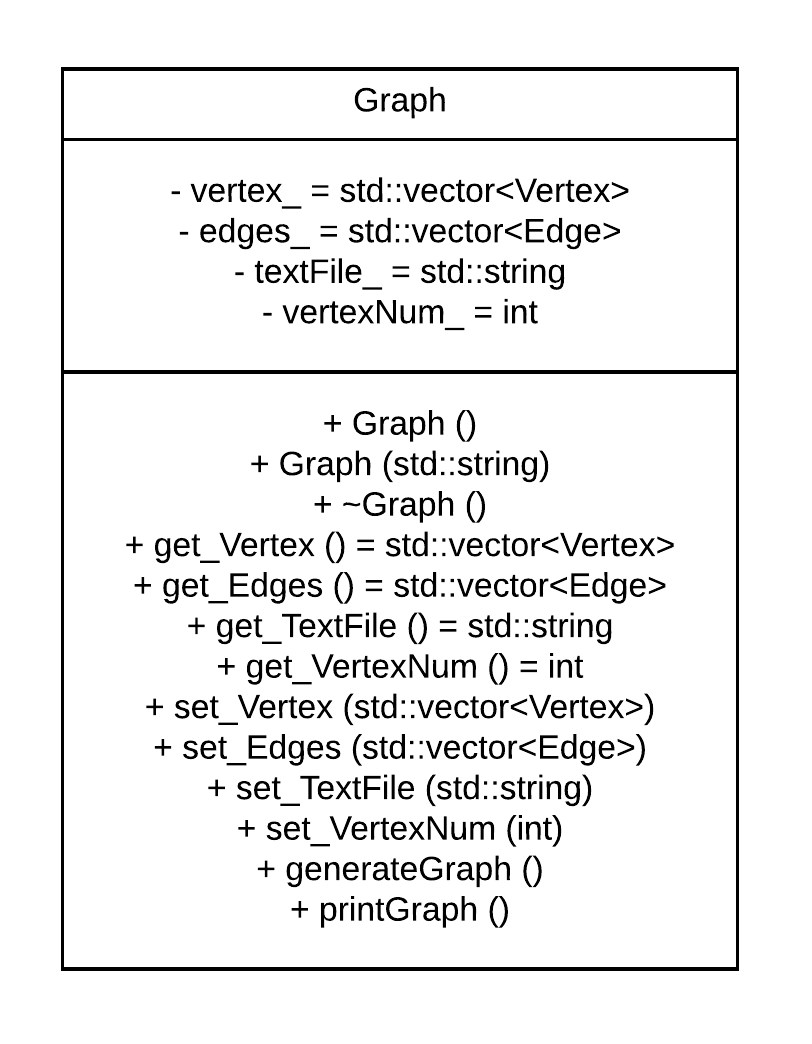
+ set\_VertexA (int vertexA): Método que establece el atributo *vertexA*\_.

+ set\_VertexB (int vertexB): Método que establece el atributo *vertexB\_*.

**2.4 Class Graph:**

Esta clase almacena los datos de un grafo, conjunto de vértices y aristas sobre los que se ejecutarán los algoritmos de búsqueda de la mejor solución.

**Diagrama UML de la clase Graph**



**Atributos de la clase Graph**

- vertex\_: Vector de vértices donde se almacenarán todos los vértices pertenecientes al grafo.

- edges\_: Vector de aristas donde se almacenarán todas las aristas pertenecientes al grafo.

- textFile\_: Nombre del fichero de entrada del que se leen los datos para generar el grafo.

- vertexNum\_: Número de vértices que posee el grafo.

**Métodos de la clase Graph**

+ Graph (): Constructor por defecto de la clase arista.

+ Graph (string textFile): Constructor de la clase que recibe como parámetro el nombre del fichero de entrada en el que se encuentran los datos del grafo.

+ ~Graph (): Destructor de la clase Graph.

+ get\_Vertex (): Método de la clase que devuelve el vector de vértices vertex\_.

+ get\_Edges (): Método de la clase que devuelve el vector de aristas edges\_.

+ get\_TextFile (): Método de la clase que devuelve el atributo textFile\_.

+ get\_VertexNum (): Método de la clase que devuelve el atributo vertexNum\_.

+ set\_Vertex (vector<Vertex> vertex): Método de la clase que establece el vector de vértices vertex\_.

+ set\_Edges (vector<Edge> edge): Método de la clase que establece el vector de aristas edges\_.

+ set\_TextFile (string textFile): Método de la clase que establece el atributo textFile\_.

+ set\_VertexNum (int vertexNum): Método de la clase que establece el atributo vertexNum\_.

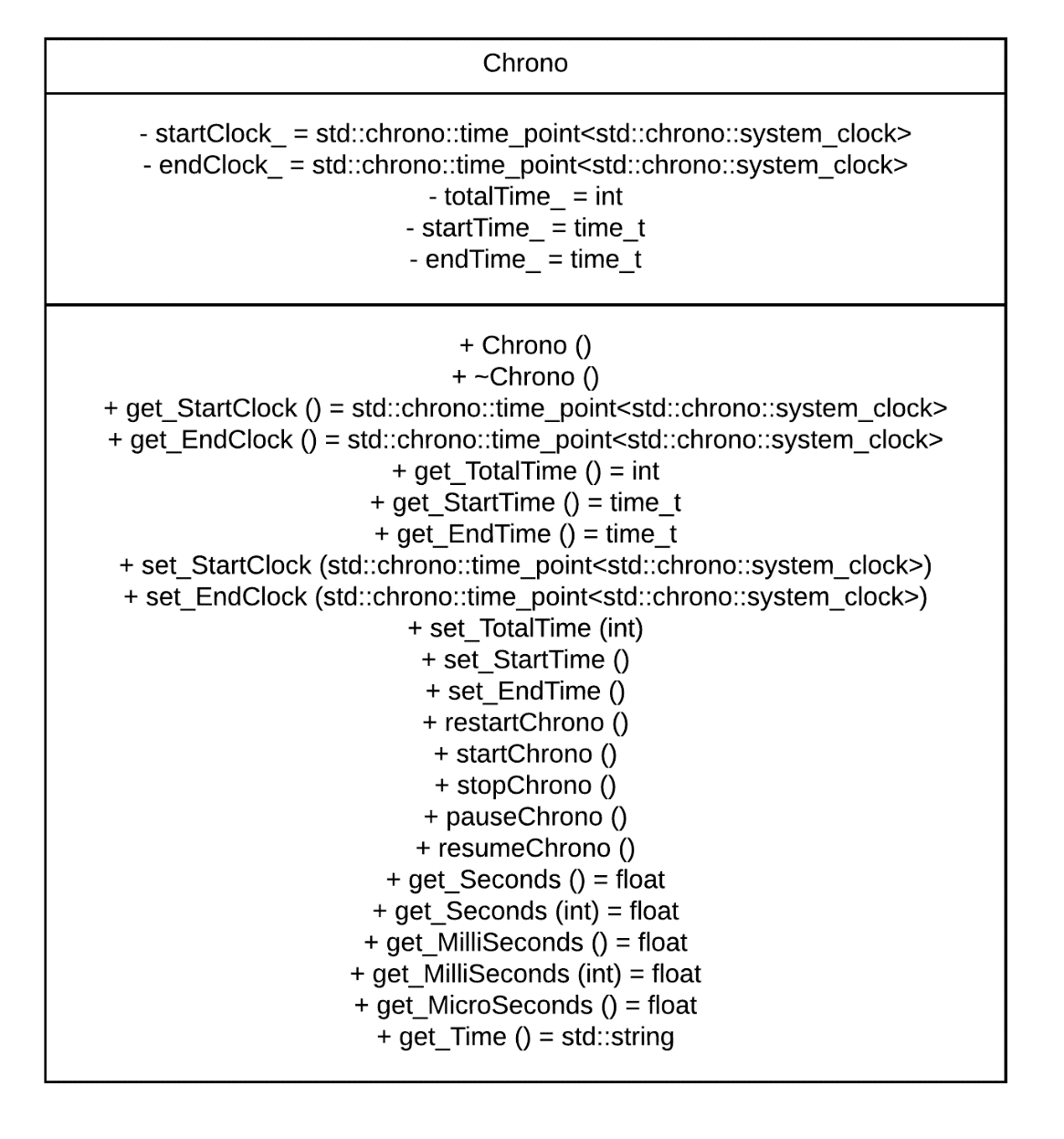
+ generateGraph (): Genera el grafo leyendo los datos del fichero de entrada.

+ printGraph (): Imprime el grafo por pantalla.

**2.5 Class Chrono**

Esta clase es empleada para crear un reloj en el programa y que este nos cronometre o calcule el tiempo que dura la ejecución de nuestros algoritmos. El cronómetro será inicializado antes de que empiece la ejecución del algoritmo y parado cuando acabe.

**Diagrama UML de la clase Chrono**



**Atributos de la clase Chrono**

- startClock\_: Objeto de la librería *std::chrono* que almacenará el momento en que se inicia el programa.

- endClock\_: Objeto de la librería *std::chrono* que almacenará el momento en que se finaliza el programa.

- totalTime\_: Número de microsegundos transcurridos en ejecución.

- startTime\_: Fecha y hora de comienzo.

- endTime\_: Fecha y hora de finalización.

**Métodos de la clase Chrono**

+ Chrono (): Constructor por defecto de un objeto Chrono.

+ ~Chrono (): Destruye el objeto Chrono.

+ get\_StartClock (): Devuelve el atributo *startClock*\_.

+ get\_EndClock (): Devuelve el atributo *endClock*\_.

+ get\_TotalTime (): Devuelve el atributo *totalTime*\_.

+ get\_StartTime (): Devuelve el atributo *startTime*\_.

+ get\_EndTime (): Devuelve el atributo *endTime*\_.

+ set\_StartClock (startClock): Establece el atributo *startClock*\_.

+ set\_EndClock (endClock): Establece el atributo *endClock*\_.

+ set\_TotalTime (int totalTime): Establece el atributo *totalTime*\_.

+ set\_StartTime (): Establece el atributo *startTime*\_.

+ set\_EndTime (): Establece el atributo *endTime*\_.

+ restartChrono (): Resetea el objeto Chrono.

+ startChrono (): Empieza a contabilizar el tiempo.

+ stopChrono (): Para el Chrono y establece el tiempo total transcurrido.

+ pauseChrono (): Pausa el Chrono y añade el tiempo total transcurrido al atributo *totalTime*\_.

+ resumeChrono (): Continúa la ejecución del Chrono en caso de que hubiese sido pausado.

+ get\_Seconds (): Devuelve el tiempo en segundos.

+ get\_Seconds (int decimalAmmount): Devuelve el tiempo en segundos con X cifras decimales donde X es valor pasado como parámetro.

+ get\_MilliSeconds (): Devuelve el tiempo en milisegundos.

+ get\_MilliSeconds (int decimalAmmount): Devuelve el tiempo en milisegundos con X cifras decimales donde X es valor pasado como parámetro.

+ get\_MicroSeconds (): Devuelve el tiempo en microsegundos.

+ get\_Time (): Devuelve el tiempo en horas, minutos y segundos (Para operaciones largas).