Semana 07

Composite Design Pattern

WebQuest

Introdução

Composite pattern is one of the **Structural design pattern** and is used when we have to represent a part-whole hierarchy. When we need to create a structure in a way that the objects in the structure has to be treated the same way, we can apply composite design pattern.

The Composite Pattern allows you to compose objects into a tree structure to represent the part-whole hierarchy which means you can create a tree of objects that is made of different parts, but that can be treated as a whole one big thing. Composite lets clients to treat individual objects and compositions of objects uniformly, that's the intent of the Composite Pattern.

There can be lots of practical examples of the Composite Pattern. A file directory system, an html representation in java, an XML parser all are well managed composites and all can easily be represented using the Composite Pattern.

Let's understand it with a real life example – A diagram is a structure that consists of Objects such as Circle, Lines, Triangle etc and when we fill the drawing with color (say Red), the same color also gets

applied to the Objects in the drawing. Here drawing is made up of different parts and they all have same operations.

Composite Pattern consists of the following objects:

1. **Base Component** – Base component is the interface for all objects in the composition, client program uses base component to work with the objects in the composition. It can be an interface or an **abstract class** with some methods common to all the objects.
2. **Leaf** – Defines the behaviour for the elements in the composition. It is the building block for the composition and implements base component. It doesn't have references to other Components.
3. **Composite** – It consists of leaf elements and implements the operations in base component.

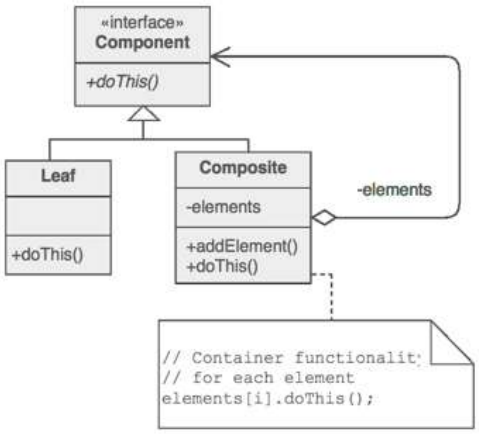
QuickTime™ and a
decompressor
are needed to see this picture.

Tarefa

Conhecer e experimentar o Composite Pattern, implementando um exemplo.

Processo: Fazer pelo menos até o item 4 em sala de aula [35 min]

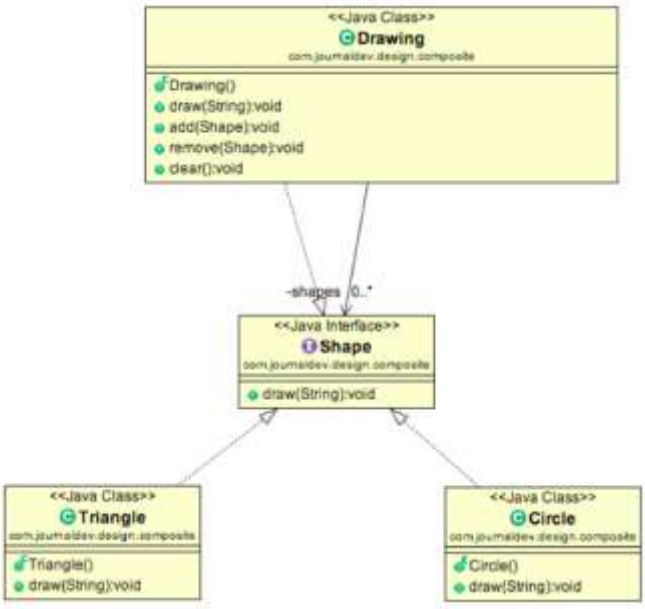
1. Em dupla, entender o padrão com o colega:



2. Como a primeira figura acima pode ser obtida a partir do diagrama de classes do padrão Composite? Sugestão de encaminhamento de reflexão:

    a. Claramente um Composite que está embutido em um Composite pode ter alguma leaf e outr Composite! figura! Ou seja, um Composite é formado de componentes que podem ser Leaf e Composite! Tente desenhar essa situação na forma de classes UML.
    b. Um Composite, por sua vez, é composto de Leafs e Composites! Ou seja, um Composite é composto de zero ou mais componentes do tipo Leaf e Composite! Tente desenhar essa situação na forma de classes UML.
    c. Tente juntar os dois diagramas num só!
    R: O diagrama UML dessa representação será exatamente a imagem descrita no item 1, pois temos uma recursão do Composite contendo elementos do tipo Componente que pode ser implementado de determinada forma para criar a árvore.

3. Examine o diagrama do exemplo, que envolve desenhos compostos de círculos, linhas, triângulos etc., verificando se ele realmente se encaixa no padrão composite. É possível ter mais de um tipo de leaf nesse padrão, como no exemplo?



R: No diagrama UML descrito acima, nota-se o padrão Composite de forma que nossa interface Component equivale a interface Shape, enquanto a classe Drawing é o exemplo de uma classe Composite, e Circle e Triangle são exemplos de classe Leaf. Dessa forma, pode-se ter mais de uma classe Leaf para justamente possibilitar a criação de uma árvore.

4. Dadas a interface Shape e a classe Triangle como exemplos, implemente as classes faltantes e teste o sistema com a classe de teste TestComposite Pattern fornecida abaixo; não esqueça de incluir o método clear () na classe Drawing, que remove todos os seus shapes num dado momento.

```
public interface Shape {
        public void draw(String fillColor);
}
public class Triangle implements Shape {
        @Override
        public void draw(String fillColor) {
System.out.println("Drawing Triangle with color "+fillColor);
        }
}
public class TestCompositePattern {
  public static void main(String[] args) {
        Shape tri = new Triangle();
        Shape tri1 = new Triangle();
        Shape cir = new Circle();
        Drawing drawing = new Drawing();
        drawing.add(tri1);
        drawing.add(tri1);
        drawing.add(cir);
        drawing.draw("Red");
        drawing.clear();
        drawing.add(tri);
        drawing.add(cir);
        drawing.draw("Green");
  }
}
```

```
Output of the above program is:
Drawing Triangle with color Red
Drawing Triangle with color Red
Drawing Circle with color Red
Clearing all the shapes from drawing
Drawing Triangle with color Green
Drawing Circle with color Green
```

5. Redesenhar a mão livre mesmo, se desejar, um novo diagrama de classes do exemplo colocando no formato da figura do padrão Composite do item 1), em que a Interface Shape fique acima das classes, as classes todas ficam no mesmo nível e a classe Drawing fica à direita das outras classes.
6. Aplicação ao seguinte problema: "Algumas vezes um componente agregado é composto de outro componente agregado. Por exemplo, poderíamos ter um objeto figura que é composto de algum texto, linhas, retângulos e outra figura. Claramente, uma figura que está embutida em uma figura pode ter algum texto, linhas, retângulos ou outra figura. Em outras palavras, um componente de um objeto desse tipo pode ser apenas um componente simples, tal como um trecho de texto, uma linha ou um retângulo ou pode ser uma agregação desses elementos, como uma figura.

→Usando UML 2.0, desenhe um diagrama de classes no padrão Composite que represente essa situação. Dê nomes apropriados a todas as classes envolvidas, considerando que o Base Component (que deve receber um nome apropriado também) neste caso, diferentemente do exemplo acima, deverá ser uma classe abstrata; defina nessa classe um método comum a todos as classes, abstratas ou concretas; e, finalmente, defina a classe do Composite (que deve receber um nome apropriado também) com base no código arquitetural dessa classe em relação à classe do Base Component".

7. Quais dos princípios SOLID são promovidos por este DP? Forneça uma explicação sucinta para cada princípio promovido.

S – Single-responsiblity principle

O – Open-closed principle

uma forma de pensar neste princípio: imagine que deseja-se implementer um framework, que será disponibilizado apenas compilado, sem acesso ao código fonte. O que fica definido (fechado para modificação) e o que fica passível de ser estendido (aberto para extensão)?.

L – Liskov substitution principle

I – Interface segregation principle

D – Dependency Inversion Principle

8. Gerar documento com a resposta dos item 2 a 6 e postá-lo, com os nomes dos membros do dupla, na atividade correspondente do TIDIA num dos membros do dupla.

Recursos

1. https://sourcemaking.com/design_patterns/composite