DS - Práctica de Diseño: Memoria

Pablo Aragunde Canabal
Pablo Costa Oubiña
Grupo 3.4

1. Introducción.

En esta práctica se nos pide diseñar una calculadora que además debe ser flexible y fácilmente ampliable. Para ello comenzamos por modelar la calculadora como una máquina de estados, ya que el funcionamiento de la misma dependerá de su estado, es decir, de lo que haya sucedido anteriormente.

A continuación dividimos la calculadora en diferentes partes o módulos independientes de forma que nos permitiera abordar cada problema de forma aislada. En nuestro diseño estos módulos son: las operaciones (unarias por un lado y binarias por otro), la memoria de la calculadora (en la que podemos guardar un número para utilizarlo posteriormente) y los estados internos de la calculadora.

2. Diseño: principios y patrones.

En este apartado expondremos qué patrones de diseño hemos utilizado en cada caso y el porqué de esa decisión, así como las adaptaciones que hemos hecho para que nuestro diseño se ciñera a los principios fundamentales.

El patrón principal en el que se basa el diseño de nuestra calculadora es el Patrón Estado. Este patrón nos permite llevar a cabo nuestro modelaje de la calculadora como máquina finita de una forma sencilla, aislando el comportamiento de cada uno de los estados, definiendo de manera clara las transiciones y permitiéndonos añadir nuevos estados fácilmente en caso de que fuera necesario.

También utilizamos el patrón estado para diseñar la memoria interna de nuestra calculadora de forma que si en un futuro se quiere aumentar la complejidad de esta memoria será mucho más sencillo con este diseño.

Para diseñar las operaciones tomamos la decisión de mantener separadas las operaciones binarias y las operaciones unarias (principio de segregación de interfaces), pero su diseño es esencialmente el mismo: sigue el patrón estrategia, de forma que podremos añadir nuevas operaciones sin modificar las ya existentes (principio abierto-cerrado), así como realizar una operación de un tipo determinado, sin saber explícitamente de cuál se trata.

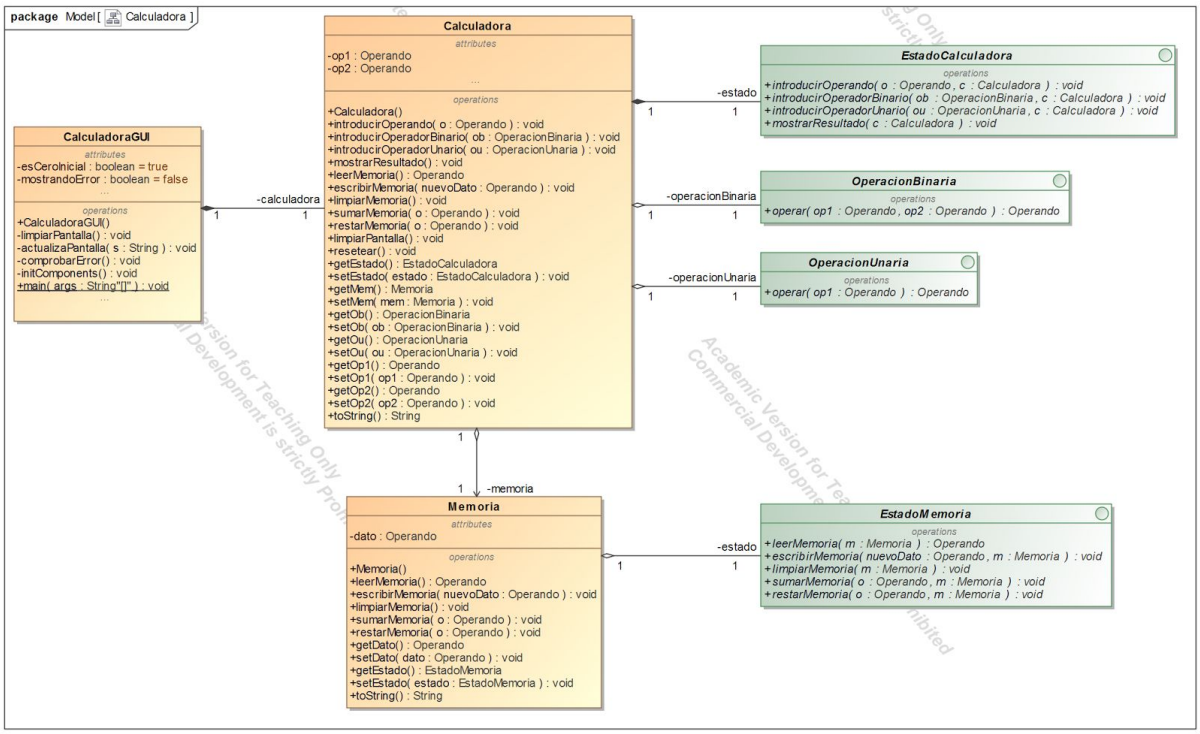
Por último hemos utilizado el patrón adaptador para que nuestras clases utilicen “Operando” y no “BigDecimal”. Todas nuestras clases (calculadora, operaciones concretas, etc.) trabajarán con Operando y es Operando quien trabaja con BigDecimal, de forma que si queremos cambiar BigDecimal por otra clase, sólo tengamos que cambiar la implementación de Operando y no de todas y cada una de las clases que utilizan Operando.

3. Diagramas.

3.1. Diagramas de clases.

A la hora de realizar los diagramas de clases hemos decidido representar las clases en varios niveles de detalle: de más general a más concreto.

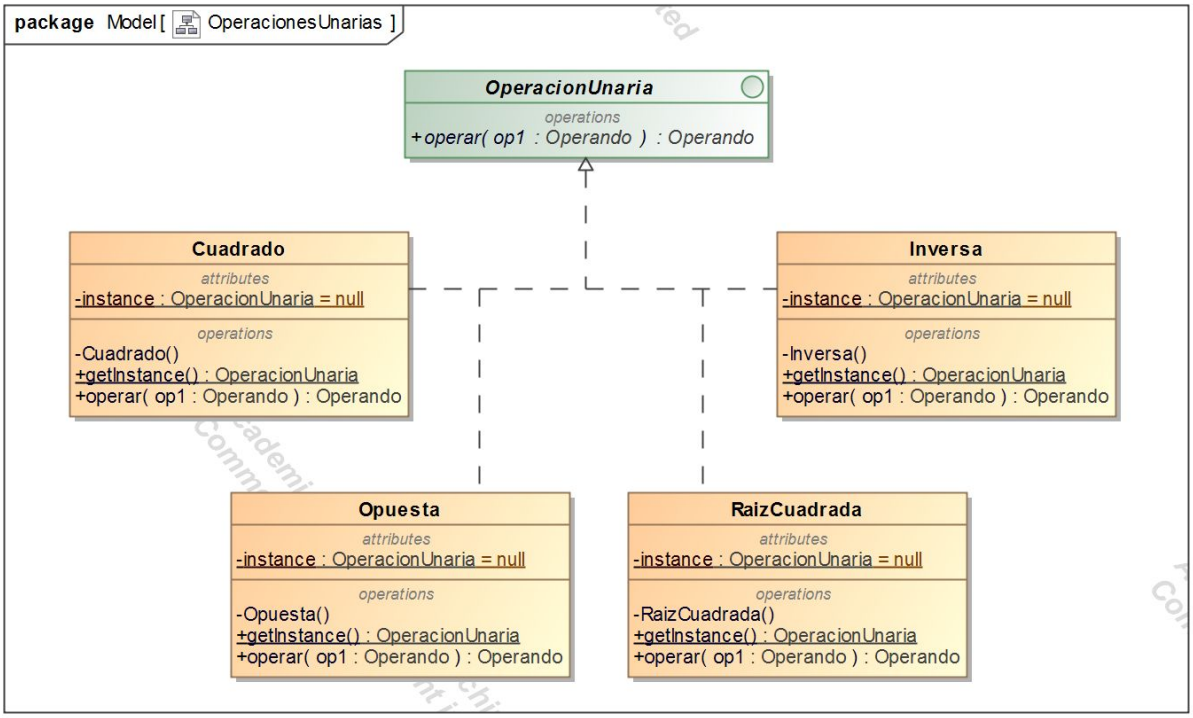
En primer lugar presentamos el diagrama de la estructura general del proyecto. La vista (CalculadoraGUI) utiliza una instancia de Calculadora realizando las operaciones necesarias en cada uno de los eventos. La calculadora está compuesta de cuatro elementos principales: memoria, operación unaria, operación binaria, y el estado de la calculadora, además de los dos operandos.



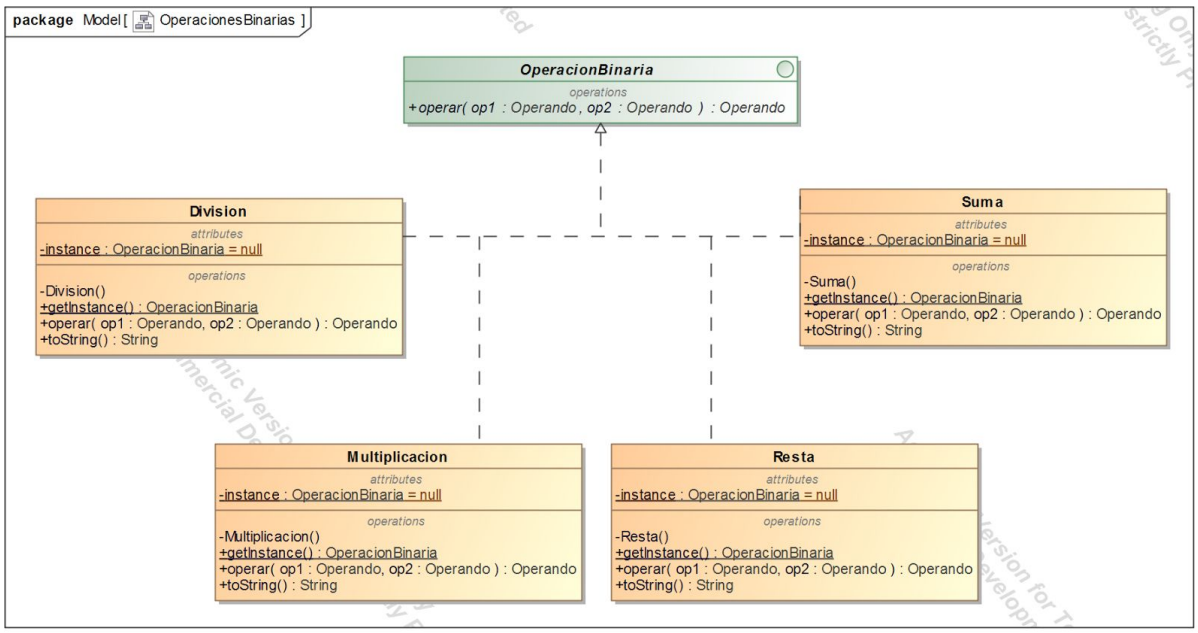
(fig. 1 - Diagrama Clases Calculadora)

Nuestra calculadora cumple con el principio de inversión de la dependencia, dado que trabaja con abstracciones (los interfaces OperacionUnaria y OperacionBinaria) y no con implementaciones concretas de estos interfaces.

Las interfaces OperacionUnaria y OperacionBinaria siguen el patrón estrategia y juegan el rol de Estrategia, mientras que Calculadora juega el rol de contexto y cada una de las implementaciones (Suma, Multiplicación, etc., para Operaciones Binarias. Cuadrado, Inversa, etc., para Operaciones Unarias) juega el rol de Estrategia Concreta.

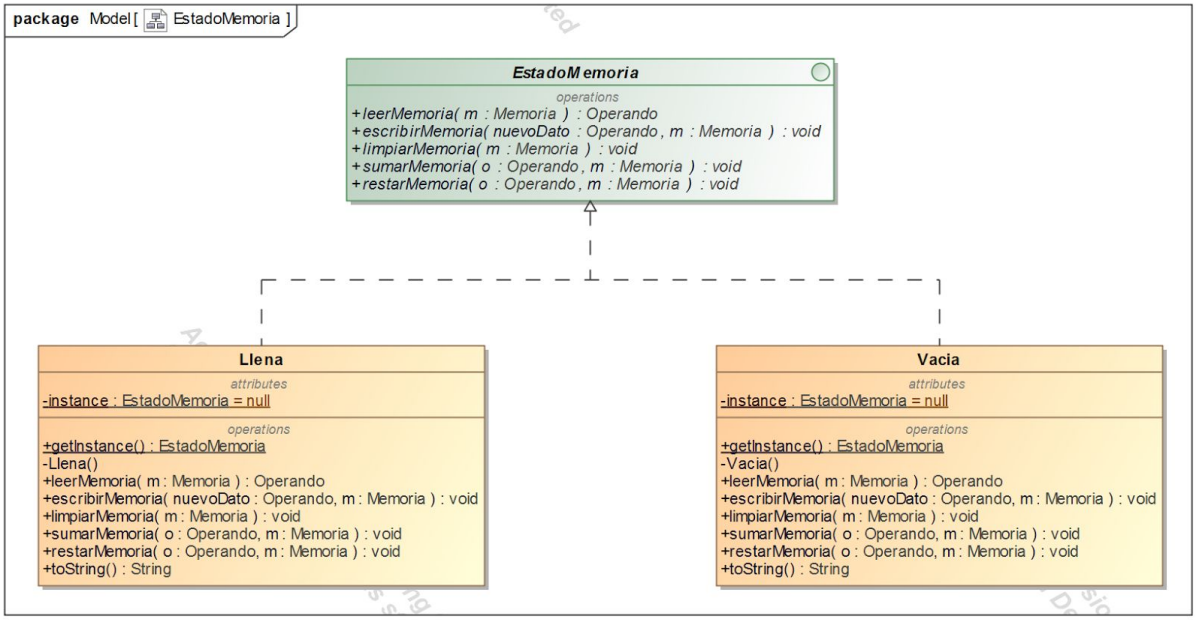


(fig. 2 - Diagrama Clases OperacionUnaria)



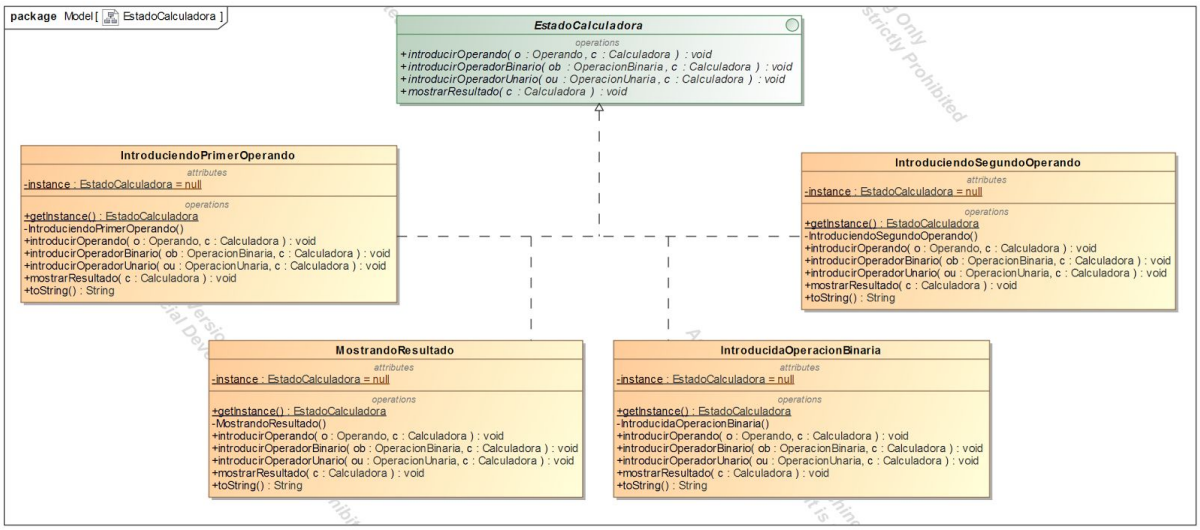
(fig. 3 - Diagrama Clases OperacionBinaria)

La memoria de la calculadora la hemos diseñado siguiendo el patrón estado, i.e., puede tener dos estados: vacía (estado inicial) y llena. Cada uno de estos estados implementa las operaciones pertinentes (escribir memoria, leer memoria, etc.)



(fig. 4 - Diagrama Clases EstadoMemoria)

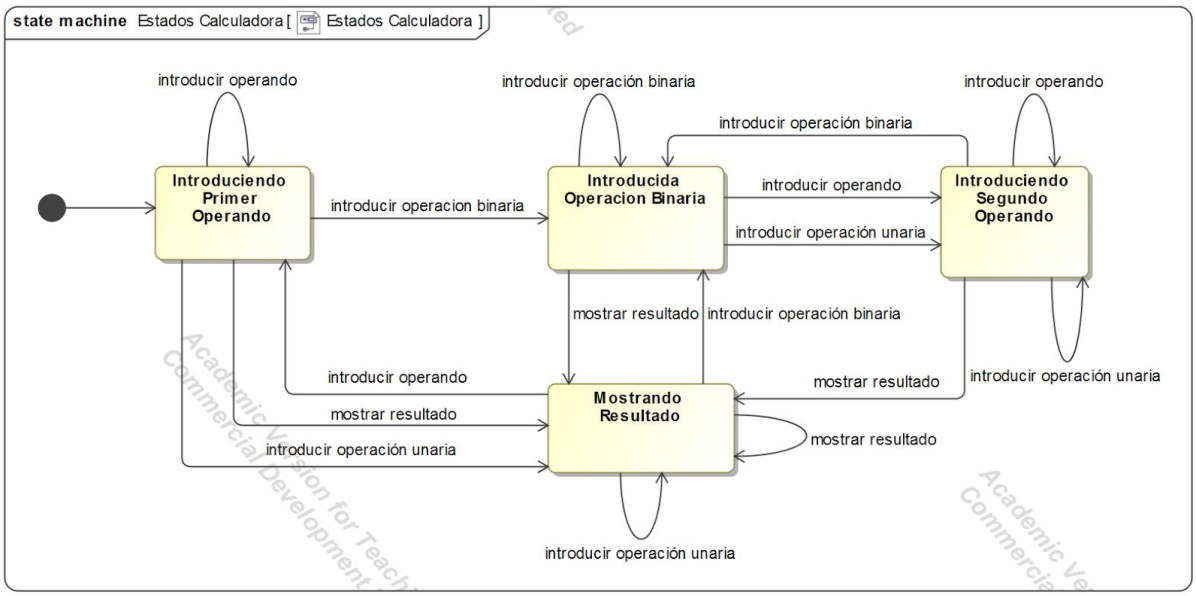
Finalmente, hemos diseñado el estado interno de la calculadora siguiendo el patrón estado de forma que la clase Calculadora juega el rol de contexto, EstadoCalculadora el rol de estado y cada uno de las implementaciones (Mostrando Resultado, Introducida Operación Binaria, etc.) juegan el rol de estado concreto.



(fig. 5 - Diagrama Clases EstadoCalculadora)

3.2. Diagramas de estados.

Ya que hemos seguido el patrón estado a la hora de diseñar la calculadora, consideramos oportuno proporcionar un diagrama de estados en el que representamos cada estado así como las transiciones entre ellos.



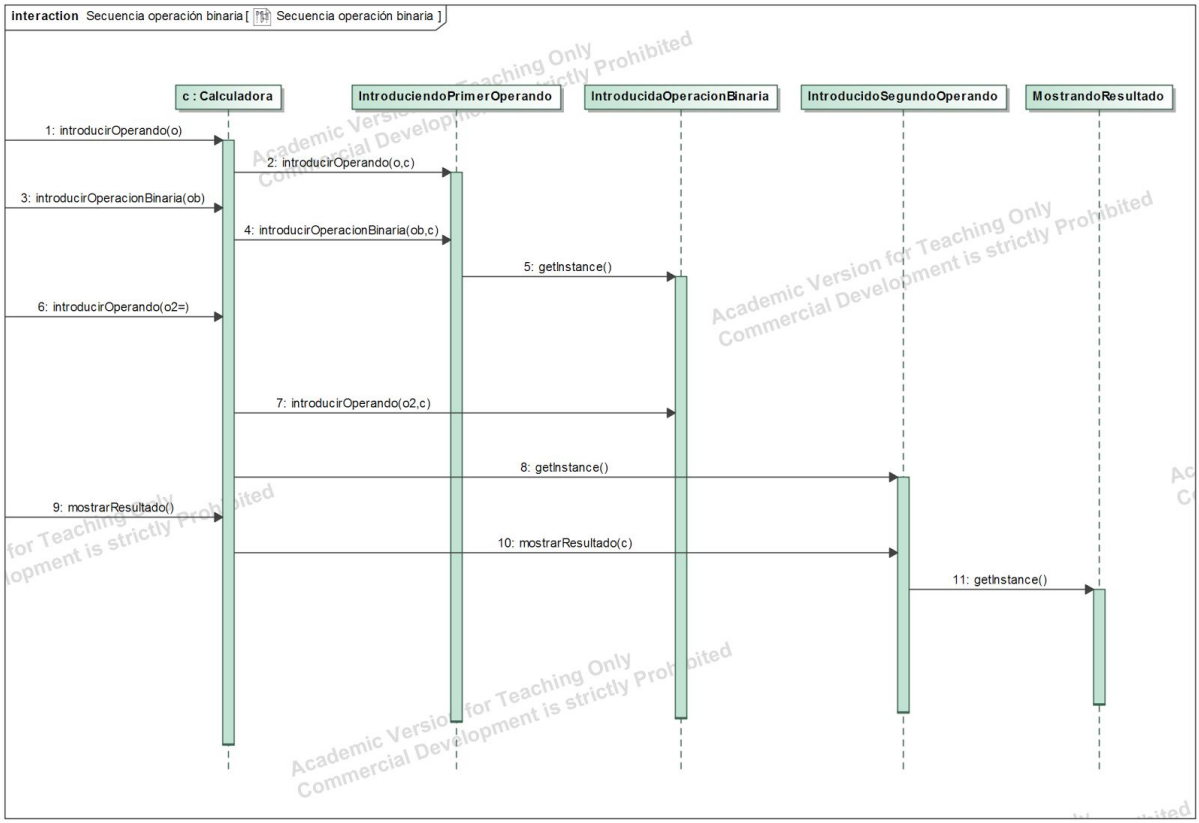
(fig. 6 - Diagrama de Estados de la Calculadora)

3.3. Diagramas de secuencia.

En el siguiente diagrama de secuencia reflejamos el funcionamiento del cálculo de una operación binaria ejecutada desde la Calculadora. La secuencia de instrucciones en el programa sería algo del estilo:

```
Calculadora c = new Calculadora();
c.introducirOperando(new Operando(“1”));
c.introducirOperacionBinaria(Suma.getInstance());
c.introducirOperando(new Operando(“2”));
c.mostrarResultado();
```

Todas las llamadas se hacen desde la calculadora, pero ésta delega en el estado correspondiente, y a su vez, cada estado es responsable de hacer la transición oportuna.



(fig. 7 - Diagrama de Secuencia de Operación Binaria)

4. Notas

- Hemos adjuntado en nuestro proyecto las imágenes de todos los diagramas aquí presentes en su resolución original para garantizar su legibilidad. Se encuentran el el directorio /DS-34-11-PD/doc/diagrams