**Flyweight Patterns**

**Flyweight + Decorator: “Flyweight extrinsic object decoration”**

The decorator is an outer shell of the core objects that wraps and associates some persistent data / methods.

**?Flyweight + Visitor: “actions on diverse extrinsic objects”**

With the visitor pattern you can perform operations on the diverse collections produced by the flyweight pattern.

**?Flyweight + Chain of Responsibility: “Flyweight Object Retrieval System”**

Needs modification apparently. Hint: template method. Interesting…

So you could route to different caches, correct? <- yes…?

Yeah, so there’s this routing system in the flyweight that is part of this whole responsibility chain. What cache will answer the request?

Flyweight algorithm to find appropriate reusable objects in the cache, & create them if not available.

Still missing something. Hint: composite. Interesting…

**?Flyweight + Proxy: “prefetching / recycling wheel”**

x

**State Patterns**

**Previously Covered Patterns:**

**Decorator + State:** **“Stateful Decoration”**

**Decorator + Strategy + State:** **“add missing functionality”**

**??State + Builder: “Builder for diverse input streams”? OR “builder of state objects”...?**

2nd easiest though not easy per se

Hint: memento

Alot of what you’d consider switching based off state is already part of the builder pattern.

Thus the “state object” – if it is adding anything to the builder – must be something different. The stream itself is what is changing state. Yes, this is it.

Official Example: iTranslate Voice. This is an iphone app that translates speech into text. It can take input from 40 different languages.

The State here is the actual type of stream input provided – the language being spoken. Here you’re building a string of text… but you’re switching the director.

There are different levels of implementation you could take this to. You could just simply have a variety of builders that switched based on the state of the input stream.

OR you could loosely couple the builder from the director (the type of stream), allowing translation between 40 different languages. That takes this pattern to a whole new level.

But that would be very difficult. Suffice it to say there is this pattern where you have different kinds of input streams, and you are building based off which kind of stream it is.

The actual implementation of this seems potentially complicated, but interesting. Should you keep it a single builder, or a bunch of different builders? Not sure.

How right: 10, how wrong: 14

Hmm… still not satisfied.

You know, this does bear resemblance to the “builder + state + composition” pattern, doesn’t it?

In that pattern… you’re just building objects that can vary in their state, both functionally (in terms of their position in the hierarchy), but also objects that just simply have variant state.

So that’s another option. You can claim that you’re supporting diverse input streams… or you can claim that you’re creating objects that are State objects, that support differences in state.

Which would it be…? It seems good numbers for the multi-input stream approach are indicated. But this is kind of a strange dilemma, because the other pattern is clearly useful.

**?State + Builder + Composite: “streaming functionally variant composition”? OR “streaming stateful composition”?**

hardest one

Official Example: KLTV First Alert Weather

Hint: builder

So you’ve got this map with varying degrees of weather. The data is streamed in… so you could say this map is being built all the time. It’s being reconstructed as new data is streamed in.

There are different levels of weather severity. And different states – like lightning, or snow, or rainy, or sunny, etc. Severe weather, mild rain, etc.

What I want to know is… where does the composite come into play? Is the composite just the actual structure of the state? So the state can go from mild rain, to moderate rain, to severe rain, etc.?

I think it’s something along these lines, but it seems like I’m missing something… the streaming data. What gets built? You’re building something in some form of a composite structure, correct?

But the composite…. It’s defined by state, isn’t it? The state is the composite. Correct…? This doesn’t seem to ring bells. But I don’t think the builder is constructing the composite…?

Maybe I should generalize this pattern beyond this app. Builders do often produce composites, correct? And those composites might be defined… by their level in the composite.

The objects might take on a different state depending on the level in the composite? So you have this composite of state objects… whose states vary depending on the level in the composite?   
 ~their states vary depending on their function in the composite. What they need to DO in the composite~.

So yes, in a sense… but nodes at different levels have different responsibilities.

So what should we call this…? Functionally variant composition? Constructed Stateful Composition?

“Stateful Composition”? OR “functionally variant composition”? The stateful composition emphasizes the individual nodes ability to change state.

The functionally variant composition emphasizes how nodes have inherent differences in their functionality, perhaps based on differences in their level in the hierarchy.

Both are true… so what to do about it?

And the only difference between this and the other pattern is that this one is constructed on the fly, like a weather map, as far as I can tell.

Perhaps this would lend itself more to the functional pattern, though I don’t know really.

Though really… this one has both. Because the weather map…. Could support lightning, tornados, or not. So you have both elements involved here.

**Bridge Patterns**

**Bridge + Chain of Responsibility: “chained object presenter”**

Official example: edging pro – music remix maker:

So in these music editing applications you have different instruments. They string notes together and form a chain – a musical score.

You can examine this score in many different ways, which facilitates different kind of editing.

So the instrument is the concrete implementation – and that’s what’s chained.

The abstraction is the different ways you can view and manipulate this chain. And any instrument, though its concrete implementation varies, can be paired with any of these abstractions.

So any instrument can be viewed as a continuous series of notes. Any instrument you can open up the editor, which is a whole different view…

So you have this cartesian product pairing of instruments and different views on the instruments. But this is all in the context of this chain – in this case music.

So I think the idea is… the implementations have these request chains. And the abstractions are various ways of viewing, or organizing, or manipulating this request chain data.

So you can pair up an implementation with an abstraction and do an arbitrary number of things with this request chain data.

That’s the closest I’ve been able to come up with for now. I had these other ideas of a tiered request chain (like the speedway rewards cards/coupons example),

and an internal request for an implementation that satisfied an interface, but those just didn’t seem to make as much sense as the one I’ve gone with. But I think this is pretty much the best I’ve got here.

Hint: observer

2nd hint: composite

Word: ossified. Having become rigid in opinion. Looks like I am not done with this yet.

2nd word: check

So infact… it almost seems like this internal check I was mentioning is a part of this whole thing. Doesn’t a request chain check something? It has to… and this compliments the bridge somehow?

3rd word: gainsay – deny or contradict.

So no… not quite that. Something else.

4th word: regret

Looks like I need to go back to the drawing board completely…? Or do I need to rethink what I’ve done…? Go over what I already have again? Modify it maybe?

5th word: consist

What does this chain consist of? Objects, right? What are objects? Concrete implementations.

Is it a varied or uniform chain…?

6th word: belong

Seems to indicate it does belong, it’s a uniform chain. Right? But then again… if it conforms to a particular abstraction, doesn’t it belong to that abstraction? So it is a varied chain in one sense, but uniform in another.

7th word: live

So this is happening at runtime. Does the object belong in the chain or not…? Is that the question?

8th word: disillusioned

So… some objects don’t belong in the chain. Isn’t it possible to construct a variety of chains in this way? A variety of chained objects. But where does this exclusive logic go…?

9th word: present (as in presentation)

So this would be presentation logic, it appears…? And aren’t these various abstract classes different ways of presenting the implementations, in many cases…? So then it goes in the abstraction?

10th word: month

Hmm… So that seems to be it, yes? Lets think of a calendar app. You have a presentation for a particular month. An abstraction of a month…? And how many of the days go into the month?

Only a specific number. So clearly there is this logic in the abstraction that can sort through the implementations.

11th word: route

There are many different ways you could do it. You’re presenting this chain. You have all these implementations. Do they all conform to a larger chain? So you have a giant chain, and you present specific chains?

Hmmm…? Not sure.

12th word: opine

To express opinion. So it is just kind of a matter of how you want to do it at this point, it seems…?

Yeah I think this is done.

**Bridge + Template Method + Prototype: “broken down & reconstructed object”**

most representative left to right order: template method > bridge > prototype >

with this pattern you’re basically breaking down an object into pieces. You’ve got the more invariant functions with templates, and the more variant components with prototypes.

Official example: Cava | Order Online

So you’ve got different food items with different toppings. If you change the food item, it appears the toppings persist, along with other preferences.

The template methods are obviously the different food items. They’re constructed in certain ways, but then they are free to vary in what dish they are.

So to implement this… you just have the template method pattern, right? And this might not be a perfect analogy, but you have this fixed way the food is constructed. Then different dishes…?

The prototypes… are bridged with these different dishes. And you can add toppings in any way you please, right?

But how does this work? So you have this abstract prototype class. It defines… what? Does the prototype just add that compositional element to the object…?

So the prototype inherits from one of these templates, but then… you have these compositional elements as well? So add on this topping or that? I’m not sure…

Seems so, yes…

But why different prototypes? What is the advantage? The prototype is already able to take any sort of item in. Are the two merged? There’s something that subclassing an abstract prototype must add.

What is it…? This is turning out quite interesting, btw – it’s like a somewhat invariant template object, combined with a more variant compositional element added on to it…

But anyway, what is the benefit of an abstract prototype and multiple prototypes?

Would it be like… dairy free? No… that needs to be higher up, even the template pattern needs that specified.

Word 1: regret – so there is something not quite right here…? Perhaps not the dairy free part?

Word 2: clock – hmm… how to interpret this? So a clock is very predictable. A template is very predictable. It’s a recipe. Right? You can change certain parts of the recipe, but it’s a recipe overall. That’s the templates.

Word 3: count – Hmm…. So how many toppings? I’m assuming the prototype is going to be this mix of toppings. Is there a limit to how many toppings? How many extra things you can add on to it…? I don’t know.

Word 4: splendid – hmm… so I think that’s basically it? You can kind of control the prototype with the prototype objects, basically. And that’s all she wrote! So maybe you’re limited to 3 toppings for this pizza. Etc.?

Remember… this entire thing is just a recipe. There’s different ways of making the recipes. There’s different sides and toppings. It’s both combinational, and kind of structured. There are parts that are… required inputs

Those are the template parameters. Then there are like these optional extra toppings…. Right? Hmm… seems a bit strange, contrasting the template parameters with the actual prototype objects.

Word 5: hypnotize. So… you are getting too caught up in all this? You are lost in this talk?

Word 6: behold – you have the answer. It is what was described in words 1 – 4. You’ve got the subclassing approach with the template method… and the compositional approach with the prototypes. Right?

There’s a tad bit left but I think this is mostly done.

**Bridge + Strategy + Proxy: “Access Firewall”?**

This will be a protection proxy, apparently – access related.

So lets say you had these proxy objects… different users access rights. And these mapped to various strategies. Strategies being… algorithms to execute.

So this would allow you to limit how things were mapped within the bridge pattern quite easily.

It also allows you to control who or what can execute code quite easily.

So this is a very interesting pattern. And I think I’ve basically got it accurately described here…

Yeah, so that’s it. That’s the correct idea right there.

**??Bridge + Template Method + Interpreter**

Seems like just a typical language interpretation & execution of a response. But the bridge mapping is interesting… How do you bridge an interpreter with a bunch of template methods?

I think just the specific language constructs can map to template methods freely…? But then you want to calibrate them to specific methods, right? Or are you using weights, or something like this?

It’s not really clear how this works, or if I’m even on the right track at all. Score: 80 Missing: 81 …. On the right track but needs works.

**Facade Patterns**

**Previously Implemented Patterns:**

**Façade + Flyweight + Singleton: “Global Flyweight Interface”**

**?Façade + Decorator: “API enhancer”**

**?Facade + Decorator + Observer: “API system Observer”**

**?Façade + State + Command: “Adaptive functional API”**

**Facade + Builder + Chain of Responsibility**

X

**Facade + Prototype + Command**

X

**Facade + Factory Method + Observer**

X

**Facade + Mediator + Composite**

X

**Facade + Singleton + Adapter**

X

**Mediator Patterns**

**Previously Implemented Patterns:**

**?Mediator + Composite + Façade: “”**

**Mediator + Composite + Iterator**: **“Composite Search Responder”**

most representative left to right order: iterator > composite > mediator >

Official Example: Lego Duplo World

So you’re iterating over this composite, which isn’t difficult. But what are you mediating…? Are you mediating between the elements of the composite…?

It’s legos… it’s almost like you’re rearranging this composite, isn’t it?

Or maybe you’re fitting them together to make them work with each other…? I don’t exactly know. There’s something to it though.

Score: 99. So yes, you’re right onto something here. But I want more details.

Isn’t this quite similar to a controller for view hierarchies, basically? Yes… that’s an example.

So would you just call this a typical controller? No… controller implies a little more.

2nd example, more focused on where you’d use this pattern: LINK – scooter sharing app.

So it’s a scooter GIS app… it’s got scooters spread around the city in various locations. You check them out. In some places… they have multiple scooters. Other places have only one.

You look for a scooter in your surrounding area, rent it, and use it for some time.

3rd example app: PetPage

So it looks like a database where you put in your pet and it pulls up information, then you can examine the information. Literally just like a tableview and detail view.

Word 1: shrill

The composite objects are going to be sending some kind of signal for the others to hear. That’s the only case where a mediator could be used.

Word 2: tendency

Inclination toward a particular behavior. So as you iterate over this thing… which you naturally tend to do with composites. You might find something. And that might send a signal out of some sort.

Word 3: constrain

Two meanings… cause or force someone to follow action. And restrict the scope or meaning of.

So you may say that only certain objects are forced to respond to the signal that was propagated. It wouldn’t be all signals, otherwise you wouldn’t need a mediator, right? So perhaps the levels idea is prudent…?

Word 4: soda

So… sit back and crack a soda, you’ve basically got it. You iterate over this thing… when you find what you’re looking for, you send a signal. This signal is propagated… forcing other objects to respond.

But surely it must be propagated in a limited area, otherwise… you could just propagate it everywhere, but in that case why use a mediator? Why not just make a broadcast…?

Word 5: writer

So keep writing… So my only remaining question is should the levels of the composite define where the messages or sent, or should it be something else like… the type of the node in a mixed composite.

Maybe certain nodes that are listening…? I guess that’s all for the mediator to decide, right?

Where would you even use something like this?

Word 6: craven (means cowardly, or to make cowardly…?)

So you could actually suppress the response. Maybe the better question is… who wouldn’t respond? It depends on what you’re trying to do. Right? It depends on what the composite is.

So what is an example of a composite where you would do this…?

Word 7: finger …

So let’s say you click on something. It’s some object in a composite. So you have this composite structure, it’s … a composite of views. We’ll call it a composite of views. That’s a common structure.

So you click on something within that … and maybe JUST those views contained within; or maybe just those views contained behind it… respond?

So maybe the mediator… is mediating these clicks at these specific in the view hierarchy? That makes sense. Causing some things to respond, but not others… yes, it makes sense.

But where was the iteration in this scenario…? It was kind of accomplished with the finger. But when would this be accomplished at any point otherwise?

Word 8: meal

So what if you’re ordering from a menu. You have different menu items you can click on. You click on one, it takes you to a page listing the ingredients… it’s a composite. Once again, where is the iteration though?

Let’s go back to the scooter sharing app. The composite is the GIS map itself. You iterate over the local scooters… right? When you choose one, you then communicate that this scooter is no longer available. Right?

So do we accept that the composite is the GIS app itself here? It’s not really a pattern that conforms to the structure we had in mind earlier.

But.. you will have to loop through some database and find all the local objects, won’t you?

It’s a strange pattern.

Word 9: synonymous:

So I guess what I’m looking for is just exactly what I just described, and kind of the same thing I’ve been saying all along. Right? So how does that map example relate to our standard architectural idea?

You iterate through a database. You are looking for a particular set of scooters – defined by a certain node or level within the composite. You rent the scooter… this needs to be communicated to the other scooters.

So yeah, though it’s a bit odd, it is an example of this… this strange pattern. What should we call it?

Word 10: cut

Something to do with removing an item from the composite? And communicating that? So we could call it… add/remove from composite? That’s too simple, that’s just a simple algorithm.

Word 11: peaceful

Hmm… so you don’t necessarily have to be radically altering the composite. You might just be finding something and sending a message, too…

Word 12: contrast

Interesting… So in one case you’re altering the composite, in another case… you’re sending a message. So then you aren’t really altering the composite, then?

Well no… you still need to a send a message the thing has been altered, right? Do you…? Wouldn’t the algorithm handle that?

If you’re doing nothing to the composite, what is there to really communicate about?

Word 13: gun

So you are sending a signal that some noteworthy event happened. It’s not a trivial event. It’s like a gunshot: “this happened”. The other things need to know this…

What if nodes higher in the composite have counts of how many nodes are lower in the composite? You’d need to tell them that one was taken out, right?

So that’s an algorithm yes, but it can be contained in a controller, can’t it? A mediator class.

That way you encapsulate everything…? And you have all these nice functions for communicating between nodes in this structure?

Word 14: magic

Hmm… not sure how to interpret this. Did I interpret the previous correctly? I’m not sure. Maybe I should go back and look over that one…

So I think it can be either a change to the composite structure, or just a passive event, but it needs to be significant enough to tell the other nodes. That’s really the critical distinction.

Word 15: fumbling

So this has been a very jumbled explanation…

But even to communicate to other nodes, you’re kind of fumbling around in the dark, aren’t you? You have to have some structure that can communicate properly with the appropriate nodes.

So that’s what the mediator is. It’s that structure that communicates some sort of thing happened, which needs to be communicated to other nodes in this composite for whatever reason.

Word for naming the pattern: adjoining.   
 hmm… so… “composite search responder”

**Mediator + Prototype: “Dynamic Session”**

Word 1: convict

Are you a part of this mediator? To the prototype…. Yes or no? Is that the question? I’m not sure. Alternatively… do we need to create our mediator using the prototype pattern?

In doing so we could pass in those links that might need to communicate with one another…? We could maybe even configure them….? Or a 3rd option maybe?

It seems the first option is closest to home.

Word 2: deca y

Interesting. Hmmm… So you can add objects to this mediator, and they can participate in this communication link. But they can get cleaned up as well, right? They can be deallocated.

So isn’t this a kind of … almost like a chatroom… an open communication network?

Word 3: incise. To cut into

Hmm… so you can also control access with this mediator pattern, can’t you? So how does this work, exactly?

You have this prototype creational class… it’s associated with the mediator. You pass it some piece of data… it can either add a newly created object to the network…

It can disallow it from entering the network… it could, based on its configuration, give it certain access privileges…?

Word 4: hesitant

So … should you think a little more about this? Hmm…. Or does this merely refer to the patterns conditional entry bit? Is this even pattern of the pattern?

Think a little more about this, it seems.

So how would this be implemented? So the mediator itself would need some function that accepts a prototype and instantiates a new node in the communication network.

Yeah, I think that’s basically it.

Assessment word: rapid

So I am being a bit hasty in my conclusion about this pattern. There’s more to it… but what?

Word 5: oatmeal

Slow… slow? Hmm….

What about a login process? So the prototype… it can be like this drawn out login process. Right? What about a login system where you pass your credentials (that’s the prototype) and then …

A node in the system is created for those credentials – your user profile is pulled up. Is that really a prototype? Well… you pass this bit of info in and you get this larger object. It must be.

Word 6: quarrelsome

Hmm…. Not correct, then? Or disputable? Maybe that’s not quite the same thing, really.

My whole description up there is a run time system. What about a compile time system? What would that look like?

Word 7: science

You’d need to know beforehand what the network nodes would look like… how they’d be connected and so fourth. But you could do it.

Would you use a combination of compiled and dynamic models? You’d just use a dynamic model.

Word 8: lopsided

So what about configuring the network? How would you actually… limit what nodes a new node could communicate with?

Or is this a flaw in my design? If so… what? You’d limit the nodes by… well, I’m not sure. Prototypes… they’d need some sort of information on them – profile information, for example – that would be used.

Word 9: ball

Would you even want to limit access? Is that the sole point of the prototype…? No. What other point does the prototype serve, then? It’s giving that thing access to communicate across this network. Right ?

So what are some more details about that?

Well… it seems it’d have to be the entry point into the network, so the object itself will have to be returned. And I’m kind of imagining the implementation currently being across the internet.

It could work as either a client/server or P2P architecture. But in any case, you have this object wired up in the mediator… Well wait, don’t you just use the mediators interface for communication?

The mediator must provide that interface, correct? Well… the prototype could provide it.

I’m not sure what other details are needed, aside from domain specific ones.

Go back to the chatroom example. There you actually had these objects added to the mediator itself. Yes you had an interface to the mediator… but these objects were actually created in the mediator.

I think that’s what we’re looking for, right…? Any other words?

Word 10: comfort – looks like we’re done!

So we got lost for a bit, but wandered back to word 2 (when we were on word 9) and found our way back. Now… what to call this?

Name hint 1: state

Hmm…. So this is a network that allows nodes to be added in / out dynamically, right? So this is…. A dynamic network?

Name hint 2: festive

Dynamic collaboration session? Dynamic session? I think that’s it, dynamic session.

**?Mediator + Factory Method**

So… is this a factory that produces mediators, or a mediator that works with factories, similar to the above?

It’s a factory that produces mediators, apparently! Interesting…

I think IOS has something similar to this, doesn’t it? It must, it has so many different types of controllers.

Still, this is a weird concept.

Design methods: 1) videos on patterns 2) python script generate pattern 3) random app from app store 4) walk through with words 5) brainstorm by self 6) construct a stack of the objects

Go over this again and use methods to double check it / rename some things:

**Decorator Patterns**

**Previously Covered Patterns:**

**Flyweight + Decorator “Flyweight extrinsic object decoration”**

**Decorator + Command: “Apply decoration”**

When issued, the command applies a decorator.

Example: text highlighting. You select some text and press the highlight button. The letters are decorated with an object that will cause them to be highlighted when drawn.

**?Decorator + State: “Stateful Decoration”**

Decorators can be applied / unapplied based on changes in state.

Though state objects traditionally change functionality via switching subclassed implementations, encapsulating functionality in decorators is an alternative, combinational approach.

Typically some mix of subclassed switching & decoration is reasonable. Pure reliance on decoration is an option also.

How correct is this? Like 85 apparently… but not entirely, it seems.

If it’s a stateful decorator… than the decorator itself alters its functionality based on state. Correct? On the other hand… if you are applying / unapplying decorators based on state change, that’s something else.

It appears the former is infact what is more intended by this pattern… it seems.

Which makes sense to me…? It seems quite obvious that you might apply / unapply decorators based on state, there’s no other way to do that. That is some approximation of a state object, though.

On the other hand, a decorator that actually changes based on state – the reverse approach…. You know, FUNCTIONALLY these two approaches accomplish the same thing, don’t they?

Not quite… with the first approach you have the combinational ability of decorators. With the second approach… you’re kind of attaching this conditional functionality. So it’s not combinational.

Though… it CAN be designed to be combination, if you want – lets say by using a bitmask. So infact… either way sort of amounts to the same thing, doesn’t it? So then what is preferred?

This may just be an implementation detail. So I will call this “stateful decoration” to account for both possible implementations.

**Decorator + Strategy + State: “add missing functionality”**

stack order: strategy > decorator > state >

State lacking functionality – such that it can’t support the use of an algorithm - can be augmented with decorators.

The state indicates what functionality is supported by an object. The decorator adds missing functionality, and the strategy then uses the full extended functionality.

Official example: some IOS games support control by remote playstation controllers, and some also support dual shock. That’s the global state in this case – what kind of functionality is supported.

You could add that functionality to a game by wrapping it in a decorator. You could then interface with the game using your dual-shock controller – which is the algorithm or strategy in this case.

**Decorator + Composite + Adapter:**  “**Diversely Decorated Collection Adapter**”

stack order: adapter > decorator > composite >

So on this conglomerate you could have any set of decorators. You need an adapter than can interface with a variety of decorators, essentially.

This is kind of like swapping different decorators and interfacing; in contrast with the below, which is multiple layers of decorators and interfacing

**Decorator + Adapter + Decorator: “Multi-Layered Decorators Adapter”**

stack outer to inner: adapter > decorator > decorator >  
So it actually looks like this refers to how you often stack decorators. There arises the need to interface with these various decorators… correct? After all… decorators may add functionality. So you must interface with them differently. So don’t you need to adapt your interface to that new stack of decorators? Yes!!!

**?Decorator + Façade: “API enhancer”**

complex system > decorator > facade >  
 The decorator pattern also can augment the facade pattern. A facade is designed to simply interface with the complex system it encapsulates, but it does not add functionality to the system.

However, the wrapping of a complex system provides a space that may be used to introduce new functionality based on the coordination of subcomponents in the system.

For example, a facade pattern may unify many different languages dictionaries under one multi-language dictionary interface.

The new interface may also provide new functions for translating words between languages.

This is a hybrid pattern - the unified interface provides a space for augmentation.

Think of decorators as not being limited to wrapping individual objects, but capable of wrapping clusters of objects in this hybrid approach as well.

**?Decorator + Observer + Façade: “API system Observer”**

complex object system > decorator > facade > observer

You are observing the API, and the decorator provides functionality that is useful for observing it.

More detail needed…