|  |  |  |  |
| --- | --- | --- | --- |
| NOMBRE | Diseño de sistemas | | |
| AREA | Tecnologías Aplicadas | REGIMEN | Semestral |
| MODALIDAD | Por examen final | CARGA HORARIA | |  | | --- | | 6 (seis) semanales,  90 totales | |

Se entrega la práctica en formato digital y papel. El formato digital debe tener el mismo encabezado que el presente texto. El formato en papel debe contener fecha, materia, curso, datos del alumno.

***Alumno:***

***Curso:***

***Fecha de entrega:***

**Tema: Patrones de diseño**

**Interprete**

Este patrón busca representar un lenguaje mediante reglas gramáticas. Para ello define estas reglas gramáticas y cómo interpretarlas. Utiliza una clase para representar una regla gramática.

Si un tipo particular de problema se presenta frecuentemente, puede ser provechoso expresar los diferentes casos del problema como sentencias de un lenguaje simple. Se puede, entonces, construir un intérprete que resuelva el problema interpretando dichas sentencias.

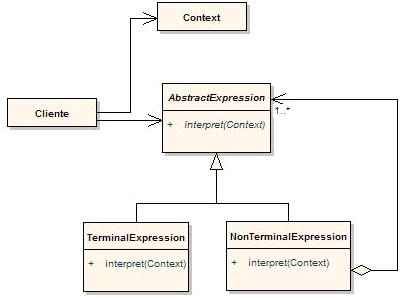
Cuando Utilizarlo.

Este patrón se debe utilizar cuando hay un lenguaje que interpretar y se puede interpretar sus palabras como árboles sintácticos abstractos. Para ello, la gramática debe ser simple.

Difícilmente el desarrollador utilice este patrón en algún momento de su vida, lo que no quita que no sea un patrón utilizado. La situación ideal que se debe considerar para aplicar este patrón es que exista un lenguaje sencillo que pueda interpretarse con palabras. El ejemplo más claro es JAVA: este lenguaje permite escribir en archivos .java entendibles por humanos y luego este archivo es compilado e interpretado para que pueda ejecutar sentencias entendibles por una máquina.

**Diagramas UML**

**Vista estática: diagrama de clases**



AbstractExpression: declara una interfaz para la ejecución de una operación.

TerminalExpression: implementa una operación asociada con los símbolos terminales de la gramática

NonterminalExpression: implementa una operación de interpretación asociada con los símbolos no terminales de la gramática.

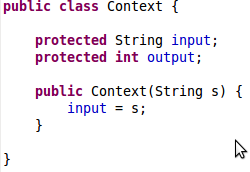
Context: contiene información global para el intérprete.

Client: construye un árbol sintáctico abstracto que representa una sentencia particular en el lenguaje que la gramática define. El cliente construye una expresión

***Caso práctico a desarrollar en clase***

Veamos un ejemplo donde se utiliza el intérprete para interpretar los números romanos mediante ciertas reglas matemáticas y convertirlo en un número de escala decimal.

Primero creamos el contexto: esto es una clase donde tiene un input (un String ya que los números romanos son letras) y un output (un int ya que la respuesta será un número).



Lo que sigue a continuación es puro algoritmo: si pensamos cómo funcionan los números romanos vamos a encontrar un patrón (pongo como ejemplo con escala de 10 pero es lo mismo para el resto de las escalas): de 1 a 3 es el mismo signo I (ocupa 1 espacio por vez), el 4 es una combinación del 5 con el 1.... IV (ocupan dos lugares), el 5 es V (ocupa 1 lugar) y el próximo signo que hace combinación es el 9 (IX) que utiliza un signo de una escala mayor.

