javapapers

Composite Design Pattern

When we want to represent part-whole hierarchy, use tree structure and compose objects. We know tree structure what a tree structure is and some of us don't know what a part-whole hierarchy is. A system consists of subsystems or components. Components can further be divided into smaller components. Further smaller components can be divided into smaller elements. This is a part-whole hierarchy.

Everything around us can be a candidate for part-whole hierarchy. Human body, a car, a computer, lego structure, etc. A car is made up of engine, tyre, … Engine is made up of electrical components, valves, … Electrical components is made up of chips, transistor, … Like this a component is part of a whole system. This hierarchy can be represented as a tree structure using composite design pattern.

"Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly." is the intent by GoF.

Real World Example

In this article, let us take a real world example of part-whole hierarchy and use composite design pattern using java. As a kid, I have spent huge amount of time with lego building blocks. Last week I bought my son an assorted kit lego and we spent the whole weekend together building structures.

Let us consider the game of building blocks to practice composite pattern. Assume that our kit has only three unique pieces (1, 2 and 4 blocks) and let us call these as primitive blocks as they will be the end nodes in the tree structure. Objective is to build a house and it will be a step by step process. First using primitive blocks, we should construct multiple windows, doors, walls, floor and let us call these structures. Then use all these structure to create a house.

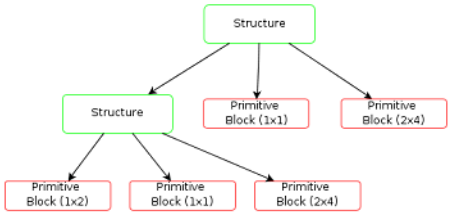
Primitive blocks combined together gives a structure. Multiple structures assembled together gives a house.

Important Points

- Importance of composite pattern is, the group of objects should be treated similarly as a single object.
- Manipulating a single object should be as similar to manipulating a group of objects. In sync with our example, we join primitive blocks to create structures and similarly join structures to create house.
- Recursive formation and tree structure for composite should be noted.
- Clients access the whole hierarchy through the components and they are not aware about if they are dealing with leaf or composites.

Tree for Composite

When we get a recursive structure the obvious choice for implementation is a tree. In composite design pattern, the part-whole hierarchy can be represented as a tree. Leaves (end nodes) of a tree being the primitive elements and the tree being the composite structure.



Uml Design for Composite Pattern

Component: (structure)

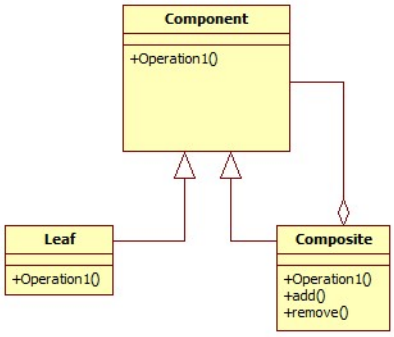
1. Component is at the top of hierarchy. It is an abstraction for the composite.
2. It declares the interface for objects in composition.
3. (optional) defines an interface for accessing a component's parent in the recursive structure, and implements it if that's appropriate.

Leaf: (primitive blocks)

1. The end nodes of the tree and will not have any child.
2. Defines the behaviour for single objects in the composition

Composite: (group)

1. Consists of child components and defines behaviour for them
2. Implements the child related operations.



Composite Pattern Implementation

```
package com.javapapers.designpattern.composite;

public class Block implements Group {

    public void assemble() {
        System.out.println("Block");
    }
}
```

```
package com.javapapers.designpattern.composite;

public interface Group {
    public void assemble();
}
```

```
package com.javapapers.designpattern.composite;

import java.util.ArrayList;
import java.util.List;

public class Structure implements Group {
 // Collection of child groups.
 private List groups = new ArrayList();

 public void assemble() {
  for (Group group : groups) {
   group.assemble();
  }
 }

 // Adds the group to the structure.
 public void add(Group group) {
  groups.add(group);
 }

 // Removes the group from the structure.
 public void remove(Group group) {
  groups.remove(group);
 }
}
```

```
package com.javapapers.designpattern.composite;

public class ImplementComposite {
  public static void main(String[] args) {
        //Initialize three blocks
        Block block1 = new Block();
        Block block2 = new Block();
        Block block3 = new Block();

        //Initialize three structure
        Structure structure = new Structure();
        Structure structure1 = new Structure();
        Structure structure2 = new Structure();

        //Composes the groups
        structure1.add(block1);
        structure1.add(block2);

        structure2.add(block3);

        structure.add(structure1);
        structure.add(structure2);

        structure.assemble();
    }
}
```

Usage of Composite Design Pattern in Sun/Oracle JDK

- java.awt.Container#add(Component) – in awt, we have containers and components – a classic implementation
- javax.faces.component.UIComponent#getChildren()

This Design Patterns tutorial was added on 13/05/2012.

« System.out.println

Java Timer »

Comments on "Composite Design Pattern"

rajmohan says: *16/08/2014 at 5:26 pm*

Thanks joe,for clear understanding of composite view….

Reply

---

rajeesh says: *17/06/2014 at 11:32 pm*

Simple and clear. thank you.

Reply

Lingareddy says: *11/06/2014 at 1:46 pm*

Thanks Joe

Reply

Peter says: *16/02/2014 at 1:46 pm*

Will it be recursive if

structure s = new structure();

s.add(s); ?

Reply

Eswar1221 says: *30/11/2013 at 10:26 pm*

Nice example,But As per GOF., Component class (in our case Group interface) should have methods add() and remove(). If not client can not access add or remove methods. In the above example Client can access only operation () method. Let me know if I misunderstand it.

Reply

Eswar says: *30/11/2013 at 10:25 pm*

Nice example,But As per GOF., Component class (in our case Group interface) should have methods add() and remove(). If not client can not access add or remove methods. In the above example Client can access only operation () method. Let me know if I misunderstand it.

Reply

Anonymous says: *08/11/2013 at 9:55 pm*

Joe, blog without no interaction with users (those who comment), is not a blog at all

Reply

Anonymous says: *29/05/2013 at 12:30 pm*

Very Nice explanation indeed !.
I have a question with regards to the UML diagram show,if you inherit the Composite from a Component ,then why has the sign of aggregation(diamond)been put next to “Composite”.
Ideally the sign of aggregation comes next to the parent class and in this case Component being the parent class .
Please explain.
Thanks.
Regards….

Reply

Samruddhi says: *17/04/2013 at 4:38 pm*

Lovely

Reply

Sinss says: *14/04/2013 at 9:42 pm*

nice blog..
waiting for behavioral patterns..

Reply

Anonymous says: *07/04/2013 at 10:56 pm*

Hello!
you said in the beginning that Component is the structure and Composite is the group, but I think in the example you do it the otherway round, when you write the group is the interface?

Reply

Anonymous says: *17/03/2013 at 6:13 am*

I can’t quite get what’s happening in the definition of the Structure class where you make a list of “Group” items “private List groups = new ArrayList();”?

Could you please explain what’s happening and if it is fine to define a list of an interface type without specifying which implementation to use?

Reply

Hamdi Mohamed Baligh says: *20/02/2013 at 11:25 pm*

Great Job :)

Reply

Said Naeem Shah says: *15/02/2013 at 12:05 pm*

Nice but example chosen is not so good and practical with composite

Reply

Anonymous says: *17/01/2013 at 4:54 pm*

can you pls give some couple of scenarios where exactly this can be implemented.You have been silent for the above comments, can I expect your response soon? it would be helpful to us

Reply

Kathy says: *12/12/2012 at 5:42 am*

Thanks. It helps to see different applications for these design patterns.

Reply

Neha says: *23/11/2012 at 1:48 pm*

Clear and concise. :-)

Reply

---

soothing says: *22/11/2012 at 3:53 pm*

really nice explanation

Reply

---

michael says: *18/10/2012 at 9:22 pm*

I have a question. We are creating a beverage vending machine as an assignment. In this, there might be 2 different kinds of coffees. Each one , I would think would be a composite, whose leafs might be a cup, sugar, cream etc. So you might have a regular coffee with sugar cream and a decaf composite coffee with sugar and cream but in a tree, that would be duplicating sugar units, for example, several times over, which I would think is not what you would want. Can the same leaf branch off of different components?

Reply

---

saurabh banerjee says: *19/09/2012 at 12:41 pm*

nice one

Reply

Anonymous says: *23/07/2012 at 4:51 pm*

Good one !!

Reply

Dharshan says: *14/07/2012 at 3:12 pm*

Nice article

Reply

Tatyana says: *27/06/2012 at 9:32 pm*

Joe, what do you expect to have in output after you run ImplementComposit example?

Reply

Tatyana says: *27/06/2012 at 9:28 pm*

Hi Joe,
if you run your example, what do you excpect to have in output?

Reply

Paramesh says: *26/06/2012 at 7:53 pm*

Wonderful blog. Thanks Joe !!

Reply

haranath says: *11/06/2012 at 7:42 pm*

Nice one.. :)

Reply

vishal says: *09/06/2012 at 1:18 am*

Thankx joe

Reply

Pallavi says: *31/05/2012 at 7:55 pm*

Thanks Joe :)

Reply

Anandan says: *23/05/2012 at 5:40 pm*

its really nice explanation to understand the pattern

Reply

goutham says: *21/05/2012 at 11:41 pm*

hai joe…
what is the difference between POJO and java bean… i know some what basic difference but i am not satisfied with that please explain briefly…..
thank you….

Reply

Forouz Parsi says: *20/05/2012 at 7:29 pm*

Thanks Joe ,
You are really good in explanation . Using simple language and good examples …
Helped me a lot for tomorrow’s exam.
Keep it up

Reply

Anonymous says: *20/05/2012 at 12:17 am*

Thanks for the article, but does your structure diagram and the implementation of the code match?

Reply

manoj says: *19/05/2012 at 1:34 pm*

I really did not understand what flaw are we trying to eradicate here. Don't we do coding like this even if we are not aware of this design pattern. I mean why do we call it a pattern and not just a requirement that anyhow will be done that way . What was the repeated mistake that this particular approach has been suggested ,documented and recommended for?

Reply

Ashish says: *17/05/2012 at 4:19 pm*

Thanks Joe…

Reply

Ashish says: *17/05/2012 at 3:53 pm*

Awesome example & explanation. Thanks Joe i m a regular reader of your blogs and every time i came to know a new thing even though i m familiar with it.

Reply

Anonymous says: *17/05/2012 at 3:53 pm*

Awesome example & explanation. Thanks Joe i m a regular reader of your blogs and every time i came to know a new thing even though i m familiar with it.

Reply

Pradeepan says: *15/05/2012 at 3:52 pm*

Hi Joe, Great Explanation and good Presentation.Keep it up Excellent…Thank a lot Joe

Reply

Jayaraman palani says: *15/05/2012 at 12:55 pm*

hi joe its good articles and also I am expecting scenarios like where we can implent this type of Design pattern

Reply

Lalit says: *15/05/2012 at 10:12 am*

Awesome ……. great explanation

Reply

Harpreet says: *14/05/2012 at 9:18 pm*

Keep up the good work joe

Reply

kartheek says: *14/05/2012 at 7:18 pm*

excellent information for a genuine java developers….thanks mate and pls carry on and give some more information….good night buddy….

Reply

Sandeep Singh says: *14/05/2012 at 2:26 pm*

Thanks Joe

Reply

Farrukh says: *14/05/2012 at 1:04 pm*

Can you, if sometime, to apply this pattern to JTree in next your post, or I did not understand the essence of this pattern.Thanks Joe.

Reply

---

krishnamurthy says: *14/05/2012 at 12:42 pm*

super Joe……Excelent

Reply

---

rao says: *14/05/2012 at 12:30 pm*

can you specify when to use this pattern?

Reply

---

vikas says: *14/05/2012 at 11:44 am*

really great explanation.

Reply

Anonymous says: *13/05/2012 at 3:07 pm*

Thanks joe..

Reply

Your Comment

Name

Email

Post Comment

TUTORIAL MENU

Java

Android

Design Patterns

Creational Design Patterns

Structural Design Patterns

Adapter Design Pattern

Bridge Design Pattern

Composite Design Pattern

Decorator Design Pattern

Facade Design Pattern

Flyweight Design Pattern

Proxy Design Pattern

Behavioral Design Patterns

Spring

Web Services

Servlet

Java Android Design Patterns Spring Web Services Servlet

Site Map