

INTERPRETER PATTERN

NOTA PRACTICAS

Nota final:
 $\text{nota prácticas} * 0.2 +$
 $\text{nota examen} * 0.8$

NOTA FINAL

NOTA EXAMEN

```
public double notaFinal (double practica, double examen) {  
    return practica * 0.2 + examen * 0.8;  
}
```


NOTA PRACTICAS

NOTA CONDUCTA

NOTA EXAMEN

Nota final:
 $\text{nota prácticas} * 0.2 +$
 $\text{nota conducta} * 0.1 +$
 $\text{nota examen} * 0.7$

NOTA FINAL

```
public double notaFinal (double practica, double conducta, double examen) {  
    return practica * 0.2 + conducta * 0.1 + examen * 0.7;  
}
```


Cada profesor puede variar los factores de ponderación

```
public double notaFinal ( double practica, double factorPractica,  
                           double conducta, double factorConducta,  
                           double examen, double factorExamen ) {  
    return  practica * factorPractica +  
            conducta * factorConducta +  
            examen * factorExamen;  
}
```

*Cada profesor puede añadir nuevos criterios de evaluación
La nota final puede no ser una suma ponderada sinó algun otro tipo de fórmula*

....

“

Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.

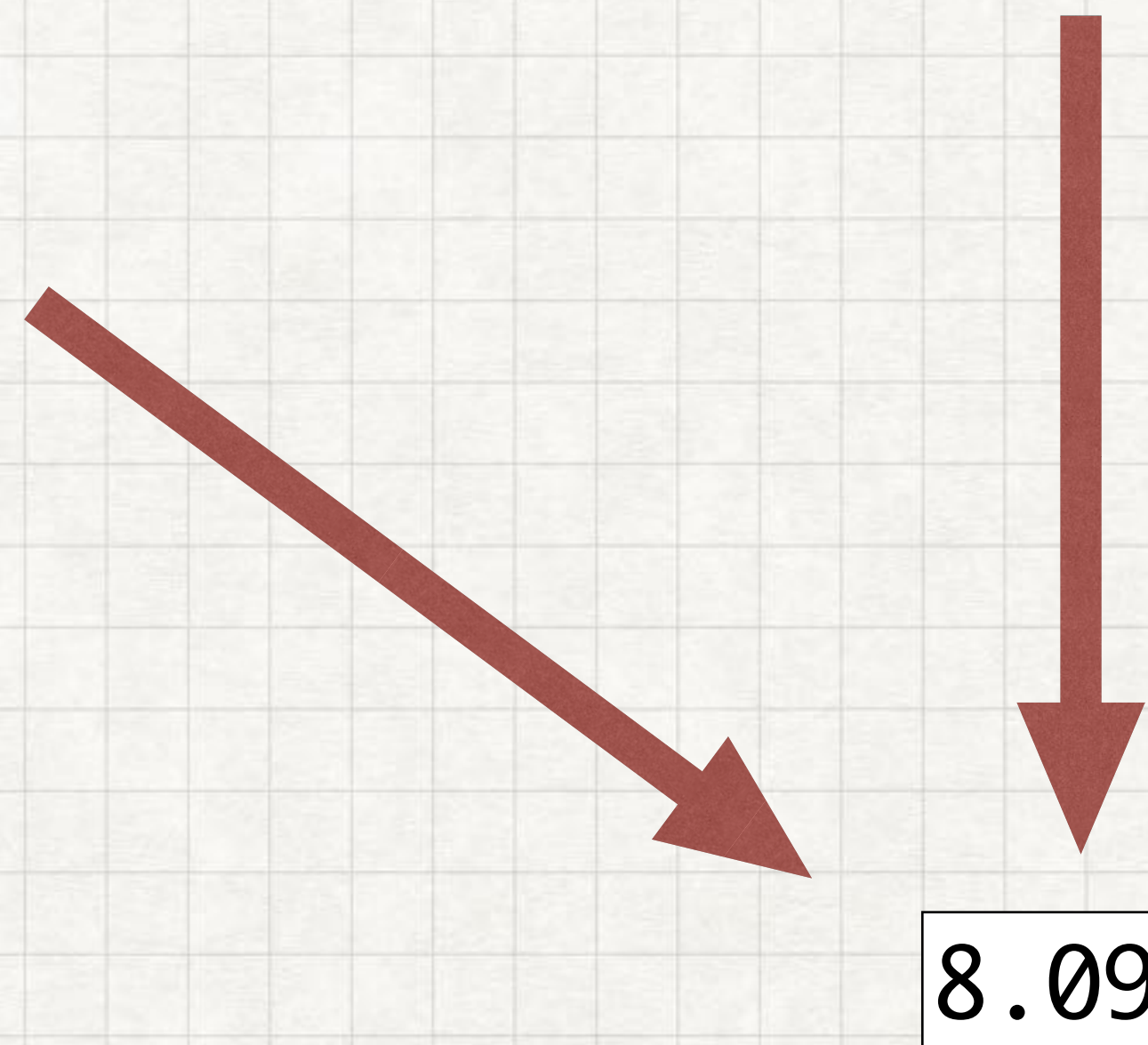
— *GoF*

”

Concepto	Nota
Práctica	7
Conducta	6
Examen	8,7
...	...

$nota_parcial * factor +$

```
String nota = "practica * 0.2 + conducta * 0.1 + examen * 0.7";
```




```
String nota = "practica * 0.2 + conducta * 0.1 + examen * 0.7";
```

Escrito en gramática para representar cálculo de la nota final

Parsing la expresión

REPRESENTACIÓN

Jerarquía de expresiones

Contexto de ejecución

Concepto	Nota
Práctica	7
Conducta	6
Examen	8,7
...	...

INTERPRETER

8.09

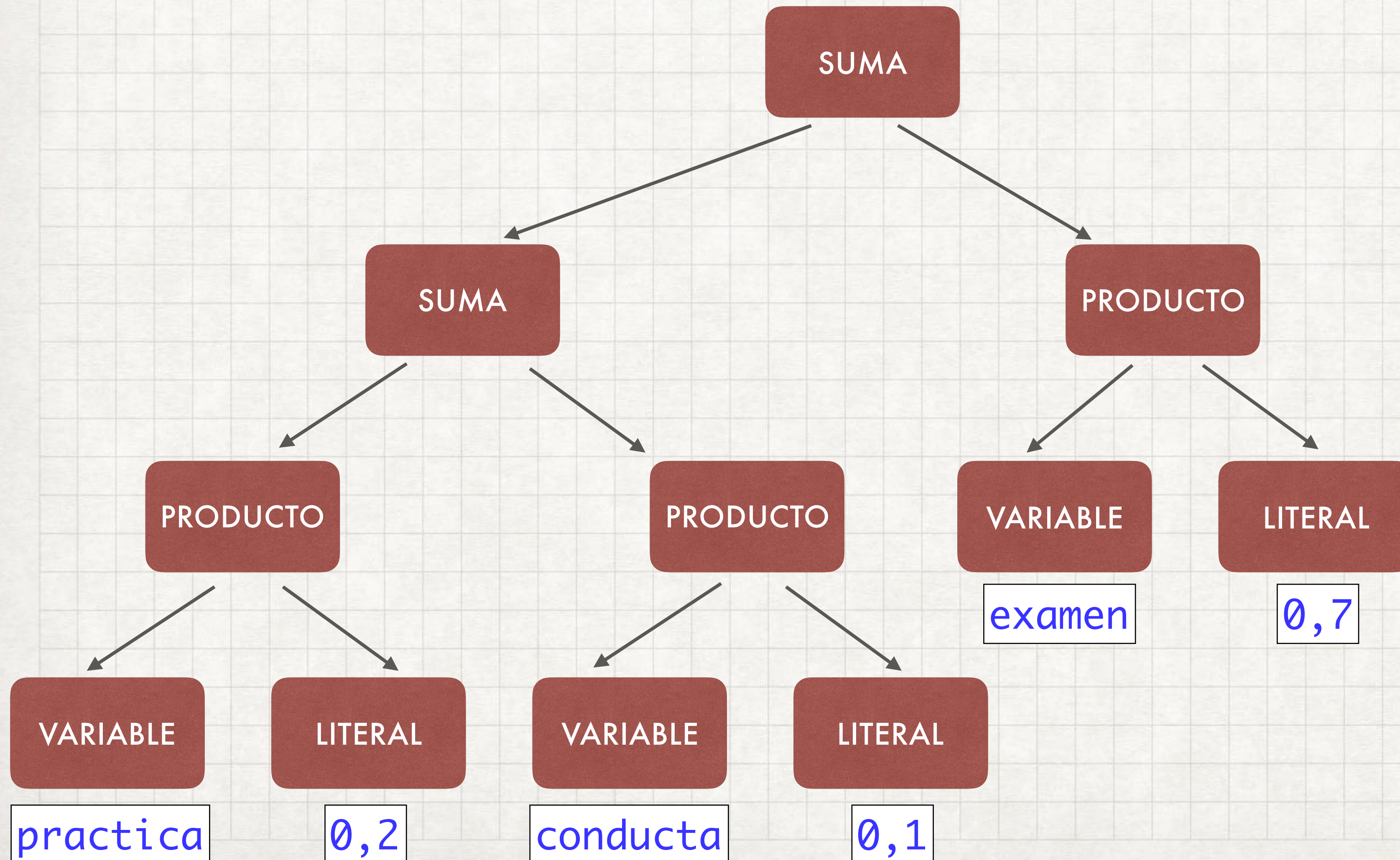
INTERPRETER PATTERN

String nota = "practica * 0.2 + conducta * 0.1 + examen * 0.7";

Parsing la expresión

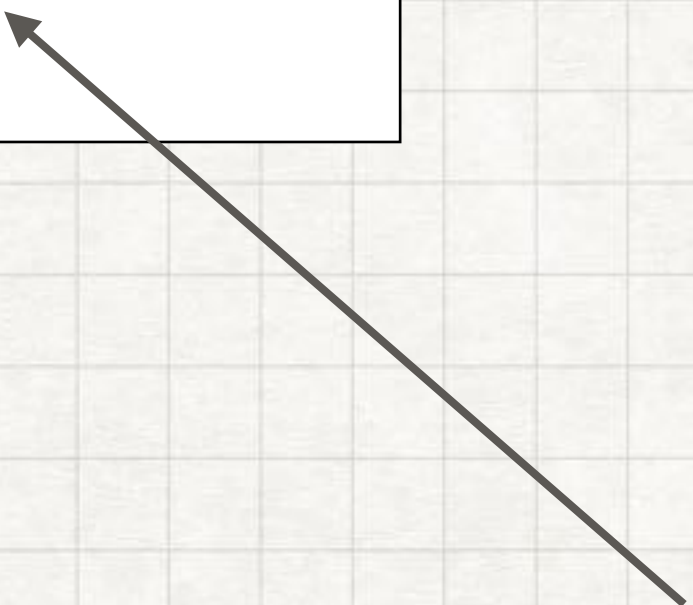


REPRESENTACIÓN

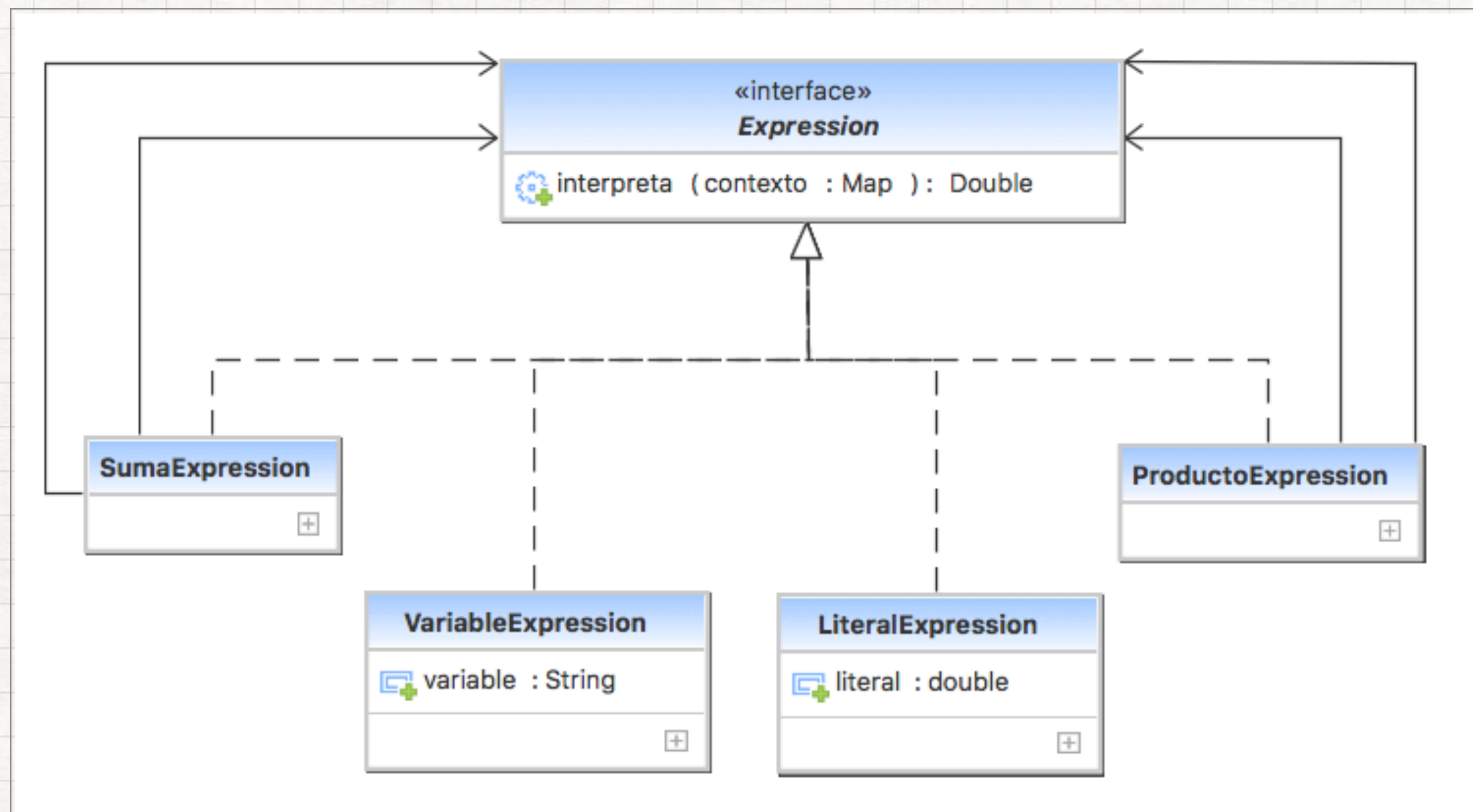



```
public interface Expression {  
    Double interpreta (Map<String, Double> contexto);  
}
```

8.09



Concepto	Nota
Práctica	7
Conducta	6
Examen	8,7
...	...




```
public class SumaExpression implements Expression {  
  
    private final Expression expr1;  
    private final Expression expr2;  
  
    public SumaExpression(Expression expr1, Expression expr2) {  
        this.expr1 = expr1;  
        this.expr2 = expr2;  
    }  
  
    @Override  
    public Double interpreta(Map<String, Double> contexto) {  
        return expr1.interpreta(contexto) + expr2.interpreta(contexto);  
    }  
}
```

*Una suma se compone
de dos expresiones
diferentes*

*Definición recursiva:
definimos la expresión en
términos de otras
expresiones*

*En el método interpreta indicamos cual es el resultado de
interpretar esta expresión*

*El contexto se propaga a las
subexpresiones*


```
public class VariableExpression implements Expression {  
    private final String variable;  
  
    public VariableExpression(String variable) {  
        this.variable = variable;  
    }  
  
    @Override  
    public Double interpreta(Map<String, Double> contexto) {  
        Double valor = contexto.get(variable);  
        if (valor == null) {  
            return 0.0;  
        }  
        return valor;  
    }  
}
```

*Una VariableExpression
evalua el valor de una
variable para un
determinado contexto*

Uso del contexto

¿Que hacemos si la variable no está definida?


```
String nota = "practica * 0.2 + conducta * 0.1 + examen * 0.7";
```



```
public class LiteralExpression implements Expression {  
    private final double literal;  
  
    public LiteralExpression(double literal) {  
        this.literal = literal;  
    }  
  
    @Override  
    public Double interpreta(Map<String, Double> contexto) {  
        return literal;  
    }  
}
```

Los literales son valores que aparecen directamente en el código

El valor del literal esta disponible justo al construirse la expresión

Para el método interpreta, LiteralExpression no hace uso del contexto pero, evidentemente, debe estar en la definición del método para cumplir con la interface Expression


```
String nota = "practica * 0.2 + conducta * 0.1 + examen * 0.7";
```



Faltaría construir el parser que construyese la Expression notaFinal a partir del texto anterior

```
Expression notaFinal =  
    new SumaExpression (  
        new SumaExpression (  
            new ProductoExpression( new VariableExpression("practica"),  
                                    new LiteralExpression(0.2)  
            ),  
            new ProductoExpression( new VariableExpression("conducta"),  
                                    new LiteralExpression(0.1)  
            )  
        ),  
        new ProductoExpression( new VariableExpression("examen"),  
                                new LiteralExpression(0.7)  
        )  
    );
```


Contexto

```
Map<String, Double> notas = new HashMap<>();
```

```
notas.put("practica", 7.0);
```

```
notas.put("conducta", 6.0);
```

```
notas.put("examen", 8.7);
```

```
System.out.println("nota final " + notaFinal.interpreta(notas));
```

Interpreter

nota final 8.09

INTERPRETER PATTERN