Bridge Design Pattern Tutorial

Posted by Derek Banas on Oct 1, 2012 in Java Video Tutorial | 24 comments

Welcome to my Bridge Design Pattern tutorial. If you ever wanted to build a group of classes that slowly added functionality from one class to the next, this is the design pattern for you.

There seems to be a lot of confusion in regards to what constitutes a bridge design pattern in both books as well as online. In this tutorial I will show you how it was explained to me and the ways in which I have found it is useful.

The code after the video goes into greater detail if you'd like to check that out.

If you like videos like this, it helps to tell Google [googleplusone]

Sharing is nice

Code From the Video

ENTERTAINMENTDEVICE.JAVA

```
01 // Implementor
02 // With the Bridge Design Pattern you create 2 layers of abstraction
03 // In this example I'll have an abstract class representing
04 // different types of devices. I also have an abstract class
05 // that will represent different types of remote controls
06
07 // This allows me to use an infinite variety of devices and remotes
08
09 abstract class EntertainmentDevice {
10
11     public int deviceState;
12
13     public int maxSetting;
14
15     public int volumeLevel = 0;
16
17     public abstract void buttonFivePressed();
18
19     public abstract void buttonSixPressed();
20
```

```
    public void deviceFeedback() {

        if(deviceState > maxSetting || deviceState < 0) { deviceState = 0;
}

        System.out.println("On Channel " + deviceState);

    }

    public void buttonSevenPressed() {

        volumeLevel++;

        System.out.println("Volume at: " + volumeLevel);

    }

    public void buttonEightPressed() {

        volumeLevel--;

        System.out.println("Volume at: " + volumeLevel);

    }

}
```

TVDEVICE.JAVA

```
// Concrete Implementor

// Here is an implementation of the EntertainmentDevice
// abstract class. I'm specifying what makes it different
// from other devices

public class TVDevice extends EntertainmentDevice {

    public TVDevice(int newDeviceState, int newMaxSetting){

        deviceState = newDeviceState;

        maxSetting = newMaxSetting;

    }

    public void buttonFivePressed() {

        System.out.println("Channel Down");

        deviceState--;

    }

    public void buttonSixPressed() {
```

```
26
27          System.out.println("Channel Up");
28
29          deviceState++;
30
31      }
32
33  }
```

REMOTEBUTTON.JAVA

```
01  // Abstraction
02
03  // This is an abstract class that will represent numerous
04  // ways to work with each device
05
06  public abstract class RemoteButton{
07
08      // A reference to a generic device using aggregation
09
10      private EntertainmentDevice theDevice;
11
12      public RemoteButton(EntertainmentDevice newDevice){
13
14          theDevice = newDevice;
15
16      }
17
18      public void buttonFivePressed() {
19
20          theDevice.buttonFivePressed();
21
22      }
23
24      public void buttonSixPressed() {
25
26          theDevice.buttonSixPressed();
27
28      }
29
30      public void deviceFeedback(){
31
32          theDevice.deviceFeedback();
33
34      }
35
36      public abstract void buttonNinePressed();
37
38  }
```

TVREMOTEMUTE.JAVA

```
01  // Refined Abstraction
02
```

```
// If I decide I want to further extend the remote I can

public class TVRemoteMute extends RemoteButton{

    public TVRemoteMute(EntertainmentDevice newDevice) {
        super(newDevice);
    }

    public void buttonNinePressed() {

        System.out.println("TV was Muted");

    }

}
```

TVREMOTEPAUSE.JAVA

```
// Refined Abstraction

// If I decide I want to further extend the remote I can

public class TVRemotePause extends RemoteButton{

    public TVRemotePause(EntertainmentDevice newDevice) {
        super(newDevice);
    }

    public void buttonNinePressed() {

        System.out.println("TV was Paused");

    }

}
```

TESTTHEREMOTE.JAVA

```
public class TestTheRemote{

    public static void main(String[] args){

        RemoteButton theTV = new TVRemoteMute(new TVDevice(1, 200));

        RemoteButton theTV2 = new TVRemotePause(new TVDevice(1, 200));

        // HOMEWORK --------------

        RemoteButton theDVD = new DVDRemote(new DVDDevice(1,14));

        // ----------------------

        System.out.println("Test TV with Mute");

```

```
        theTV.buttonFivePressed();
        theTV.buttonSixPressed();
        theTV.buttonNinePressed();

        System.out.println("\nTest TV with Pause");

        theTV2.buttonFivePressed();
        theTV2.buttonSixPressed();
        theTV2.buttonNinePressed();
        theTV2.deviceFeedback();

        // HOMEWORK

        System.out.println("\nTest DVD");

        theDVD.buttonFivePressed();
        theDVD.buttonSixPressed();
        theDVD.buttonNinePressed();
        theDVD.buttonNinePressed();

    }

}
```

HOMEWORK

DVDDEVICE.JAVA

```
// Concrete Implementor

// Here is an implementation of the EntertainmentDevice
// abstract class. I'm specifying what makes it different
// from other devices

public class DVDDevice extends EntertainmentDevice {

    public DVDDevice(int newDeviceState, int newMaxSetting){

        super.deviceState = newDeviceState;

        super.maxSetting = newMaxSetting;

    }

    public void buttonFivePressed() {

        System.out.println("DVD Skips to Chapter");

        deviceState--;

    }

    public void buttonSixPressed() {

```

```
27          System.out.println("DVD Skips to Next Chapter");
28
29          deviceState++;
30
31      }
32
33
34  }
```

DVDREMOTE.JAVA

```
01  // Refined Abstraction
02
03  // If I decide I want to further extend the remote I can
04
05  public class DVDRemote extends RemoteButton{
06
07      private boolean play = true;
08
09      public DVDRemote(EntertainmentDevice newDevice) {
10          super(newDevice);
11      }
12
13      public void buttonNinePressed() {
14
15          play = !play;
16
17          System.out.println("DVD is Playing: " + play);
18
19      }
20
21  }
```

24 Responses to “Bridge Design Pattern Tutorial”

1.  *Ayodeji* says:
   February 4, 2013 at 7:02 am

   Nice Tutorial. Is this pattern main theme to teach how to use abstraction properly or is their more to it than what you have discussed.

   Reply

   -  *Derek Banas* says:
     February 4, 2013 at 12:43 pm

     The Bridge Design Pattern is all about preferring composition over inheritance. It helps in lowering coupling. I cover it further in the refactoring tutorial. It can really be used in so many ways

     Reply

2.  *Atif tinimo* says:
February 11, 2013 at 4:27 am

Hello sir , First off all I would like to thank you so much for these great tutorials . Hats off 🙂

PLease I would like to know if this remote controller that you've created is like the universal remote controller in the real world (I mean the one that can work with any type of equipments) ?.

an other question please .

in the class remoteButton.java , you've mentioned that

" // A reference to a generic device using composition
private EntertainmentDevice theDevice; "

Don't you think that this should rather be called an aggregation instead of composition .

Sorry for being too long .

Thank youu so much , and please keep it up !

Reply

-  *Derek Banas* says:
February 11, 2013 at 4:33 pm

You're very welcome 🙂 Thank you for stopping by my little site. Yes the remote is acting like a universal remote. As per your question about composition versus aggregation. With composition the contained object can't exist without the containing object. Based off of that, yes you are correct that the entertainment device can exist without the remote. Thank you for pointing that out and I will change the code

Reply

3.  *Manesh* says:
March 12, 2013 at 11:02 am

Excellent tutorial.. You saved my day.. Keep it coming 🙂

Reply

-  *Derek Banas* says:
March 13, 2013 at 6:48 am

Thank you 🙂

Reply

4. *brijesh* says:
March 15, 2013 at 3:15 am

1)Simplicity of presentations
2)Simplicity of examples
3)Crisp voice

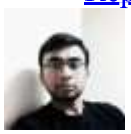
I really appreciate each of your videos. Thank you for all the effort that you have put in.

Reply

   -  *Derek Banas* says:
   March 16, 2013 at 10:00 am

   You are very welcome and thank you for showing your appreciation 🙂

   Reply

5.  *Sachin* says:
April 28, 2013 at 12:50 pm

Nice way to explain Bridge pattern. However, I just have 2 points to make, and would appreciate your thoughts.

1) In example above, I find it difficult to digest that I can create a TVRemote and can pass it a DVDDevice. The code will still work.

So shouldn't the constructor be changed so that a TVRemote can only take in a TVDevice, and similarly for the DVDRemote.

2) Would it perhaps make the example slightly better if we did this instead?

You extend the EntertainmentDevice to create TVDevice. Every TVDevice provides functionality of muting and pausing.

Then you create specific Remotes to choose what the button nine does. So for a tvRemotePause.buttonNinePressed() will actually call the device.pause(), and tvRemoteMute.buttonNinePressed() will call the device.mute().

This said, I really appreciate your effort. Nicely done! :).

Reply

   -  *Derek Banas* says:
   April 28, 2013 at 1:36 pm

   You are understanding the pattern and I like your ideas. Understand that I try to focus in and explain the pattern rather then try to optimize the code completely. Design patterns aren't exactly recipes,

but instead they are guides to follow. It says this much in the GOF book, but I can't find the quote right now. Either way you are getting the idea

Reply

6. *Davies M. Arthur* says:
May 13, 2013 at 2:12 pm

man!! thanks so so much. i am a university student, had a Design patterns course unit this semester, and your tutorials have helped me a great deal.

thanks alot, thumbs up!

Reply

   -  *Derek Banas* says:
   May 16, 2013 at 6:54 am

   You're very welcome 🙂 I'm very happy that you enjoyed them. This was a fun tutorial to make

   Reply

7. *Mahesh* says:
May 30, 2013 at 6:39 am

Great video! My respect for you increases with each video I see. Simply tremendous!
One request though, if you could provide diagrams like those given for Abstract Factory and Factory method Patterns, it would greatly help us to visualise the patterns and grasp the code more quickly.

Reply

   -  *Derek Banas* says:
   May 31, 2013 at 11:26 am

   Thank you very much for the kind message. I'll see if I can get a diagram for the bridge pattern.

   Reply

8. *FTP* says:
November 15, 2013 at 10:56 am

Thank you very much for this tutorial, the concept of brigde is getting clearer.

I have a question in DVDRemote there is an attribute "play", it is not clear to me why would a Remote have "play" instead should it be in the Device?
Assuming "play" would instead be in the EntertainmentDevice.
Then could the DVDRemote have following code

```
public void buttonNinePressed() {
newDevice.toggelPlay();
System.out.println("DVD is Playing: " + newDevice.isPlaying());
}
```

Reply

-  *Derek Banas* says:
  November 16, 2013 at 12:20 pm

  You're very welcome 🙂 I just put play there as a personal preference. There wasn't any other real reason except when I'm sitting on the couch I love to use the remote 😀

  Reply

9.  *FTP* says:
November 26, 2013 at 8:17 pm

Thank you for training video its very helpful and your previous replies too.

In the definition of Bridge..its stated
"Bridge is designed up-front to let the abstraction and the implementation vary independently"
I do not understand what does "abstraction" mean ? Can you help me .

Reply

-  *Derek Banas* says:
  November 29, 2013 at 1:10 pm

  Basically by abstracting we try to keep the user from needing to understand the details. So here we have a remote control. So, if they want to have something different occur when button 9 is pressed for example they just extend RemoteButton and define a new buttonNinePressed method and it will just work.

  Good code can be changed without breaking, or without needing to change anything else. Abstraction helps us do that. i hope that helps.

  Reply

10.  *Firas Ataya* says:
June 2, 2014 at 12:01 pm

Hello Derik,
First of all I want to tell you that I really like your tutorials, they've been very helpful to me, you are brilliant.
I have a little question about the bridge pattern and the remote example, I don't know if I'm missing something, but shouldn't the remote interface has all the methods in the entertainmentDevice class?!!!
I mean what if I want to press the seven and eight buttons using a TVRemoteMute object?

Reply

-  *Derek Banas* says:
  June 8, 2014 at 5:04 pm

  Thank you 🙂 I set this up this way so that not only can we have a multitude of different entertainment devices, but also numerous remotes. Sorry if I didn't explain that better

  Reply

11.  *Marcus* says:
October 5, 2014 at 3:42 am

Hi,

thanks for your great explantion. However, something I have not yet understood – why is it that you implement
– TVRemoteMute
– TVRemotePause
(and probably also a – not shown here – DVDRemoteMute and a DVDRemotePause)
instead of a more general RemoteMute and a RemotePause?
In the latter cause, this would add the flexibility
to – at runtime – combine 2 Devices with 2 Different
Remote-Types, and as such beeing able to dynamically create
4 Different Combinations – like:

AbstractRemote mutingTVRemote = new MutingRemote(tvDevice);
AbstractRemote mutingDVDRemote = new MutingRemote(dvdDevice);
AbstractRemote pausingTVRemote = new PausingRemote(tvDevice);
AbstractRemote pausinfDVDRemote = new PausingRemote(dvdDevice);

I also read the (rather short) explantion in Head-First-Design-Patterns –
and Kathy describes the pattern just like you did – so if I don't (automatically)
trust you (hey, I don't know you! ;-), at least I am trusting Java-God Kathy Sierra, which means your explanation
of the pattern must also be correct – I just don't understand why to limit a pattern, if it could be so much more dynamic, the other way round.
(And I am sure, if I didn't get that part, I didn't get the true entire meaning of the pattern either).

Thanks in advance,

Marcus

Reply

-  *Marcus* says:
  October 5, 2014 at 3:55 am

  p.s. Thinking about it, I guess I have done something like the Strategy-Pattern – However this knowledge still does not answer my question (then I could define the question as- of why I would

use the Bridge Pattern istead of the Strategy-Pattern).

Last but not least – it's said (Internet, Head First book) the bridge pattern would separate Abstraction from it's implementation – well thinking of abstraction and implementation, I would think of something like "AbstractCar and concrete BMW extends AbstractCar" – however (in your bridge-pattern example) – the abstraction is like a real-world-abstraction – a remote control ABSTRACTS the use of the TV, but a remote control is not an abstraction of a TV.
?????? sorry, I am really confused now… ??????

Reply

- *Derek Banas* says:
  October 5, 2014 at 5:29 pm

  Patterns confuse everyone in the beginning. The strategy pattern is used when you have many ways of doing one thing. It also allows you to decide how to do them at runtime. In my tutorial on it I show how we can change the flying abilities of an animal at runtime.

  The bridge pattern allows me to add additional functionality easily. Here I have a remote that can easily be changed to work with not only tvs, but any other type of electronics device.

  I hope that helps 🙂

  Reply

  - *Marcus* says:
    October 11, 2014 at 9:58 am

    well, from what I've found out so far, the bridge pattern and the strategy pattern actually are / seem to be the exact same pattern, the very few comments about it I found on internet forums say it would depend on the context / motivation of what is actually required – if you want to replace functionality at runtime – it's called the strategy pattern, if you want to separate implementation and interface, it's the bridge pattern. However it's still not 100% clear to me why this would justify two distinct names for technically the same thing. Hmmmm.

    Reply

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)

Donate

My Facebook Page

Archives

- March 2022
- February 2022
- January 2022
- June 2021
- May 2021
- April 2021
- March 2021
- February 2021
- January 2021
- December 2020
- November 2020
- October 2020
- September 2020
- August 2020
- July 2020
- June 2020
- May 2020
- April 2020
- March 2020
- February 2020
- January 2020
- December 2019
- November 2019
- October 2019
- August 2019
- July 2019
- June 2019

- May 2019
- April 2019
- March 2019
- February 2019
- January 2019
- December 2018
- October 2018
- September 2018
- August 2018
- July 2018
- June 2018
- May 2018
- April 2018
- March 2018
- February 2018
- 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

About the Author Google+