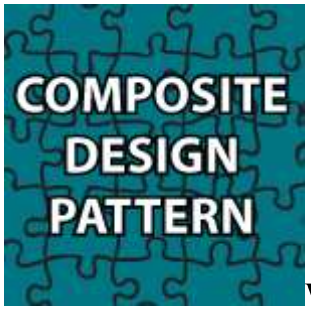
Composite Design Pattern Tutorial

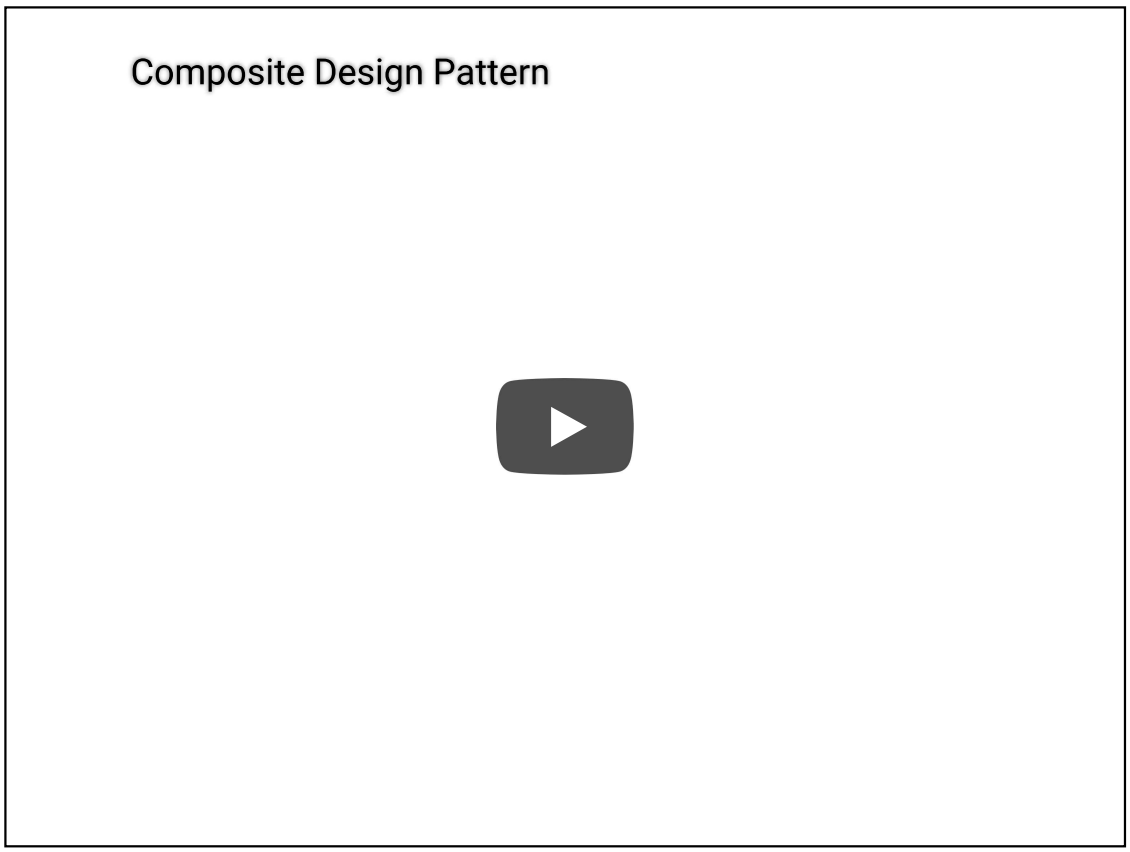
Posted by Derek Banas on Oct 10, 2012 in Java Video Tutorial | 24 comments



Welcome to my Composite Design Pattern Tutorial! The Composite design pattern is used to structure data into its individual parts as well as represent the inner workings of every part of a larger object.

The composite pattern also allows you to treat both groups of parts in the same way as you treat the parts polymorphically. This allows your objects to maximize complexity while also remaining dynamic.

All of the code follows the video to help you learn. It is heavily commented.



If you think others would benefit from this video, it helps to tell Google with a click [googleplusone]

Share if you no others may like this

Code from the Video

SONGCOMPONENT.JAVA

```
01 // This acts as an interface for every Song (Leaf)
02 // and SongGroup (Composite) we create
03
04 public abstract class SongComponent {
05
06     // We throw UnsupportedOperationException so that if
07     // it doesn't make sense for a song, or song group
08     // to inherit a method they can just inherit the
09     // default implementation
10
11     // This allows me to add components
12
13     public void add(SongComponent newSongComponent) {
14
15         throw new UnsupportedOperationException();
16
17     }
18
19     // This allows me to remove components
20
```

```
21          public void remove(SongComponent newSongComponent) {
22
23              throw new UnsupportedOperationException();
24
25      }
26
27      // This allows me to get components
28
29      public SongComponent getComponent(int componentIndex) {
30
31          throw new UnsupportedOperationException();
32
33      }
34
35      // This allows me to retrieve song names
36
37      public String getSongName() {
38
39          throw new UnsupportedOperationException();
40
41      }
42
43      // This allows me to retrieve band names
44
45      public String getBandName() {
46
47          throw new UnsupportedOperationException();
48
49      }
50
51      // This allows me to retrieve release year
52
53      public int getReleaseYear() {
54
55          throw new UnsupportedOperationException();
56
57      }
58
59      // When this is called by a class object that extends
60      // SongComponent it will print out information
61      // specific to the Song or SongGroup
62
63      public void displaySongInfo() {
64
65          throw new UnsupportedOperationException();
66
67      }
68
69  }
```

SONGGROUP.JAVA

```
01  import java.util.ArrayList;
02  import java.util.Iterator;
```

```
03
04 public class SongGroup extends SongComponent {
05
06     // Contains any Songs or SongGroups that are added
07     // to this ArrayList
08
09     ArrayList songComponents = new ArrayList();
10
11     String groupName;
12     String groupDescription;
13
14     public SongGroup(String newGroupName, String newGroupDescription){
15
16         groupName = newGroupName;
17         groupDescription = newGroupDescription;
18
19     }
20
21     public String getGroupName() { return groupName; }
22     public String getGroupDescription() { return groupDescription; }
23
24     public void add(SongComponent newSongComponent) {
25
26         songComponents.add(newSongComponent);
27
28     }
29
30     public void remove(SongComponent newSongComponent) {
31
32         songComponents.remove(newSongComponent);
33
34     }
35
36     public SongComponent getComponent(int componentIndex) {
37
38         return (SongComponent)songComponents.get(componentIndex);
39
40     }
41
42     public void displaySongInfo(){
43
44         System.out.println(getGroupName() + " " +
45                 getGroupDescription() + "\n");
46
47         // Cycles through and prints any Songs or SongGroups added
48         // to this SongGroups ArrayList songComponents
49
50         Iterator songIterator = songComponents.iterator();
51
52         while(songIterator.hasNext()) {
53
54             SongComponent songInfo = (SongComponent) songIterator.next();
55
56             songInfo.displaySongInfo();
```

```
57
58          }
59
60      }
61
62
63  }
```

SONG.JAVA

```
01  public class Song extends SongComponent {
02
03      String songName;
04      String bandName;
05      int releaseYear;
06
07      public Song(String newSongName, String newBandName, int
    newReleaseYear){
08
09          songName = newSongName;
10          bandName = newBandName;
11          releaseYear = newReleaseYear;
12
13      }
14
15      public String getSongName() { return songName; }
16      public String getBandName() { return bandName; }
17      public int getReleaseYear() { return releaseYear; }
18
19      public void displaySongInfo(){
20
21          System.out.println(getSongName() + " was recorded by " +
22                  getBandName() + " in " + getReleaseYear());
23
24      }
25
26  }
```

DISCJOCKEY.JAVA

```
01  public class DiscJockey{
02
03      SongComponent songList;
04
05      // newSongList contains every Song, SongGroup,
06      // and any Songs saved in SongGroups
07
08      public DiscJockey(SongComponent newSongList){
09
10          songList = newSongList;
11
12      }
13
14      // Calls the displaySongInfo() on every Song
```

```
    // or SongGroup stored in songList

    public void getSongList(){

        songList.displaySongInfo();

    }

}
```

SONGLISTGENERATOR.JAVA

```
public class SongListGenerator {

    public static void main(String[] args){

        SongComponent industrialMusic =
                new SongGroup("Industrial",
                        "is a style of experimental music that draws on
transgressive and provocative themes");

        SongComponent heavyMetalMusic =
                new SongGroup("\nHeavy Metal",
                        "is a genre of rock that developed in the late
1960s, largely in the UK and in the US");

        SongComponent dubstepMusic =
                new SongGroup("\nDubstep",
                        "is a genre of electronic dance music that
originated in South London, England");

        // Top level component that holds everything

        SongComponent everySong = new SongGroup("Song List", "Every Song
Available");

        // Composite that holds individual groups of songs
        // This holds Songs plus a SongGroup with Songs

        everySong.add(industrialMusic);

        industrialMusic.add(new Song("Head Like a Hole", "NIN", 1990));
        industrialMusic.add(new Song("Headhunter", "Front 242", 1988));

        industrialMusic.add(dubstepMusic);

        dubstepMusic.add(new Song("Centipede", "Knife Party", 2012));
        dubstepMusic.add(new Song("Tetris", "Doctor P", 2011));

        // This is a SongGroup that just holds Songs

        everySong.add(heavyMetalMusic);

        heavyMetalMusic.add(new Song("War Pigs", "Black Sabath", 1970));
```

```
39          heavyMetalMusic.add(new Song("Ace of Spades", "Motorhead", 1980));
40
41          DiscJockey crazyLarry = new DiscJockey(everySong);
42
43          crazyLarry.getSongList();
44
45      }
46
47  }
```

24 Responses to “Composite Design Pattern Tutorial”

1.  *Ellle* says:
January 4, 2013 at 3:53 am

   This website rocks! There is so much good information here. Thanks for making JAVA easy to understand!

   Reply

   -  *admin* says:
   January 4, 2013 at 11:34 am

     You’re very welcome 🙂 I do my best. Thank you for taking the time to tell me you liked it

     Reply

2.  *boris* says:
March 29, 2013 at 1:04 am

   thank you very much!

   Reply

3.  *Ally* says:
April 30, 2013 at 8:42 pm

   line 29 of SongListGenerator: industrialMusic.add(dubstepMusic);
   should this read everySong.add(dubstepMusic);?

   these are awesome – i dont know what i’d do without your videos

   Reply

   -  *Derek Banas* says:
   May 1, 2013 at 4:39 pm

Thank you 🙂 No, industrialMusic.add(dubstepMusic) is correct. I may have confused you because I considered DubStep to be a version of industrial. That may not be true. I don't know because I'm definitely not a music expert

Reply

4.  *Hadar* says:
May 29, 2013 at 8:04 am

Hi Derek

And thanks for the great tutorials

I have a Q: How come a "throws" statement, is not required from methods of the abstract class SongComponent ?

Reply

   -  *Derek Banas* says:
   May 31, 2013 at 11:42 am

   You're welcome 🙂 Yes, I should have caught that error. Sorry about that

   Reply

5.  *Pedrito* says:
June 11, 2013 at 8:30 pm

Dude, I'm from Chile. You're helping the entire world with your vids. How awesome is that?
Take care bro and thanks

Reply

   -  *Derek Banas* says:
   June 14, 2013 at 5:54 am

   Thank you very much 🙂 It is extremely gratifying to help and speak with people all over the world!

   Reply

6.  *Duggs* says:
July 9, 2013 at 11:18 pm

awsume tutorial man!! u rock thnk u very mch

Reply

-  *Derek Banas* says:
  July 15, 2013 at 7:33 am

  Thank you 🙂

  Reply

7.  *Idan* says:
   July 27, 2013 at 3:27 am

   Great stuff!
   Thank you very much for these videos. I learn a lot.

   Reply

   -  *Derek Banas* says:
     July 27, 2013 at 3:34 pm

     Thank you very much 🙂 You're welcome

     Reply

8.  *Dylan* says:
   September 3, 2013 at 1:08 am

   This is awesome, great work you did with these tutorials 🙂

   Keep it on !

   Reply

   -  *Derek Banas* says:
     September 3, 2013 at 8:05 pm

     Thank you very much 🙂

     Reply

9.  *nizam* says:
   December 7, 2013 at 9:17 am

   Great work !. Thank you very much for the tutorials 🙂
   learning a lot !.

   Reply

-  *Derek Banas* says:
  December 7, 2013 at 1:30 pm

  You're very welcome 🙂 I'm happy that you are enjoying them.

  Reply

10.  *Mojo Jojo* says:
    January 12, 2014 at 5:10 pm

    Great work! Thank you!

    I know it can be many leaf classes, as long as they implement the same interface. What is not clear to me is if can it be many composite classes. It's ok to have different ways of grouping composite/leaf objects? Can you think of an example? (I can't 😛)

    Reply

11.  *Varun Risbud* says:
    October 26, 2014 at 8:18 pm

    Hi Derek,

    Really nice tutorial. This explains the Pattern and code so well. Thanks a lot!

    Thanks & Regards,
    Varun

    Reply

    -  *Derek Banas* says:
      October 29, 2014 at 8:02 am

      Thank you 🙂 I'm happy I could help

      Reply

12.  *Rafael Angelo* says:
    October 30, 2014 at 11:40 am

    I'm from Brasil and your video help me a lot in my work school. Thank you.

    Reply

    -  *Derek Banas* says:

October 31, 2014 at 5:17 pm

Thank you very much 🙂 I'm glad I could help

Reply

13.  *Thiago F. Oliveira* says:
December 10, 2014 at 11:33 am

Hey Derek, first of all, thank you so much.
But, i'm a little bit confused with this pattern.
Can you create a UML Class Diagram with this example and with an UML Object Diagram? Cause I don't know how to represents "Heavey Metal" "Industrial" and "Dubstep" in the diagram.
Thanks one more time!

🙂

Reply

Trackbacks/Pingbacks

1. Design Patterns Cheat Sheet by Richard Sumilang - […] Composite Design Pattern: Allows you to attach individual objects and a composition of objects uniformly. For example, imagine a folder…

Leave a Reply

Your email address will not be published.

Comment

Name

Email

Website

Submit Comment

Search

Search

Help Me Make Free Education

Donate Crypto

Social Networks

Facebook

YouTube

Twitter

LinkedIn

Buy me a Cup of Coffee

"Donations help me to keep the site running. One dollar is greatly appreciated." - (Pay Pal Secured)

My Facebook Page

Archives

- January 2018
- December 2017
- November 2017
- October 2017
- September 2017
- August 2017
- July 2017
- June 2017
- May 2017
- April 2017
- March 2017
- February 2017
- January 2017
- December 2016
- November 2016
- October 2016
- September 2016
- August 2016
- July 2016
- June 2016
- May 2016
- April 2016
- March 2016
- February 2016
- January 2016
- December 2015
- November 2015
- October 2015
- September 2015
- August 2015
- July 2015
- June 2015
- May 2015
- April 2015
- March 2015
- February 2015
- January 2015
- December 2014
- November 2014
- October 2014
- September 2014
- August 2014
- July 2014
- June 2014
- May 2014
- April 2014
- March 2014
- February 2014
- January 2014
- December 2013
- November 2013
- October 2013
- September 2013
- August 2013

- July 2013
- June 2013
- May 2013
- April 2013
- March 2013
- February 2013
- January 2013
- December 2012
- November 2012
- October 2012
- September 2012
- August 2012
- July 2012
- June 2012
- May 2012
- April 2012
- March 2012
- February 2012
- January 2012
- December 2011
- November 2011
- October 2011
- September 2011
- August 2011
- July 2011
- June 2011
- May 2011
- April 2011
- March 2011
- February 2011
- January 2011
- December 2010
- November 2010
- October 2010
- September 2010
- August 2010
- July 2010
- June 2010
- May 2010
- April 2010
- March 2010
- February 2010
- January 2010
- December 2009