****

**Universidad Nacional de General Sarmiento**

**Programación III**

**Trabajo Práctico: Calculadora**

**Profesores:**

* Patricia Bagnes.
* Javier Marenco.

**Grupo:**

* Andrés Geiser
  + 2018/42239150
  + geiserandres00@gmail.com
* Hernán Rodrigo Fernández
  + 2017/41584220
  + fernandezhr\_1998@hotmail.com

**Objetivo**

El objetivo que planteamos resolver en este trabajo práctico es implementar una aplicación que simule una calculadora simple de bolsillo, separando el código responsable de la lógica de la calculadora del código encargado de la interfaz (implementada con Swing).

**Responsabilidad de clases:**

El proyecto cuenta con 3 clases

* **GuiCalculadora** se encarga de modelar la interfaz de la aplicación, la cual esta implementada bajo el modelo de diseño form and control.
* **Calculadora** se encarga del funcionamiento de la lógica de la aplicación.
* **CalculadoraTest** contiene la suite de test correspondientes a la lógica.

**Implementación de la clase Calculadora**

**Variables de clase**

* **ArrayList<Double> numeros** contendrá la secuencia de números que ingresaran mediante el método *obtenerNumero* que se detallara más adelante.
* **ArrayList<String> signos** al igual que el array de números esta contendrá una secuencia, pero de signos (+, -, \* o /) que se obtendrán a través del método *obtenerOperador*.
* **double resultado** contendrá el resultado final que se obtendrá del método *calcular( ).*

**Métodos de clase**

A continuación, veremos la descripción de los métodos más importantes de esta clase:

* ***void obtenerNumero(double numero):*** Se encarga de agregar el double recibido a ***numeros***. Este método tiene una condición, antes de agregar, verifica si por última vez la calculadora recibió un signo, de ser verdadero agrega al array el double recibido, en caso contrario (es decir, que por última vez la calculadora recibió un numero) el último número agregado es reemplazado por el recibido.
* ***void obtenerOperador(String operador):***Se encarga de agregar el String recibido a ***signos***. Este método también tiene condiciones a la hora de agregar, en primer lugar verifica si el String es válido (si es uno distinto de "+", "-", "\*", "/" tira una excepción), en segundo lugar chequea si posteriormente la calculadora recibió algún número (si el array de números contiene por lo menos un elemento), y en tercer lugar revisa si por última vez la calculadora recibió un signo, en caso de que esta condición sea verdadera el ultimo signo guardado en el array es reemplazado por el recibido y en caso de ser falso lo agrega al array normalmente.
* ***void calcular( ):*** Se encarga de resolver la operación y almacenar en ***resultado*** el valor obtenido. Antes de operar se fija si al menos el array de números tiene uno guardado (tira excepción sino), también borra el ultimo signo ingresado en caso de que por última vez se haya ingresado un signo ya que este no tendrá relevancia a la hora de resolver la operación y además evitamos futuros resultados mal calculados. Luego de pasar por las condiciones mencionadas este método pasa a resolver la operación, antes de operar con las sumas y restas, llama al método *resolverOperadoresPrimarios( )* que opera primero con las multiplicaciones y divisiones.
* ***void resolverOperadoresPrimarios( ):*** Tiene como funcionalidad resolver las operaciones de multiplicación y la división. Supongamos que tenemos los siguientes datos dentro de los arrays:

**Números**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 3 | 18 | 7 | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 3 | 18 | 7 | 1 |

0 1 2 3 4

**Signos**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 3 | 18 | 7 | 1 |

Representación:

**4 + 3 \* 18 / 7 - 1**

|  |  |  |  |
| --- | --- | --- | --- |
| + | \* | / | - |

0 1 2 3

Se recorre el array de signos hasta encontrar un *“\*”* o un **“/”** y se toma del array de números el elemento con el mismo índice del signo, luego lo multiplicamos o dividimos por el número que le sigue (en este caso seria 3\*18). Luego en la posición donde estaba el número 3 cambiamos el valor por el resultado de esa operación (54), luego removemos el signo y el número que le sigue porque ya no servirán. Los arrays quedaran así:

­­­­­

**Números**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 3 | 18 | 7 | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 54 | 18 | 7 | 1 |

0 1 2 3

**Signos**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 3 | 18 | 7 | 1 |

Representación:

**4 + 54 / 7 - 1**

|  |  |  |  |
| --- | --- | --- | --- |
| + | \* | / | - |

0 1 2

Siguiendo esta lógica dejara el array de signos solo con sumas y restas y al array de números con los resultados:

**Números**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 3 | 18 | 7 | 1 |

|  |  |  |
| --- | --- | --- |
| 4 | 3.78 | 18 |

0 1 2

**Signos**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 3 | 18 | 7 | 1 |

Representación:

**4 + 3.78 - 1**

|  |  |
| --- | --- |
| + | - |

0 1

* ***void reset( ):*** Restaura la calculadora a sus valores predeterminados, asigna el valor 0 al resultado y vacía los arreglos ***numeros*** y ***signos***.
* ***void borrar( ):*** Método encargado de eliminar el último signo agregado(si por última vez se agregó un signo) o bien el último dígito de un número(en caso de que por última vez se haya agregado un numero) a través de:
  + ***borrarUltimoSigno( ):*** Remueve el último signo agregado (si existe alguno).
  + ***borrarDigitoUltimoNumero( ):*** Remueve el último dígito del último número ingresado, o lo elimina del array si es un numero de un digito. Se trabaja con el número convertido a un String para facilitar el recorrido del mismo.
* ***float getResultado( ):***  Retorna el resultado final de la operación. Notar que se devuelve al double parseado a float para que el resultado en caso de ser decimal, sea redondeado y evite números con demasiadas cifras.

Esta clase cuenta con algunos métodos que realizan alguna operación auxiliar específica o se utilizan para verificar condiciones:

* **cantSignos( ):** retorna la cantidad de signos.
* **ultimoNumero( ):** retorna el último número ingresado.
* ***hayAlgunSigno( ):*** retorna verdadero si ***signos*** tiene al menos un elemento.
* ***ultimaVezSeIngresoSigno( )*:** retorna verdadero si el tamaño de ***numeros*** y ***signos*** es equivalente.
* ***reemplazarUltimoSigno(String operador)*:** reemplaza el último elemento de ***signos*** por el recibido.

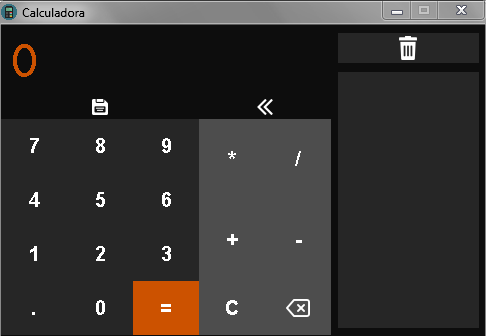
**Implementación de la clase GuiCalculadora**

No nos proponemos como objetivo explicar el funcionamiento de esta clase debido a que el código que contiene en su mayoría es el generado por Swing como es el caso de la inicialización del Jframe, el JTextField, los JPanel y los diferentes JButton con sus respectivos ActionListener.

Esta clase cuenta con una variable String llamada **numeroActual** la cual se usará para almacenar el número que el usuario formara a medida que presione los botones numéricos para posteriormente pasárselo a la lógica de la calculadora para que lo guarde.

Además de los colores de la interfaz, la ubicación de los JComponents dentro de la ventana y la organización del código generado por Swing, donde intervenimos especialmente fue en el ActionListener de cada botón tanto para que **numeroActual** no quede con un formato invalido (que tenga signos entre medio del número **“44\*6”,** que tenga varios puntos **“51…666.”**, o situaciones similares que al momento de convertir de String a double no arroje ningún tipo de error) al momento de pasárselo a la lógica de la calculadora como también para que en el JTextField se muestre la operación correctamente y no con una forma extraña como podría ser **“4 + - 56.5 / ...\* + 000054**” que pueda confundir al usuario o hacer que el código de negocio genere algún tipo de error.

Esta clase también tiene métodos auxiliares que nos ayudaron a cumplir lo mencionado anteriormente con más facilidad y tener un código mejor organizado, que especialmente en esta clase que está caracterizada por poseer bastante código es de mucha ayuda al momento del entendimiento y la claridad.

****

Elimina todos los resultados guardados

Elimina todos los resultados guardados

Elimina todos los resultados guardados

Elimina todos los resultados guardados

Elimina todos los resultados guardados

**Problemas encontrados y soluciones**

Resetea la calculadora

Borra el ultimo numero o signo

Panel donde se guardarán los resultados

Botón que gurda los resultados

Muestra o oculta el panel de resultado s

Durante el desarrollo de la aplicación surgieron algunos inconvenientes que tuvimos que afrontar:

* Cuando probamos el funcionamiento de la calculadora, notamos que algunas operaciones con números decimales arrojaban más dígitos de lo esperado, lo que quitaba claridad al resultado final. Frente a esto nos propusimos cambiar el tipo de retorno de *getResultado( )* de double a float, lo que permite que el resultado final se redondee a menos cifras.
* Una vez solucionado el problema anterior notamos que las operaciones que combinaban signos no arrojaban el resultado correcto. Detectamos que la lógica de nuestra clase Calculadora resolvía operaciones de corrido sin tener en cuenta el orden de los operadores, lo que cambiaba drásticamente el resultado, por lo que decidimos hacer un cambio crítico en la forma en la que se procesaban las operaciones modificando completamente el método *calcular* y agregando *resolverOperadoresPrimarios* que fue explicado con anterioridad.
* Hacia el final del desarrollo de la aplicación, nos propusimos agregarle “memoria” a la calculadora con el fin de poder almacenar tantos resultados como se deseen, y fue aquí donde decidimos implementarlo dentro de la clase encargada de la interfaz donde un Jpanel sería quien cumpla esta función. Esto se debe a que esta implementación no requiere manejar otra estructura de datos específica para poder mantener la lista y toda la lógica que conlleva.