Modelos, Métodos e Técnicas de Engenharia de Software
Visão e análise de projeto Padrões Prática 3 – Visitor (23)

Prof. Osmar de Oliveira Braz Junior

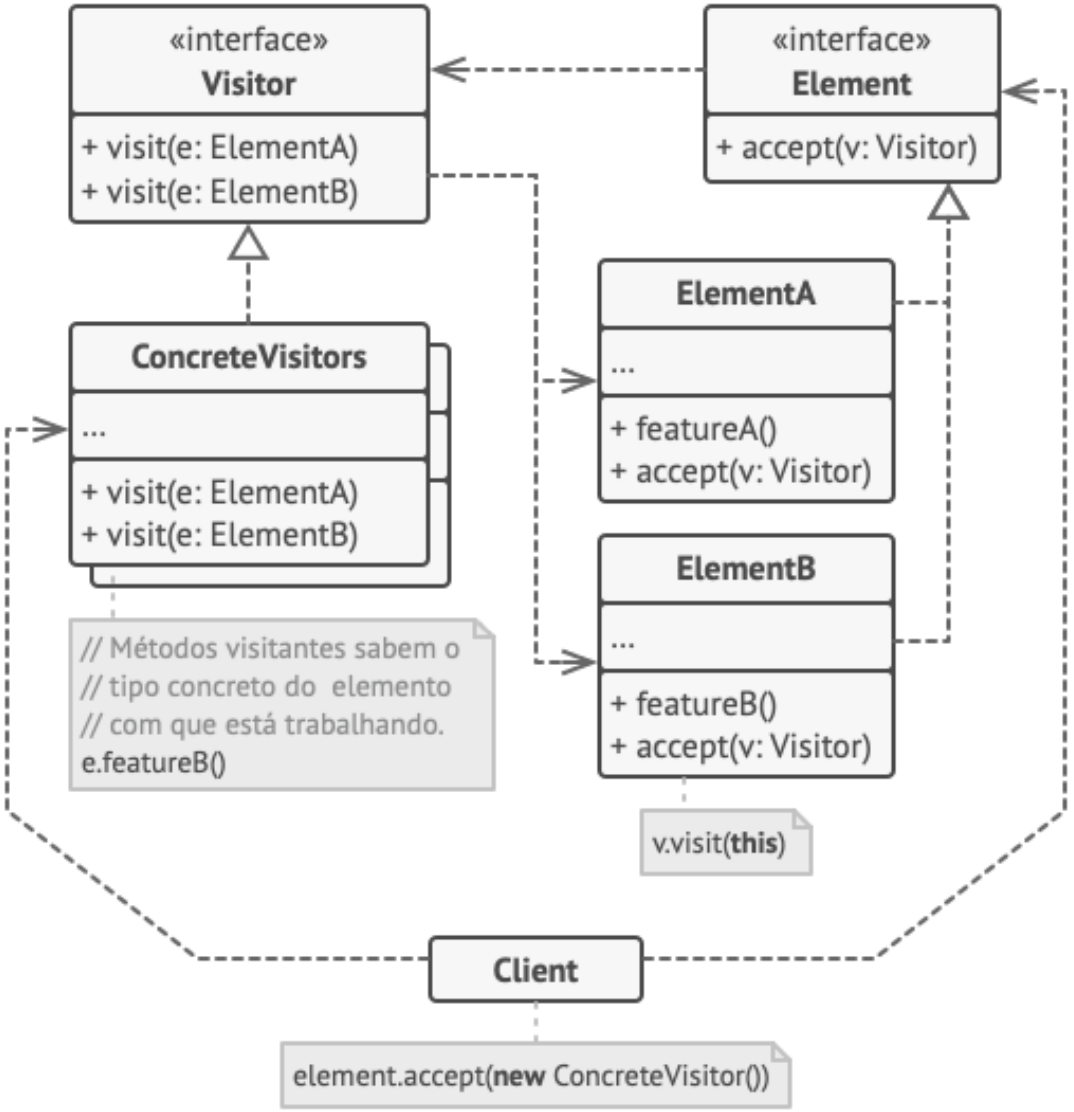
Prof. Richard Henrique de Souza

Objetivos

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

23. Visitor

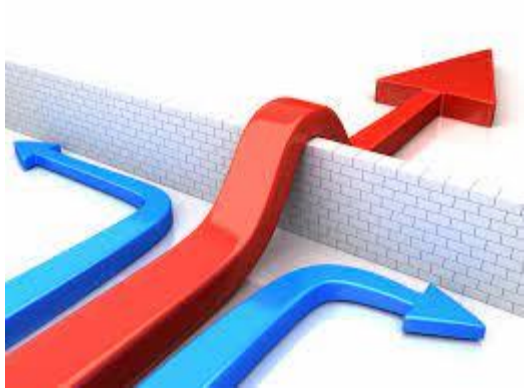
Estrutura



https://refactoring.guru/pt-br/design-patterns/visitor

Importante

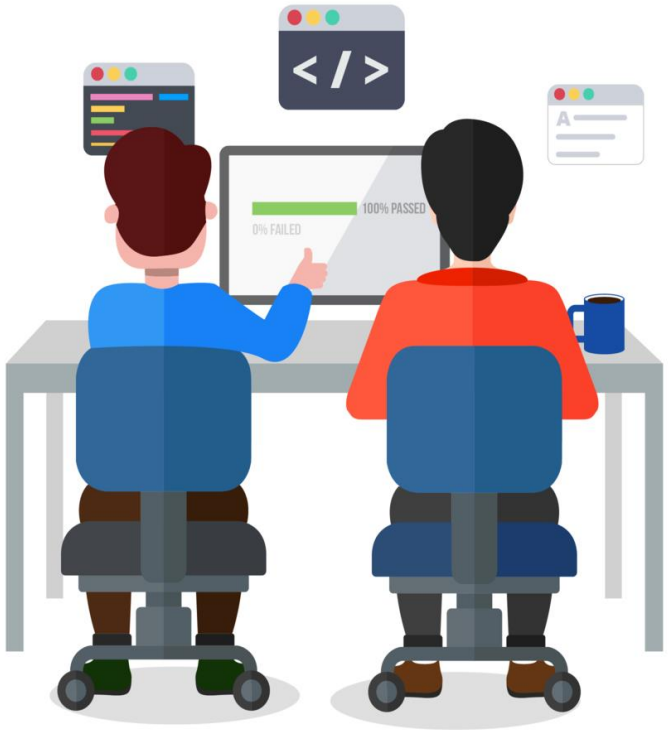
Siga os ROTEIROS !!!



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

23. Visitor
Propósito

- Representar uma operação a ser efetuada em objetos de uma certa classe como outra classe.
- Permite que você defina uma nova operação sem alterar a classe na qual a operação é efetuada.
- Também conhecido como: Visitante

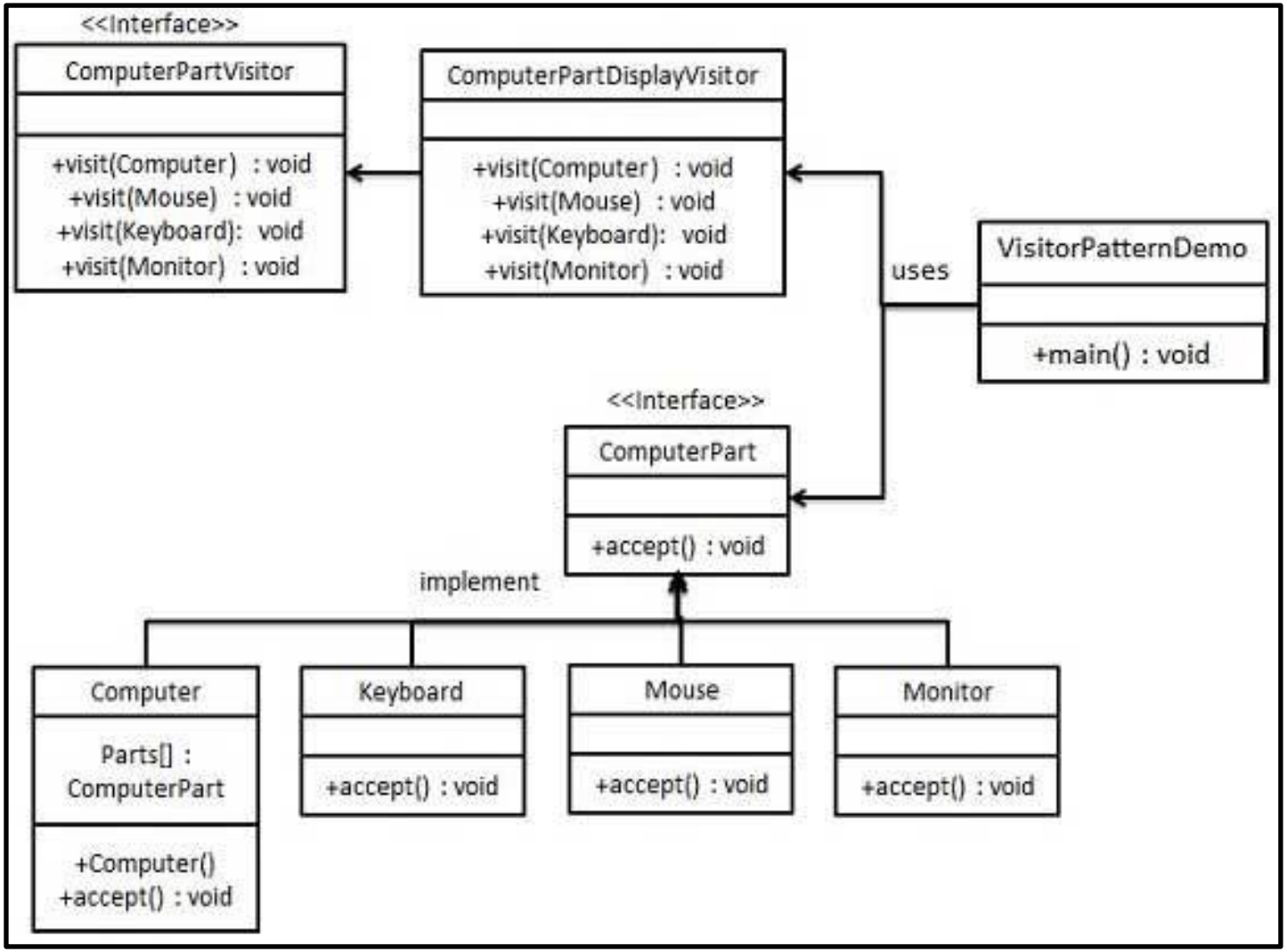
23. Visitor

- Usar este padrão quando…
  - Uma estrutura de objetos contém muitas classes com muitas operações diferentes;
  - Quiser separar as operações dos objetos-alvo, para não “poluir” seu código;
  - O conjunto de objetos-alvo raramente muda, pois cada novo objeto requer novos métodos em todos os visitors.

23. Visitor

- Vantagens e desvantagens
  - Organização:
    - Visitor reúne operações relacionadas.
  - Fácil adicionar novas operações:
    - Basta adicionar um novo Visitor.
  - Difícil adicionar novos objetos:
    - Todos os Visitors devem ser mudados.
  - Transparência:
    - Visite toda a hierarquia transparentemente.
  - Quebra de encapsulamento:
    - Pode forçar a exposição de estrutura interna para que o Visitor possa manipular.

23. Visitor
Passo 1



23. Visitor
Passo 1

Defina uma interface para representar o elemento.

ComputerPart.java

```
public interface ComputerPart {
   public void accept(ComputerPartVisitor computerPartVisitor);
}
```

23. Visitor
Passo 2

Crie classes concretas estendendo a classe acima.

Keyboard.java

```
public class Keyboard implements ComputerPart {

   @Override
   public void accept(ComputerPartVisitor computerPartVisitor) {
      computerPartVisitor.visit(this);
   }
}
```

23. Visitor
Passo 2 - Continuação

Monitor.java

```
public class Monitor implements ComputerPart {

    @Override
    public void accept(ComputerPartVisitor computerPartVisitor) {
        computerPartVisitor.visit(this);
    }
}
```

23. Visitor
Passo 2 - Continuação

Mouse.java

```
public class Mouse implements ComputerPart {

   @Override
   public void accept(ComputerPartVisitor computerPartVisitor) {
      computerPartVisitor.visit(this);
   }
}
```

23. Visitor
Passo 2 - Continuação

Computer.java

```
public class Computer implements ComputerPart {

   ComputerPart[] parts;

   public Computer(){
      parts = new ComputerPart[] {new Mouse(), new Keyboard(), new Monitor()};
   }


   @Override
   public void accept(ComputerPartVisitor computerPartVisitor) {
      for (int i = 0; i < parts.length; i++) {
         parts[i].accept(computerPartVisitor);
      }
      computerPartVisitor.visit(this);
   }
}
```

23. Visitor
Passo 3

Defina uma interface para representar o visitante.

ComputerPartVisitor.java

```
public interface ComputerPartVisitor {
        public void visit(Computer computer);
        public void visit(Mouse mouse);
        public void visit(Keyboard keyboard);
        public void visit(Monitor monitor);
}
```

23. Visitor
Passo 4

Crie visitantes concretos implementando a classe acima.

ComputerPartDisplayVisitor.java

```
public class ComputerPartDisplayVisitor implements ComputerPartVisitor {

   @Override
   public void visit(Computer computer) {
      System.out.println("Displaying Computer.");
   }

   @Override
   public void visit(Mouse mouse) {
      System.out.println("Displaying Mouse.");
   }

   @Override
   public void visit(Keyboard keyboard) {
      System.out.println("Displaying Keyboard.");
   }

   @Override
   public void visit(Monitor monitor) {
      System.out.println("Displaying Monitor.");
   }
}
```

23. Visitor
Passo 5

Use o ComputerPartDisplayVisitor para exibir partes do computador.

VisitorPatternDemo.java

```
public class VisitorPatternDemo {
   public static void main(String[] args) {

      ComputerPart computer = new Computer();
      computer.accept(new ComputerPartDisplayVisitor());
   }
}
```

23. Visitor
Passo 6

- 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