Modelos, Métodos e Técnicas de Engenharia de Software
Visão e análise de projeto Padrões Prática 3 – Interpreter (15)

Prof. Osmar de Oliveira Braz Junior

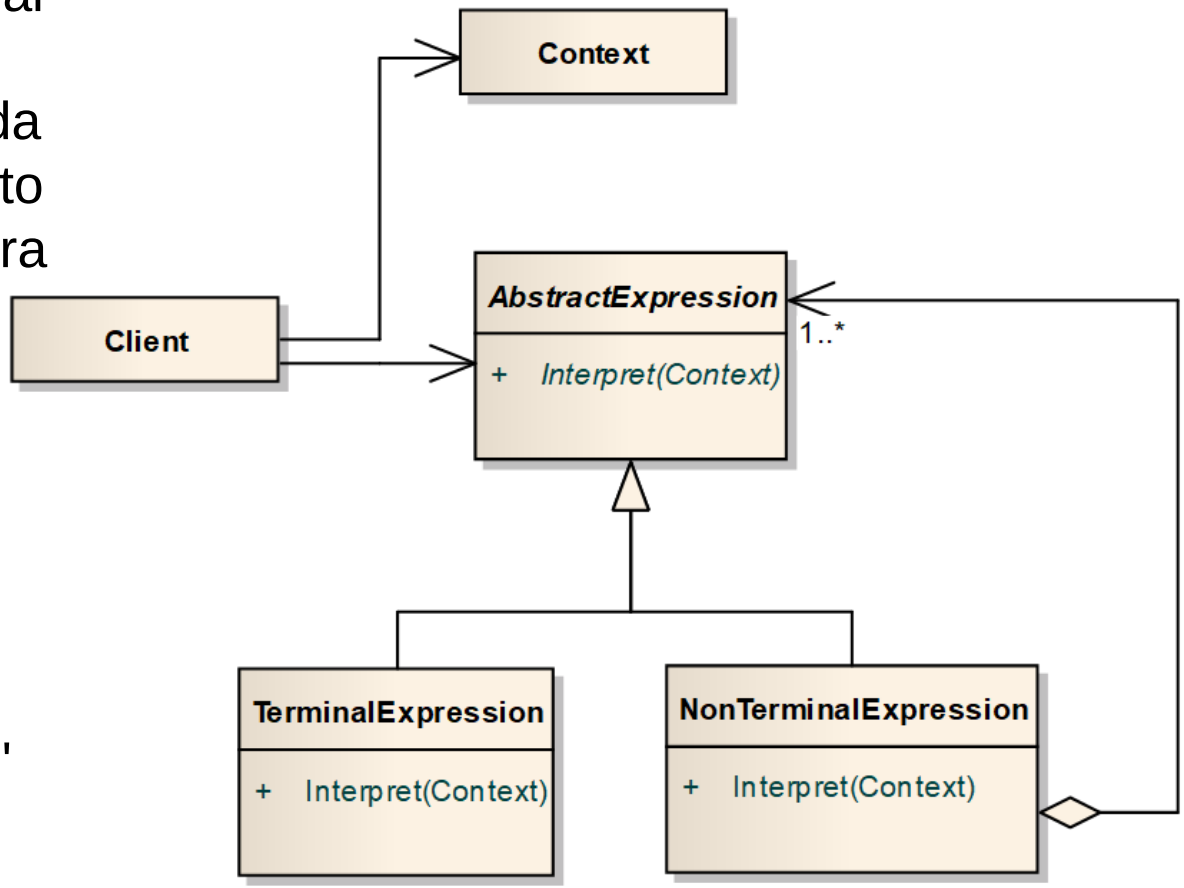
Prof. Richard Henrique de Souza

Objetivos

- Aplicar padrão comportamental ***Interpreter*** em situação problema.

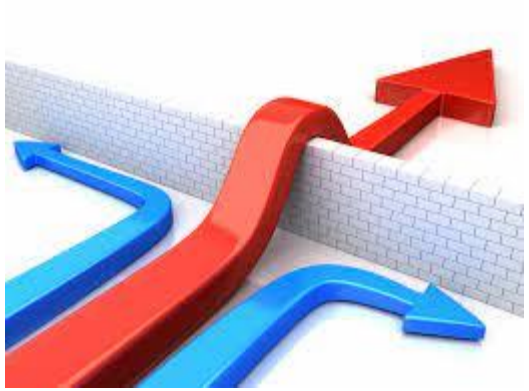
15. Interpreter

Estrutura: O padrão Interpreter sugere modelar o domínio com uma gramática recursiva. Cada regra na gramática é tanto um 'composite' (uma regra que referencia outras regras) ou um 'terminal' (uma folha/nó numa estrutura de árvore). O Interpreter baseia-se na travessia recursiva do padrão Composite para interpretar as 'sentenças' que ele deve processar.



Importante

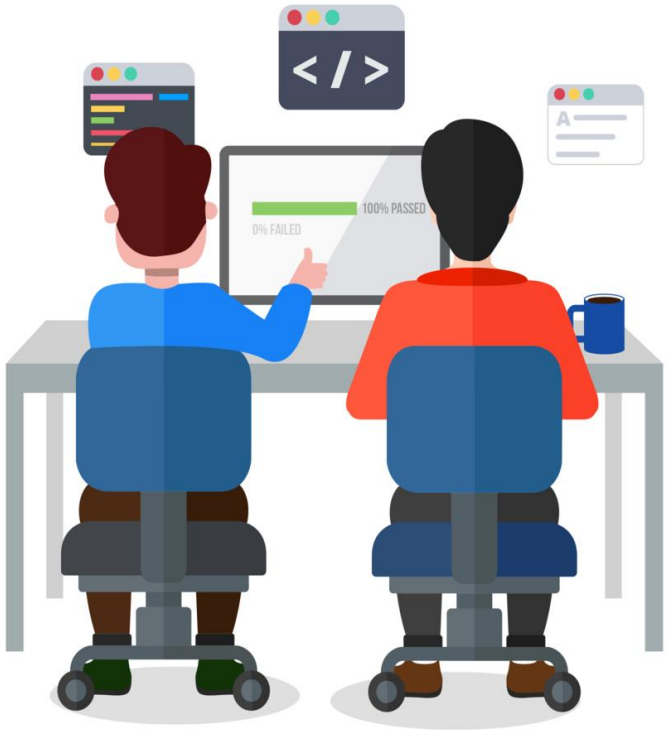
Siga os ROTEIROS !!!



Atividade em Grupo

Para esta atividade crie grupos de 2 alunos, para desenvolver a atividade segundo ***Pair Programming.***

Navegador



Piloto

Extreme Programming, metodologia ágil criada por Kent Beck em 1996.

Pair Programming

- Um é o **piloto**, responsável por escrever o código, o outro o navegador, acompanha a escrita de código e verificar se está de acordo com os **padrões do projeto** e de encontro à solução necessária.
- A intenção desta técnica é **evitar** erros de lógica, e ter um código mais confiável e melhor estruturado, utilizando-se para isso a máxima de que “**duas cabeças pensam melhor do que uma**”.

Preparação do ambiente

- Acesso a ferramenta **draw.io(https://app.diagrams.net/)** para realizar a modelagem.
- Escolha a sua linguagem de programação de preferência
- Escolha uma IDE ou o **git.dev**
- Crie um repositório no github(https://github.com/) para que todos os membros da equipe possam colaborar no desenvolvimento.

Atividade prática

1

15. Interpreter
Propósito

- Definir a gramática de uma linguagem e criar um interpretador que leia instruções nesta linguagem e interprete-as para realizar tarefas.

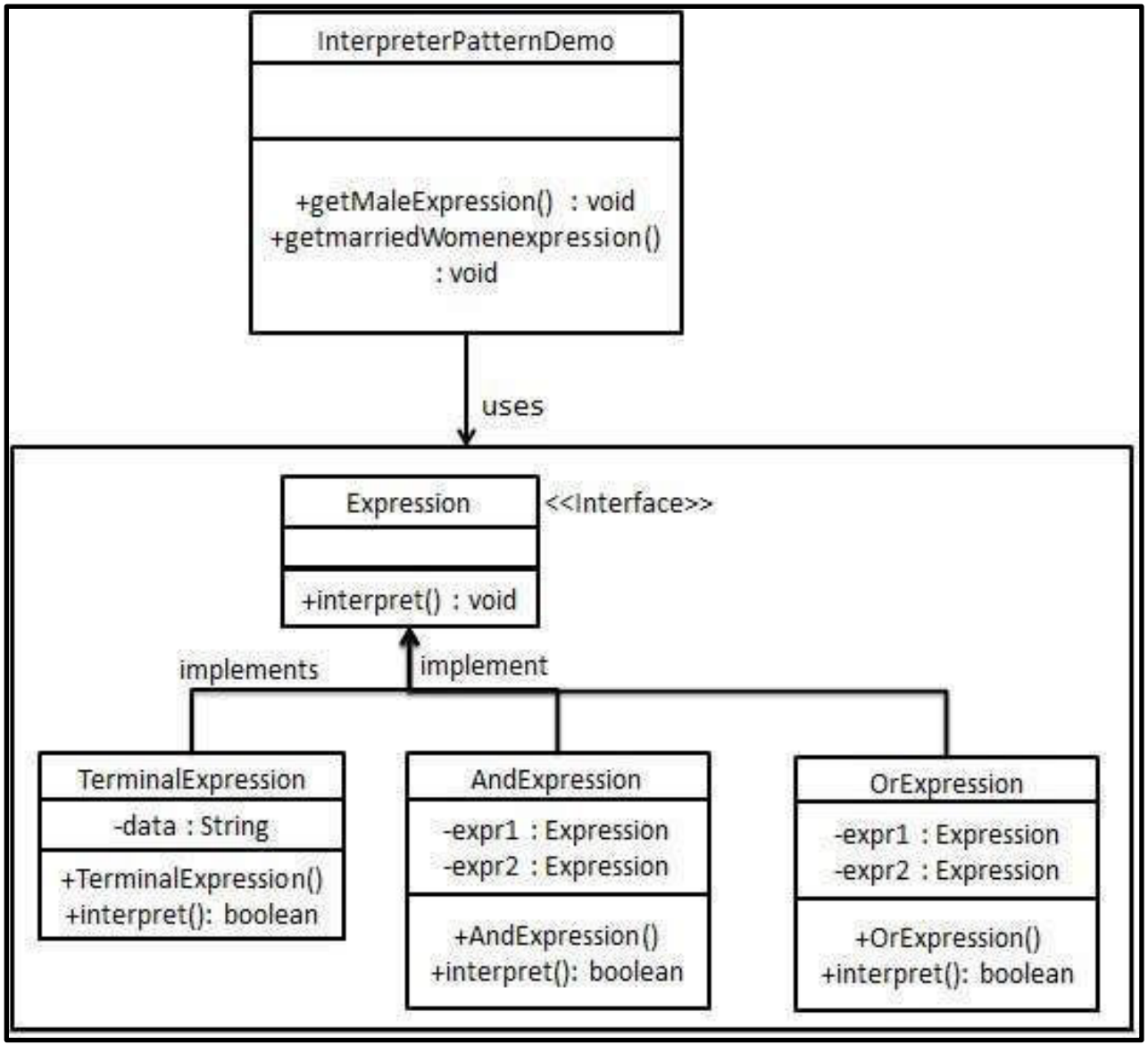
15. Interpreter

- Usar este padrão quando…
  - Existe uma linguagem a ser interpretada que pode ser descrita como uma árvore sintática;
  - Funciona melhor quando:
    - A linguagem é simples;
    - Desempenho não é uma questão crítica

15. Interpreter

- Vantagens e desvantagens
  - É fácil mudar e estender a gramática:
    - Pode alterar expressões existentes, criar novas expressões, etc.;
    - Implementação é simples, pois as estruturas são parecidas.
  - Gramáticas complicadas dificultam:
    - Se a gramática tiver muitas regras complica a manutenção.

15. Interpreter



15. Interpreter
Passo 1

Crie uma interface de expressão.

Expression.java

```
public interface Expression {
   public boolean interpret(String context);
}
```

15. Interpreter
Passo 2

Crie classes concretas implementando a interface acima.

TerminalExpression.java

```
public class TerminalExpression implements Expression {

   private String data;

   public TerminalExpression(String data){
      this.data = data;
   }

   @Override
   public boolean interpret(String context) {

      if(context.contains(data)){
         return true;
      }
      return false;
   }
}
```

15. Interpreter
Passo 2 - Continuação

OrExpression.java

```
public class OrExpression implements Expression {

   private Expression expr1 = null;
   private Expression expr2 = null;

   public OrExpression(Expression expr1, Expression expr2) {
      this.expr1 = expr1;
      this.expr2 = expr2;
   }

   @Override
   public boolean interpret(String context) {
      return expr1.interpret(context) || expr2.interpret(context);
   }
}
```

15. Interpreter
Passo 2 - Continuação

AndExpression.java

```
public class AndExpression implements Expression {

   private Expression expr1 = null;
   private Expression expr2 = null;

   public AndExpression(Expression expr1, Expression expr2) {
      this.expr1 = expr1;
      this.expr2 = expr2;
   }

   @Override
   public boolean interpret(String context) {
      return expr1.interpret(context) && expr2.interpret(context);
   }
}
```

15. Interpreter
Passo 3

O InterpreterPatternDemo usa a classe Expression para criar regras e, em seguida, analisá-las.

InterpreterPatternDemo.java

```
public class InterpreterPatternDemo {

   //Rule: Robert and John are male
   public static Expression getMaleExpression(){
      Expression robert = new TerminalExpression("Robert");
      Expression john = new TerminalExpression("John");
      return new OrExpression(robert, john);
   }

   //Rule: Julie is a married women
   public static Expression getMarriedWomanExpression(){
      Expression julie = new TerminalExpression("Julie");
      Expression married = new TerminalExpression("Married");
      return new AndExpression(julie, married);
   }

   public static void main(String[] args) {
      Expression isMale = getMaleExpression();
      Expression isMarriedWoman = getMarriedWomanExpression();

      System.out.println("John is male? " + isMale.interpret("John"));
      System.out.println("Julie is a married women? " + isMarriedWoman.interpret("Married Julie"));
   }
}
```

15. Interpreter
Passo 4

- Terminamos
  - Teste sua implementação

Compile e **Mostre** o código para o professor
  - Pense, o que você fez aqui ?

Lembre de salvar no seu github

Conclusão

Os padrões comportamentais tem como principal função designar responsabilidades entre objetos.

Referências

- PRESSMAN, Roger; MAXIM, Bruce. Engenharia de software: uma abordagem profissional. 8.ed. Bookman, 2016. E-book. Disponível em: https://integrada.minhabiblioteca.com.br/books/9788580555349

- SOMMERVILLE, Ian. Engenharia de software. 9. ed. São Paulo: Pearson Prentice Hall, 2011. E-book. Disponível em: https://plataforma.bvirtual.com.br/Leitor/Publicacao/2613/epub/0

- LARMAN, Craig. Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e desenvolvimento iterativo. 3. ed Porto Alegre: Bookman, 2007. E-book. Disponível em: https://integrada.minhabiblioteca.com.br/books/9788577800476

Fim