Пермский филиал федерального государственного автономного
образовательного учреждения высшего образования
«Национальный исследовательский университет
«Высшая школа экономики»

Факультет экономики, менеджмента и бизнес-информатики

Рязанов Иван Дмитриевич

ОРГАНИЗАЦИЯ ПАТТЕРНОВ ПРОЕКТИРОВАНИЯ. ПОВЕДЕНЧЕСКИЙ ПАТТЕРН «ИНТЕРПРЕТАТОР»

Лабораторная работа

студента образовательной программы «Программная инженерия»
по направлению подготовки *09.03.04 Программная инженерия*

Руководитель
к.т.н., доцент кафедры
Информационных технологий
в бизнесе НИУ ВШЭ-Пермь

А.В. Кычкин

Пермь, 2020 год

Оглавление

Глава 1. Паттерн «Интерпретатор»

Название и классификация паттерна. Интерпретатор — паттерн, определяющий представление грамматики для заданного языка и интерпретатор предложений этого языка. Как правило, данный шаблон проектирования применяется для часто повторяющихся операций.

Назначение. Паттерн «Интерпретатор» определяет грамматику простого языка для проблемной области, представляет грамматические правила в виде языковых предложений и интерпретирует их для решения задачи. Для представления каждого грамматического правила паттерн «Интерпретатор» использует отдельный класс. А так как грамматика, как правило, имеет иерархическую структуру, то иерархия наследования классов хорошо подходит для ее описания.

1. Для заданного языка определяет представление его грамматики, а также интерпретатор предложений этого языка.

2. Отображает проблемную область в язык, язык – в грамматику, а грамматику – в иерархии объектно-ориентированного проектирования.

Применимость. Интерпретатор следует использовать, когда необходимо интерпретировать запись в другом языке и т.д. Как один из примеров может служить перевод римских цифр в арабские. В общих случаях паттерн следует использовать тогда, когда задача соответствует следующему описанию: - пусть в некоторой, хорошо определенной области, периодически случается некоторая проблема. Если эта область может быть описана некоторым «языком», то проблема может быть легко решена с помощью «интерпретирующей машины».

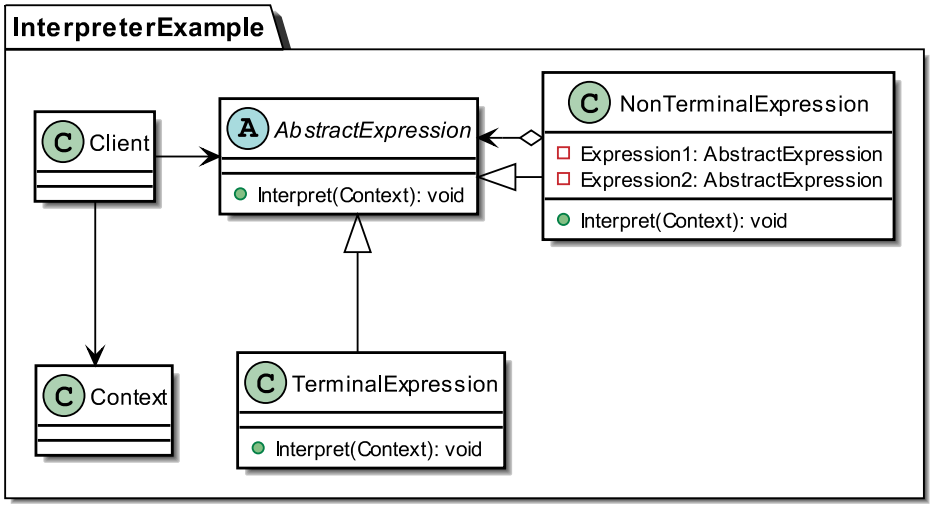

Рис. 1.1. Диаграмма классов паттерна «Интерпретатор»

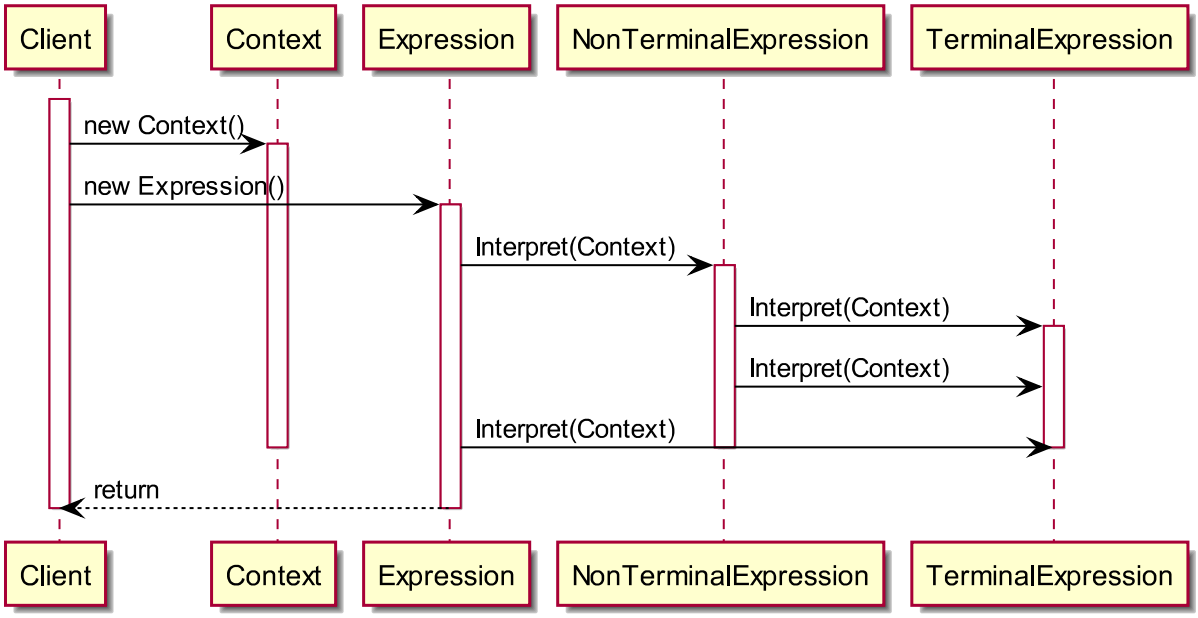

Рис. 1.2. Диаграмма последовательности паттерна «Интерпретатор»

Участники

1. `AbstractExpression`: 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`: содержит общую для интерпретатора информацию. Может использоваться объектами терминальных и нетерминальных выражений для сохранения состояния операций и последующего доступа к сохраненному состоянию.

5. `Client`: строит предложения языка с данной грамматикой в виде абстрактного синтаксического дерева, узлами которого являются объекты `TerminalExpression` и `NonterminalExpression`.

Отношения.

Клиент строит (или получает в готовом виде) предложение в виде абстрактного синтаксического дерева, в узлах которого находятся объекты классов `NonterminalExpression` и `TerminalExpression`. Затем клиент инициализирует контекст и вызывает операцию `Interpret()`; В каждом узле вида `NonterminalExpression` через операции `Interpret` определяется операция `Interpret` для каждого подвыражения. Для класса `TerminalExpression` операция `Interpret` определяет базу рекурсии; Операции `Interpret` в каждом узле используют контекст для сохранения и доступа к состоянию интерпретатора

Плюсы и минусы.

Плюсы:

1. Грамматику легко изменять и расширять. Поскольку для представления грамматических правил в паттерне используются классы, то для

изменения - или расширения грамматики можно применять наследование. Существующие выражения можно модифицировать постепенно, а новые определять как вариации старых (компоновка, агрегация старых).

2. Простая реализация грамматики. Реализации классов, описывающих узлы абстрактного синтаксического дерева, похожи. Такие классы легко кодировать, а зачастую их может автоматически сгенерировать компилятор или генератор синтаксических анализаторов.

Минусы:

1. Сложные грамматики трудно сопровождать. В паттерне интерпретатор определяется по меньшей мере один класс для каждого правила грамматики (для правил, определенных с помощью формы Бэкуса-Наура – BNF, может понадобиться и более одного класса). Поэтому сопровождение грамматики с большим числом правил иногда оказывается трудной задачей.

Области применения

1. Интерпретаторы языков программирования

2. Интерпретация и вычисление алгебраических выражений

Глава 2. Проектирование и реализация

2.1. Проектирование

С помощью паттерна «Интерпретатор» попробуем реализовать интерпретатор эзотерического языка программирования «Brainfuck» (*вынос мозга*). Язык состоит из 8 операторов:

1. `>` — переход к следующей ячейке памяти.
2. `<` — переход к предыдущей ячейке памяти.
3. `+` — увеличить значение в текущей ячейке на 1.
4. `-` — уменьшить значение в текущей ячейке на 1.
5. `.` — напечатать значение из текущей ячейки.
6. `,` — ввести извне значение и сохранить в текущей ячейке.
7. `[` — если значение текущей ячейки ноль, перейти вперёд по тексту программы на ячейку, следующую за соответствующей `]` (с учётом вложенности).
8. `]` — если значение текущей ячейки не нуль, перейти назад по тексту программы на символ `[` (с учётом вложенности).

Операторы `[` и `]` используются для объявления цикла `while (memory[pointer] != 0) {...}`. Всего память состоит из 30000 байт, поэтому её можно реализовать массивом `byte[30000]`. Изначально во всех ячейках памяти записаны нули.

Пример программы «Hello World»:

```
++++++++[>++++[>++>+++>+++>+<<<<-]
>+>+>->>+[<]<-]>>.>---.+++++++..+++.>>
.<-.<.+++.------.--------.>>+.>++.
```

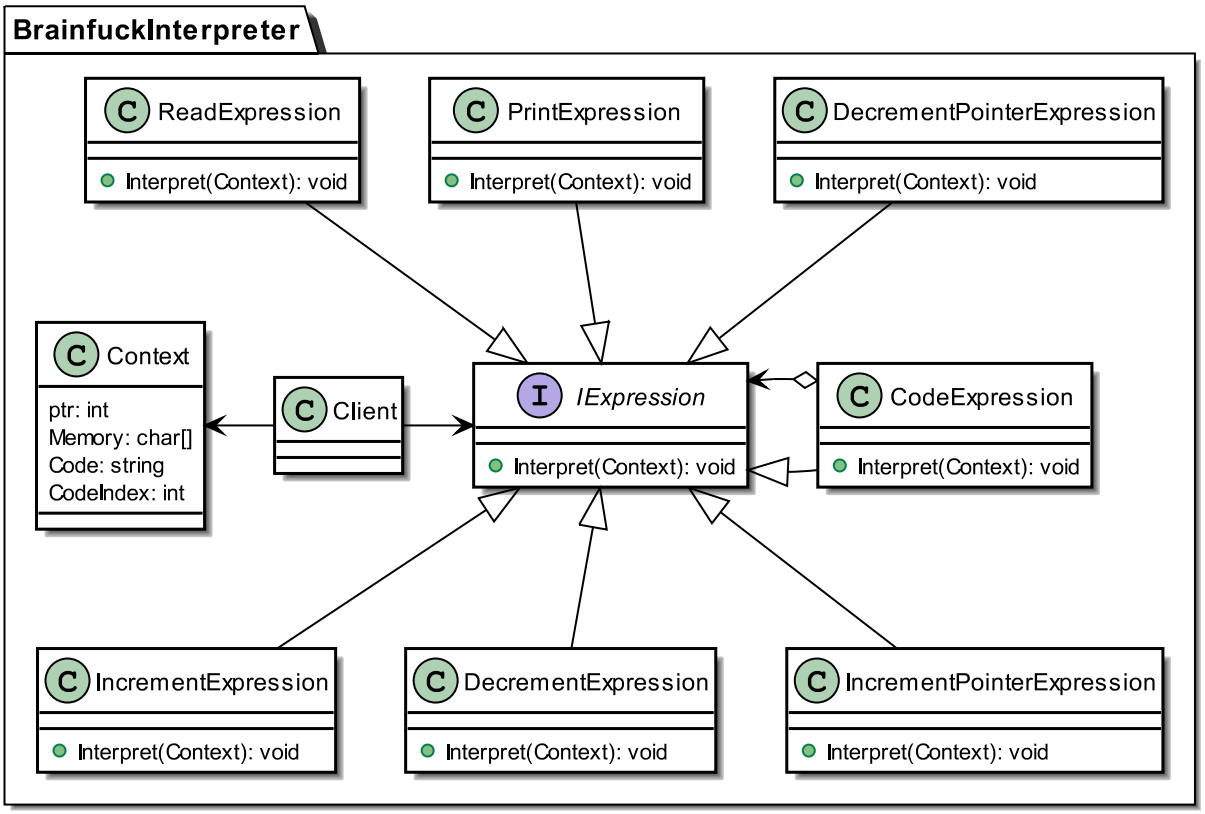

Рис. 2.1. Диаграмма классов

Участники

1. `IExpression`: определяет интерфейс выражения, объявляет метод `Interpret()`.

2. `IncrementExpression`, `DecrementExpression`, `IncrementPointerExpression`, `DecrementPointerExpression`, `PrintExpression`, `ReadExpression`: терминальные выражение, реализуют метод `Interpret()` для терминальных (`<>+-.,`) символов грамматики.

3. `CodeExpression`: нетерминальное выражение. Используется для интерпретации кода и для выполнения циклов.

4. `Context`: содержит общую для интерпретатора информацию (память, указатель на текущую ячейку, код программы, указатель на текущую инструкцию в коде).

2.2. Реализация

Реализация паттерна «Интерпретатор» находится в git-репозитории по ссылке: github.com