Centro Universitário UniBH
Ciência da Computação
Práticas de Programação
Professor: Lucas Schmidt

Aula Prática - 23 Padrões de Comportamento - Visitor

Intenção

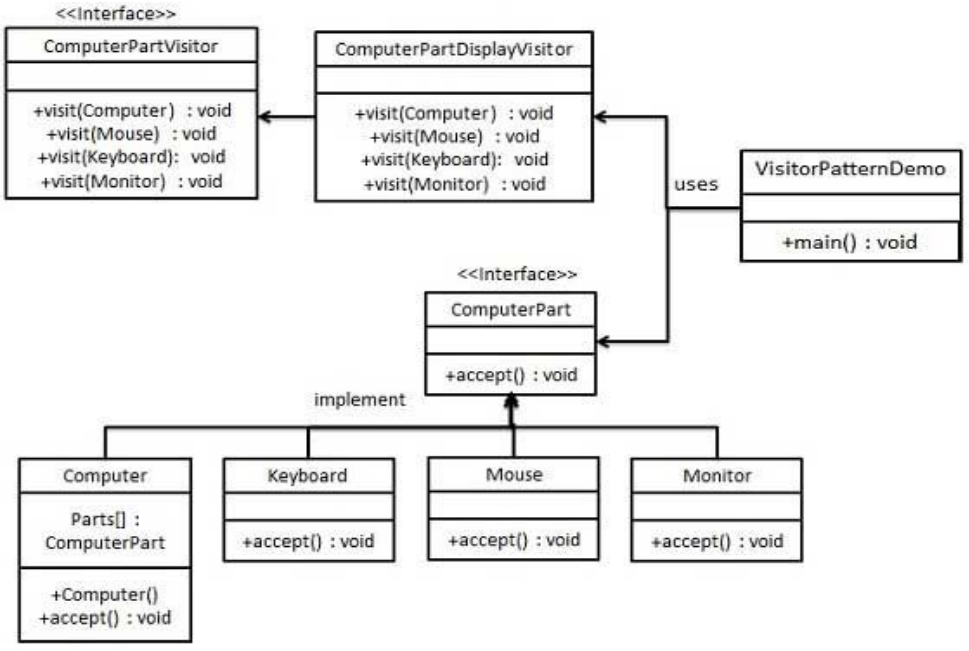
Representar uma operação a ser efetuada em objetos de uma certa classe como outra classe. Permite que você defina uma nova operação sem alterar a classe na qual a operação é efetuada.

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.

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.
  -



Passo 1
Defina uma interface para representar o elemento.

ComputerPart.java

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

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);
   }
}
```

Monitor.java

```
public class Monitor implements ComputerPart {

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

Mouse.java

```
public class Mouse implements ComputerPart {

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

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);
   }
}
```

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);
}
```

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.");
   }
}
```

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());
   }
}
```

Passo 6

Teste sua implementação!