***T.P. N° 3***

# Introducción

La finalidad de este trabajo es el procesamiento de funciones, dadas en notación polaca inversa. Para esto, se disponen de diversas herramientas como lo son los patrones de diseño Singleton y Composite.

Se analizará en este trabajo como emplear dichas herramientas de la mejor forma posible para solucionar la problemática planteada (Funciones dadas en Notación Polaca Inversa).

Se ha mencionado la notación polaca inversa, la cual:

“*...método algebraico alternativo de introducción de datos.*”, Según Wikipedia.

“*... En ciencias de la computación, la notación de postfijo es frecuentemente usada en lenguajes de programación concatenativos y basados en pila.* “, Según Wikipedia

# Documentación de clases e interfaces

**Interfaz: Funcion.**

Interfaz que utilizan la mayoría de las clases para combinarse como una única función.

Método:

**public double calcular()**. Calcula el valor de una operación o valor y retorna el resultado.

**Clase: Archivero**

Esta clase es utilizada para leer los archivos de entrada, escribir los archivos de salida, y almacenar las entradas en los objetos correspondientes.

Metodos:

**public static double[ ][ ] leer(String ruta, FuncionCompuesta fn).** Lee un archivo alojado en “ruta” y guarda la función de entrada en “fn”.

**public static void escribir(String ruta, double[] res).** Escribe un archivo de salida alojado en “ruta” con los valores resultado de los puntos evaluados.

**Clase: Funcion compuesta.**

Esta clase es utilizada para manejar el armado de las funciones, utilizando un COMPOSITE para ello.

Atributos:

* func : LinkedList<Funcion>. Lista utilizada para apilar las funciones del composite.

Métodos:

**public void agregar(Funcion f).** Agrega una función al composite de funciones.

**public Funcion remover().** Quita la última función agregada al composite y la retorna.

**public double calcular()**. Calcula el valor de las funciones del composite y devuelve su valor.

**Clase: División**

Se encarga de las operaciones de división y el manejo de los resultados.

Atributos:

* valor1: Funcion. Dividendo.
* valor2: Funcion. Divisor.

Métodos:

**public Division (Funcion valor1, Funcion valor2).** Constructor que recibe el dividendo (valor1) y el divisor (valor2).

**public double calcular().** Calcula el cociente de una división y lo retorna.

**public String toString().** Retorna la operación con sus valores en forma de cadena de caracteres. No incluye el cociente.

**Clase: Logaritmo**

Se encarga de las operaciones logaritmo natural y el manejo de sus resultados.

Atributos:

* valor: Funcion. Argumento del logaritmo.

Métodos:

**public Logaritmo(Funcion valor).** Constructor que recibe el argumento del logaritmo(valor).

**public double calcular().** Calcula el valor del logaritmo natural y retorna el resultado.

**public String toString()**. Retorna la operación con sus valores en forma de cadena de caracteres. No incluye el resultado.

**Clase: Multiplicacion**

Se encarga de las operaciones de multiplicación y el manejo de los resultados.

Atributos:

* valor1: Funcion. Operando.
* valor2: Funcion. Operando.

Metodos:

**public Multiplicacion(Funcion valor1, Funcion valor2).** Contructor que recibe ambos operandos del producto (valor1 y valor2).

**public double calcular().** Calcula el producto y devuelve el resultado.

**public String toString()**. Devuelve la operación con sus valores en forma de cadena de caracteres. No incluye el resultado.

**Clase: Potencia**

Se encarga de las operaciones de potencia y el manejo de los resultados.

Atributos:

* valor1: Funcion. Base.
* valor2: Funcion. Exponente.

Métodos:

**public Potencia(Funcion valor1, Funcion valor2).** Constructor que recibe la base de la potencia (valor1) y el exponente (valor2).

**public double calcular().** Calcula la potencia y retorna el resultado.

**public String toString().** Devuelve la operación con sus valores en forma de cadena de caracteres. No incluye el resultado.

**Clase: Resta**

Se encarga de las operaciones de resta y el manejo de los resultados.

Atributos:

* valor1: Funcion. Operando
* valor2: Funcion. Operando.

Métodos:

**public Resta(Funcion valor1, Funcion valor2).** Constructor que recibe los operandos de la resta (valor1 y valor2).

**public double calcular().** Calcula la resta y retorna el resultado.

**public String toString().** Devuelve la operación con sus valores en forma de cadena de caracteres. No incluye el resultado.

**Clase: Suma**

Se encarga de las operaciones suma y el manejo de los resultados.

Atributos:

* valor1: Funcion. Operando
* valor2: Funcion. Operando.

Métodos:

**public Suma(Funcion valor1, Funcion valor2).** Constructor que recibe los operandos de la suma (valor1 y valor2).

**public double calcular().** Calcula la suma y retorna el resultado.

**public String toString().** Devuelve la operación con sus valores en forma de cadena de caracteres. No incluye el resultado.

**Clase: Incógnita**

Se encarga de la utilización de incógnitas en la función y de su reemplazo por un valor numérico.

Atributos:

* indice: int. Valor que indica qué incógnita es la almacenada. 0 para X, 1 para Y, 2 para Z.

Métodos:

**public Incognita(String nombre).** Constructor que recibe el caracter incognita y en base a él pone el valor correcto en “indice”.

**public double calcular().** Devuelve el valor de la incógnita al evaluar la función en un punto determinado.

**public String toString().** Devuelve el caracter incógnita.

**Clase: ListaIncognitas**

Se encarga de asistir a la clase Incognita en el manejo de los caracteres incognita y sus valores numéricos. Funciona como singleton.

* lista: ListaIncognitas. Unica instancia de la clase.
* valor: double [ ] [ ]. vector que almacena los valores numéricos de las incógnitas.
* nombre: String [ ][ ]. vector que almacena los caracteres incógnita.

Métodos:

**private ListaIncognitas(int i).** Constructor que recibe la cantidad de incógnitas.

**public static ListaIncognitas getIncognita(int i).** Recibe el la cantidad de incógnitas (i) y devuelve la instancia de incógnitas.

**public void setValor(int i, double valor).** Recibe el indice (i) que indica la incógnita a la que le asignará el valor (valor).

**public double getValor(int i).** Devuelve el valor numérico que le corresponde a la incógnita correspondiente al índice recibido (i).

**public String getNombre(int i).** Devuelve el caracter correspondiente a la incógnita “i”.

**public static int getIndice(String n).** Devuelve el índice que le corresponde a la incógnita “n”.

**Clase: Numero**

Se encarga del manejo de términos numéricos.

Atributo:

* value: double.

Métodos:

**public Numero(double value)**. Constructor que recibe el valor numérico (value) a almacenar.

**public double calcular()**. Devuelve el valor numérico almacenado en la instancia.

**public String toString()**. Devuelve el valor numérico almacenado en la instancia convertido a caracter.

Conclusión

Finalmente, se definió que la mejor manera de llevar a cabo la resolución de la problemática planteada es emplear Composite para los operadores (unarios y binarios) y para los operandos constantes. Por otro lado, se decidió emplear Singleton para las incógnitas ya que al darle un valor a X, Y o Z, este valor se usa en todas las menciones de dichas variables, en la función.