**Интерпретатор**

# **Назначение и цель применения паттерна**

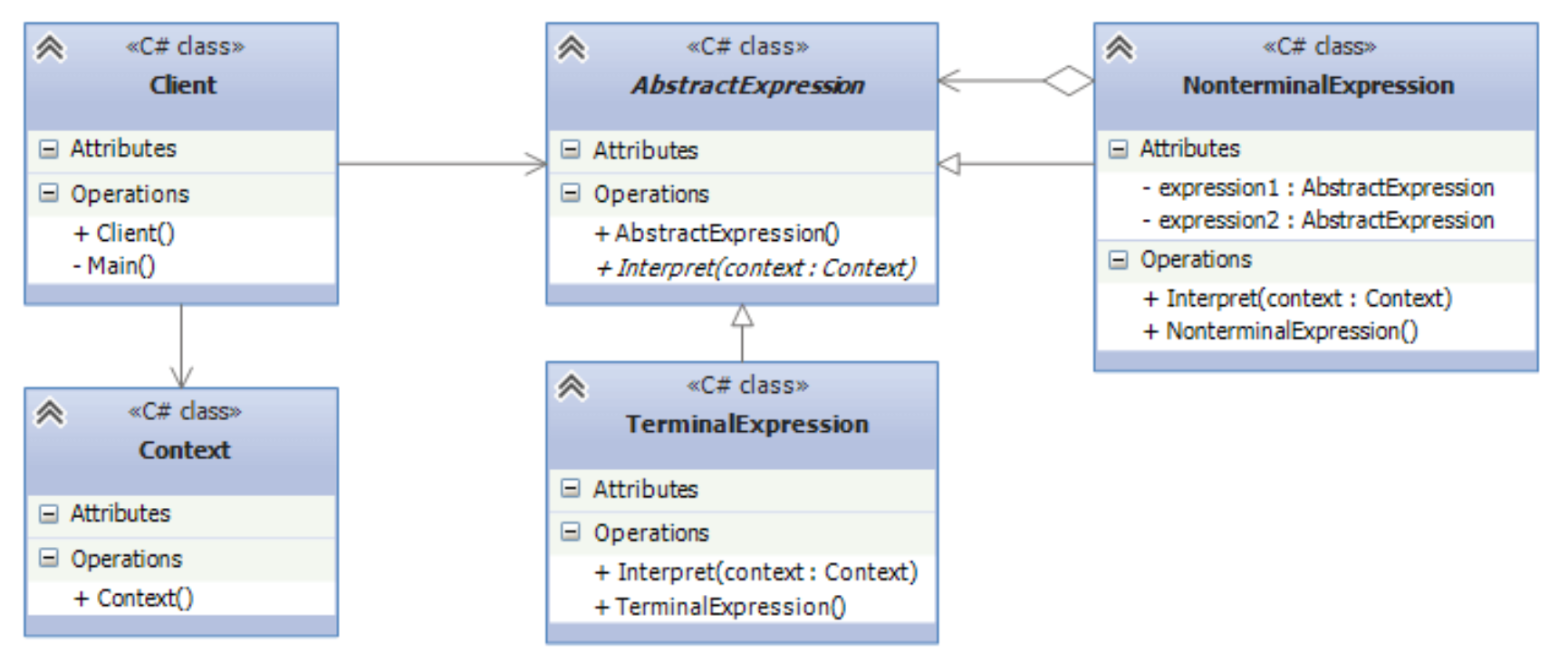
**Интерпретатор** — это поведенческий паттерн проектирования, который определяет представление грамматики для заданного языка и интерпретатор предложений этого языка. Как правило данный шаблон проектирования применяется для часто повторяющихся операций.

Основное назначение: решение часто встречающейся, подверженной изменениям задачи.

Цель применения паттерна: отображение проблемной области в язык, язык в грамматику, грамматику – в иерархию объектно-ориентированного проектирования.

# **UML диаграммы паттерна**

Для описания структуры паттерна Interpreter необходимо построить UML диаграмму классов.



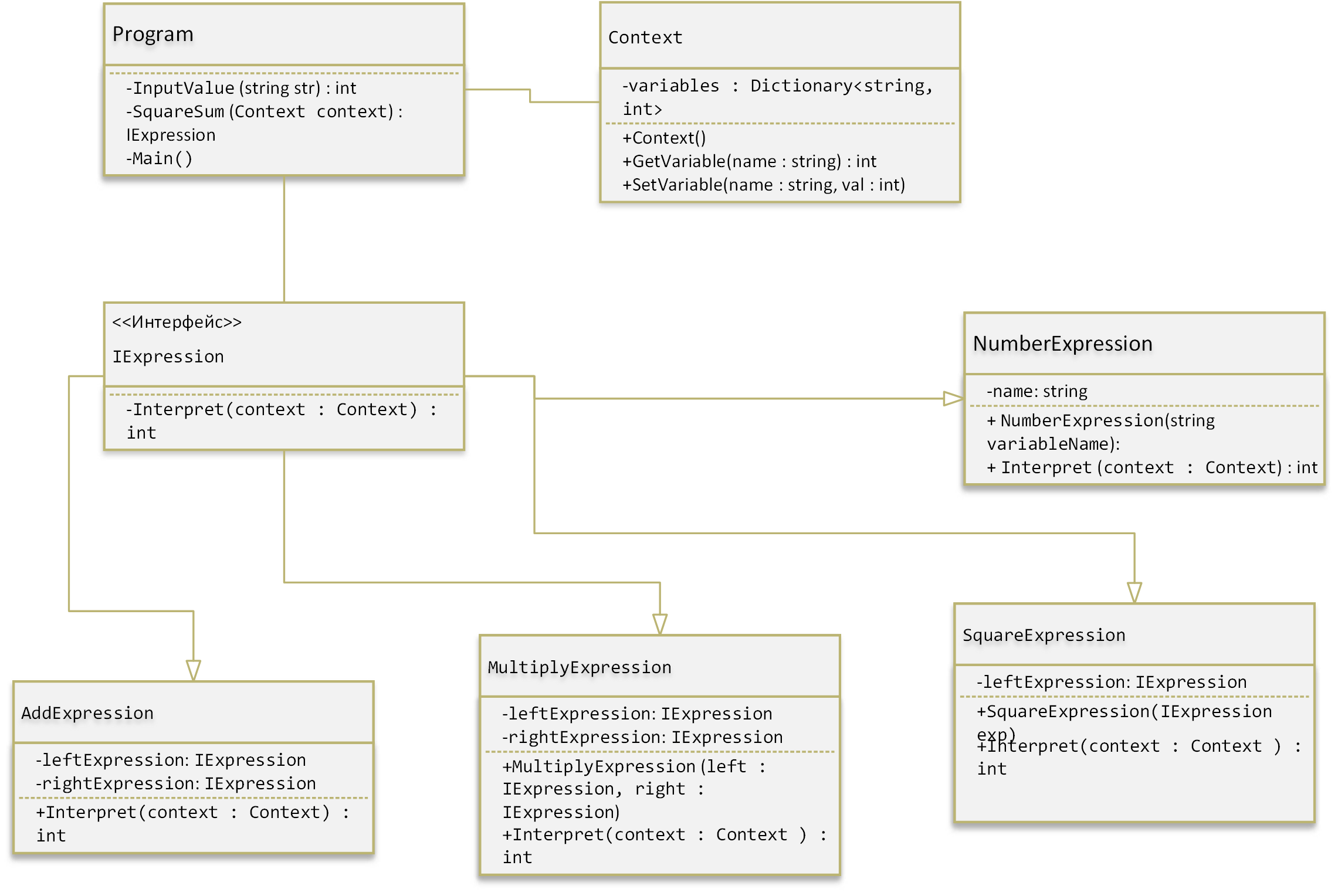
UML диаграмма **Interpreter**

Основные элементы диаграммы:

* + - 1. **AbstractExpression (Абстрактное выражение)** определяет интерфейс выражения, объявляет метод Interpret().
      2. **TerminalExpression (Терминальное выражение)**, реализует метод Interpret() для терминальных символов грамматики. Для каждого символа грамматики создается свой объект TerminalExpression.
      3. **NonterminalExpression (Нетерминальное выражение)**, представляет правило грамматики. Для каждого отдельного правила грамматики создается свой объект NonterminalExpression.
      4. **Context (Контекст)** содержит общую для интерпретатора информацию. Может использоваться объектами терминальных и нетерминальных выражений для сохранения состояния операций и последующего доступа к сохраненному состоянию.
      5. **Client (Клиент)** строит предложения языка с данной грамматикой в виде абстрактного синтаксического дерева, узлами которого являются объекты TerminalExpression и NonterminalExpression.

Методы Interpret в нетерминальных выражениях позволяют реализовать правила грамматики. При этом мы легко может добавить новые правила грамматики, определив новые объекты NonterminalExpression со своей реализацией метода Interpret. Однако данный паттерн подходит только для тех случаев, когда правила грамматики относительно простые. В более сложных случаях следует выбирать другие способы проектирования приложения.

Для реализуемого примера была построена диаграмма классов:



UML Диаграмма классов программы InterpretorOfSquareSum

# **Области применения**

Примеры областей, в которых можно применять паттерн Interpreter:

* + - 1. Вычисление простейших операций вычисления.
      2. Интерпретация чисел из одной системы счисления в другую.
      3. Интерпретация звуков можно программно описать с помощью нот.

Данные области являются наиболее распространенными для применения интерпретатора.

# **Особенности паттерна**

Достоинства паттерна:

* грамматику становится легко расширять и изменять;
* реализации классов, описывающих узлы абстрактного синтаксического дерева похожи (легко кодируются);
* можно легко изменять способ вычисления выражений.

Недостатки паттерна:

* Сопровождение грамматики с большим числом правил затруднительно.

# **Пример реализации**

Была разработана программа, интерпретирующая вычисление квадратного корня из суммы двух чисел.

interface IExpression

{

int Interpret(Context context);

}

Создан интерфейс IExpression, для которого был определен метод Interpret, который при определении новых выражений будет переопределен в зависимости от грамматики.

class NumberExpression : IExpression

{

string name; //имя переменной

public NumberExpression(string variableName)

{

name = variableName;

}

public int Interpret(Context context)

{

return context.GetVariable(name);

}

}

Создан класс NumberExpression, наследованный от интерфейса IExpression, который интерпретирует числовые (терминальные) выражения.

class AddExpression : IExpression

{

IExpression leftExpression;

IExpression rightExpression;

public AddExpression(IExpression left, IExpression right)

{

leftExpression = left;

rightExpression = right;

}

public int Interpret(Context context)

{

return

leftExpression.Interpret(context) +

rightExpression.Interpret(context);

}

}

class MultiplyExpression : IExpression

{

IExpression leftExpression;

IExpression rightExpression;

public MultiplyExpression(IExpression left, IExpression right)

{

leftExpression = left;

rightExpression = right;

}

public int Interpret(Context context)

{

return

leftExpression.Interpret(context) \*

rightExpression.Interpret(context);

}

}

class SquareExpression : IExpression

{

IExpression expression;

public SquareExpression(IExpression exp)

{

expression = exp;

}

public int Interpret(Context context)

{

return

expression.Interpret(context) \*

expression.Interpret(context);

}

}

Были созданы классы AddExpression, MultiplyExpression и SquareExpression, которые интерпретируют нетерминальные выражения сложения, умножения и возведения числа в квадрат соответственно.

class Program

{

//int a = 0;

//int b = 0;

static int InputValue(string str)

{

bool ok = false;

int inp = 0;

do

{

Console.Write(str + ": ");

ok = Int32.TryParse(Console.ReadLine(), out inp);

}

while (!ok);

return inp;

}

Функция для введения числа, с проверкой на правильность ввода.

static IExpression SquareSum(Context context)

{

return new AddExpression(

new AddExpression(

new SquareExpression(

new NumberExpression("a")),

new MultiplyExpression(

new MultiplyExpression(

new NumberExpression("2"), new NumberExpression("a")),

new NumberExpression("b")

)

),

new SquareExpression(new NumberExpression("b")));

}

Функция квадрат суммы, которая описывает последовательность выполнения интерпретируемых действий при получении квадрата суммы двух введенных чисел.

static void Main(string[] args)

{

Console.WriteLine("Вычисление квадрата суммы первых двух чисел.");

int a = InputValue("Введите первое число");

int b = InputValue("Введите второе число");

Context context = new Context();

context.SetVariable("a", a);

context.SetVariable("b", b);

context.SetVariable("2", 2);

IExpression expression = SquareSum(context);

int result = expression.Interpret(context);

Console.WriteLine("Результат: {0}", result);

Console.Read();

}

}