Programação Orientada a Objetos
Padrões Interpreter e Visitor

Rodrigo Bonifácio

27 de junho de 2013

Proponha um design para um interpretador de expressões

Proponha um design para um interpretador de expressões, algo bastante simples.

- Dois tipos de valores: inteiros e booleanos
- Expressões de soma e subtração de inteiros
- Expressões booleanas *and*, *or* e *not*
- Expressões podem basicamente ser avaliadas

Padrão Interpreter: Objetivo

Dada uma linguagem, define uma representação para a gramática em conjunto com um interpretador que usa a representação para interpretar sentenças na linguagem.

Padrão Interpreter: Motivação

- Se um problema ocorre frequentemente, pode ser útil expressar instâncias do problema como sentenças em uma linguagem mais simples. Em seguida, é necessário construir um interpretador que soluciona o problema com a interpretação das sentenças.
- O padrão Interpreter descreve como definir uma gramática para linguagens simples, representar sentenças da linguagem e interpretar tais sentenças. O padrão Interpreter usa uma classe para representar cada regra gramatical.

Padrão Interpreter: Aplicabilidade

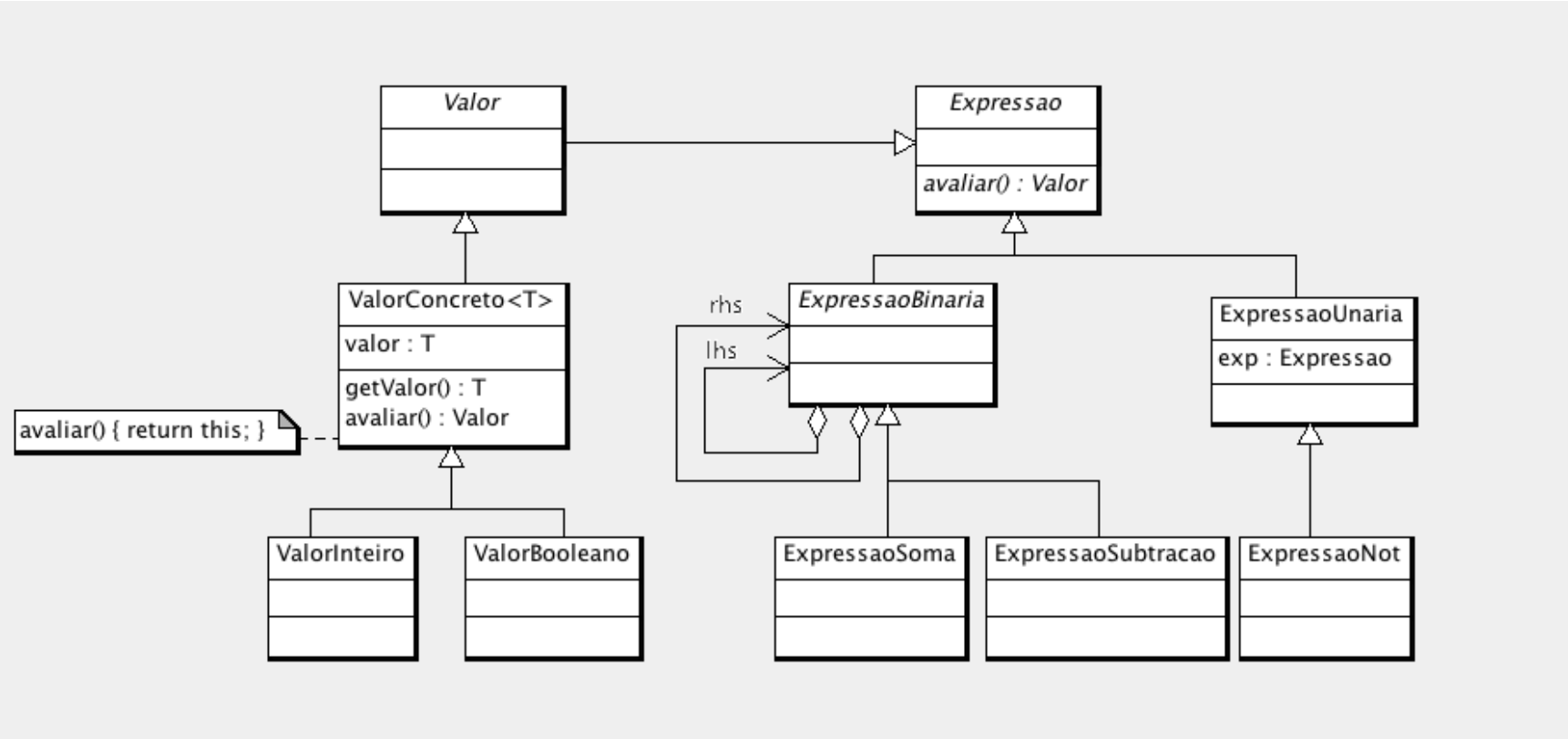
Usar o padrão Interpreter quando existe uma linguagem a ser interpretada, e se deseja representar sentenças nessa linguagem como árvores sintáticas abstratas.

Padrão Interpreter: Aplicabilidade

Usar o padrão Interpreter quando existe uma linguagem a ser interpretada, e se deseja representar sentenças nessa linguagem como árvores sintáticas abstratas. Funciona melhor quando:

- a gramática é simples
- eficiência não é uma preocupação crítica

Padrão Interpreter: Estrutura



Padrão Interpreter: Implementação

Questões específicas do interpretador:

- Como criar a árvore sintática
- Quem é responsável pela avaliação das expressões?

Padrão Interpreter: Código de Exemplo

Baixe o código pelo Moodle, e implemente a expressão de `Let` para se familiarizar com a solução.

Padrão Interpreter: Código de Exemplo

Baixe o código pelo Moodle, e implemente a expressão de `Let` para se familiarizar com a solução.

- Interessante criar testes primeiro (TDD)
- Depois implementar uma classe `ExpressaoLet`

Segundo desafio: implemente o mecanismo de pretty printing, que permite imprimir expressões de forma atrativa para o usuario.

Qual das alterações foi mais modular

Qual das alterações foi mais modular, ou seja, exigiu menor impacto em código existente?

(a) Introdução da expressão subtração

(b) Introdução do mecanismo de pretty printing

Expression Problem

Uma decomposição em termos de dados (árvore sintática) favorece a extensão de elementos sintáticos; mas não a definição de novas operações.

- O padrão Interpreter é aberto a inclusão de novos tipos de expressões; mas fechado para a inclusão de novas operações como pretty printing, type checking, etc.

Expression Problem

Uma decomposição em termos de dados (árvore sintática) favorece a extensão de elementos sintáticos; mas não a definição de novas operações.

- O padrão Interpreter é aberto a inclusão de novos tipos de expressões; mas fechado para a inclusão de novas operações como pretty printing, type checking, etc.
- Esse é uma das perspectivas do *Expression Problem*.

Expression Problem

Uma decomposição em termos de dados (árvore sintática) favorece a extensão de elementos sintáticos; mas não a definição de novas operações.

- O padrão Interpreter é aberto a inclusão de novos tipos de expressões; mas fechado para a inclusão de novas operações como pretty printing, type checking, etc.
- Esse é uma das perspectivas do *Expression Problem*. Na outra perspectiva, a decomposição em termos de operações, usando o padrão de projeto Visitor, favorece a inclusão de novas operações; mas torna não modular a inclusão de novos tipos de expressões.

Padrão de projeto Visitor

Padrão Visitor: Objetivo

Representa uma operação que deve ser executada nos elementos de uma estrutura de objetos. Permite definir uma nova operação sem alterar as classes dos elementos que as operações são aplicadas.

Padrão Visitor: Motivação

Distribuir todas as operações aplicáveis aos nós de uma hierarquia de classes (como uma árvore sintática abstrata) leva a um sistema difícil de entender, manter e evoluir.

- Uma abordagem mais interessante consiste em pemitir que as operações sejam adicionadas de forma independente, e que os nós da hierárquia de classes sejam independentes das operações que são aplicadas aos mesmos.

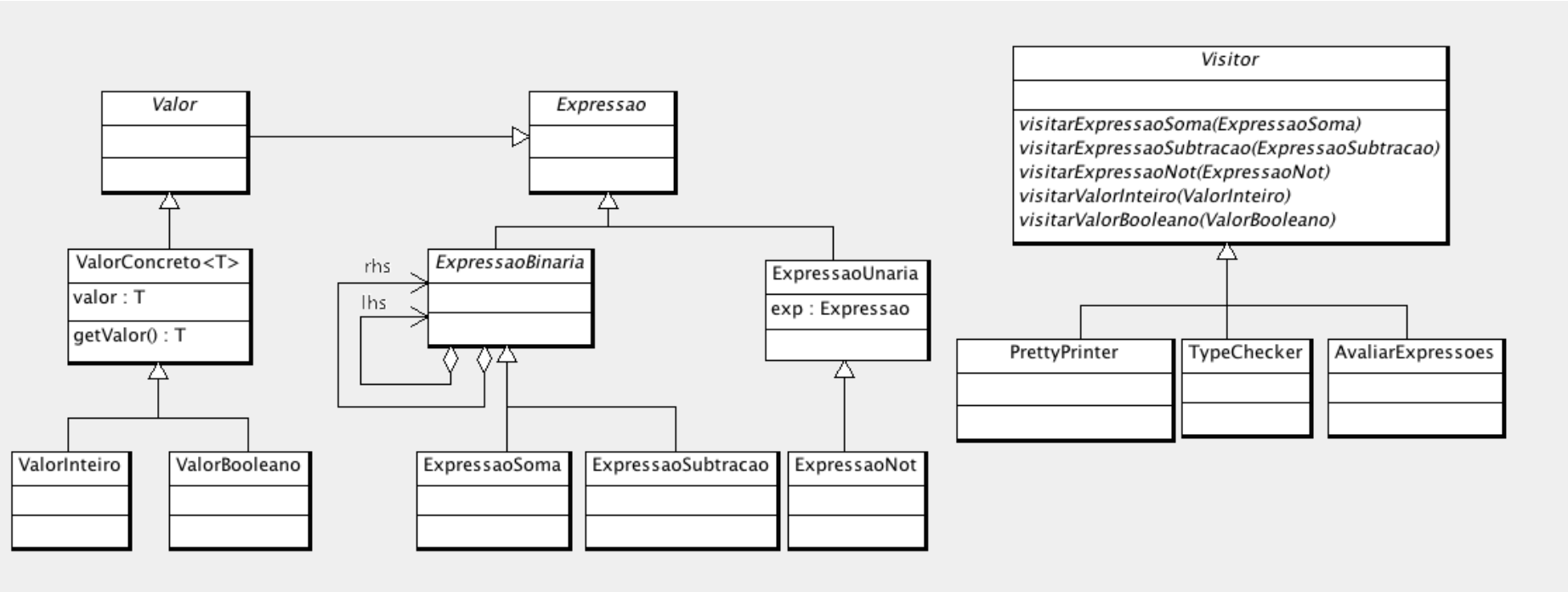
Padrão Visitor: Aplicabilidade

- Operações distintas podem ser aplicadas a uma árvore de objetos, mas queremos uma melhor separação entre as classes que definem a árvore de objetos e as operações em si.
- As classes que definem a estrutura dos objetos raramente muda, mas freqüentemente novas operações sobre os objetos podem ser propostas.

Padrão Visitor: Aplicabilidade

- Operações distintas podem ser aplicadas a uma árvore de objetos, mas queremos uma melhor separação entre as classes que definem a árvore de objetos e as operações em si.
- As classes que definem a estrutura dos objetos raramente muda, mas freqüentemente novas operações sobre os objetos podem ser propostas. Se a estrutura dos objetos altera com certa freqüência, é mais vantajoso definir as operações nas próprias classes (como fizemos usando apenas o padrão de projeto Interpreter).

Padrão Visitor: Estrutura



Padrão Visitor: Consequencias

- Facilita a inclusão de novas operações
- Modulariza operações semelhantes em uma única classe
- Adicionar novas classes a estrutura é uma operação não modular

Padrão Visitor: Implementação

- Single x Double Dispatch
- Responsabilidade da traversia
  - implementação do visitor
  - implementação de um nó da hierarquia
  - iterator

Padrão Interpreter: Código de Exemplo

Baixe o código pelo Moodle, e implemente a expressão de subtração de inteiros e um visitor para verificar os tipos das expressões.