

Forum:[Hack Forums](#)

Thread: [Design Patterns](#)

?1???po?N? - January 7, 2017, 2:20 pm

Lately got classes about different Design Patterns within Java. (Or other OO languages acutally)

Since I'm quite new to the concept of Design Patterns I bought the gang of four design patterns book and started reading through the purpose of the different design patterns.

I mean it totally depends on the scenario but I was wondering what design patterns do you guys use the most and why?

clrlslwi - January 7, 2017, 3:50 pm

Design patterns are just a way to communicate with other people. It's useful to know about them, but at the end of the day, they're just common ways to achieve very specific goals.

To answer your question, I don't know. If you're constantly thinking about design patterns while you're programming, you might want to reevaluate your decisions.

Stealth™ - January 7, 2017, 7:21 pm

(Jan 7, 2017 04:50 PM)Demmonic Wrote: [\[CITE\]](#) Design patterns are just a way to communicate with other people. It's useful to know about them, but at the end of the day, they're just common ways to achieve very specific goals.

To answer your question, I don't know. If you're constantly thinking about design patterns while you're programming, you might want to reevaluate your decisions.

I've never read something more untrue in my life. Please do not pursue a career in Computer Science.

clrlslwi - January 7, 2017, 7:25 pm

(Jan 7, 2017 08:21 PM)Stealth™ Wrote: [\[CITE\]](#) I've never read something more untrue in my life. Please do not pursue a career in Computer Science.

Wow. I love the part where you refute what I was saying! Thanks for the talk, good stuff.

You should think about design in general while programming, and you should not be focusing heavily on implementing strictly defined design patterns.

edit: wait wait wait, you're in college? Holy shit, you're a joke! I am technically a professional, thank you very much, and I guarantee I have more experience than you. Please go back to your HW services, and rsps development.

3cli??pse - January 7, 2017, 7:49 pm

(Jan 7, 2017 08:21 PM)Stealth™ Wrote: [\[CITE\]](#) I've never read something more untrue in my life. Please do not pursue a career in Computer Science.

Imao what. Coming from some kid who's selling homework and is proud of being an Oracle Certified Programmer. I bet you've never even had an actual job requiring you to design software.

Dostoevsky - January 8, 2017, 12:45 am

Without going into much esoteric detail, if you're doing Java professionally (this actually also applies to pretty much all OOP languages), you should know:

- SOLID principles
- a bit about GRASP
- common idioms that are language agnostic e.g. DRY - mostly so you can be pretentious on HF.
- design patterns

In C++, I often find myself using the "factory" pattern a lot to control the instantiation of what I call "pseudo-singletons" - probably a really bad naming strategy.

I go by my own rule that is: an object should never manage its own creation, lifetime, and destruction. It's all part of a bigger system. E.g. defer the strategic responsibility of handling these things to other, single purpose, classes with often vague names.

I often find myself using the visitor pattern (I do a lot of file parsing stuff where it's really useful - e.g. I ripped off OW2's ASM library design to make my own C++ version) - obviously how I store the other shit like the constant pool etc. differs greatly.

But yeah, to list the common ones I use that I know the name for: factory, singleton, bridge, adapter, visitor, and mediator. They're probably the most common ones I use. You'll find yourself reading up on design patterns and being like "oh, I kinda did this instinctively a while back" and seeing where they get their ideological roots.

As a general rule, you should employ dependency injection into most things you do (for the most part, anyway).

sarcosuchus1 - January 8, 2017, 7:05 am

facade pattern is mostly used based on my experience; also factory pattern is important

?I???po?N? - January 10, 2017, 11:02 am

Nice to see all those diverged opinions about design patterns, didn't want this thread to cause any chaos.

Anyways thanks for all replies! I personally lately used the factory pattern in combination with a singleton and observer pattern for a school project.

(Jan 8, 2017 01:45 AM)Dostoevsky Wrote: [\[CITE\]](#) Without going into much esoteric detail, if you're doing Java professionally (this actually also applies to pretty much all OOP languages), you should know:

- SOLID principles
- a bit about GRASP
- common idioms that are language agnostic e.g. DRY - mostly so you can be pretentious on HF.
- design patterns

In C++, I often find myself using the "factory" pattern a lot to control the instantiation of what I call "pseudo-singletons" - probably a really bad naming strategy.

I go by my own rule that is: an object should never manage its own creation, lifetime, and destruction. It's all part of a bigger system. E.g. defer the strategic responsibility of handling these things to other, single purpose, classes with often vague names.

I often find myself using the visitor pattern (I do a lot of file parsing stuff where it's really useful - e.g. I ripped off OW2's ASM library design to make my own C++ version) - obviously how I store the other shit like the constant pool etc. differs greatly.

But yeah, to list the common ones I use that I know the name for: factory, singleton, bridge, adapter, visitor, and mediator. They're probably the most common ones I use. You'll find yourself reading up on design patterns and being like "oh, I kinda did this instinctively a while back" and seeing where they get their ideological roots.

As a general rule, you should employ dependency injection into most things you do (for the most part, anyway).

Especially thanks for this extensive reply Dostoevsky, a lot of useful information in there.

(Jan 7, 2017 04:50 PM)Demmonic Wrote: [\[CITE\]](#) Design patterns are just a way to communicate with other people. It's useful to know about them, but at the end of the day, they're just common ways to achieve very specific goals.

To answer your question, I don't know. If you're constantly thinking about design patterns while you're programming, you might want to reevaluate your decisions.

Correct me if I'm wrong here, I'm also still just a student.

But isn't the idea you are thinking about design patterns before you start programming and not while you are programming?

I mean for most of my school projects I first have to make a UML design that meets all requirements before I can even start programming.

When I started this study I had the feeling that I was about to program a lot, but actually next to programming I am consuming 50% of my time designing.

clrlslwi - January 10, 2017, 12:02 pm

(Jan 10, 2017 12:02 PM)?I???po?N? Wrote: [\[CITE\]](#) Nice to see all those diverged opinions about design patterns, didn't want this thread to cause any chaos.

Anyways thanks for all replies! I personally lately used the factory pattern in combination with a singleton and observer pattern for a school project.

Especially thanks for this extensive reply Dostoevsky, a lot of useful information in there.

Correct me if I'm wrong here, I'm also still just a student.

But isn't the idea you are thinking about design patterns before you start programming and not while you are programming?

I mean for most of my school projects I first have to make a UML design that meets all requirements before I can even start programming.

When I started this study I had the feeling that I was about to program a lot, but actually next to programming I am consuming 50% of my time designing.

The whole idea of design patterns are a way to communicate with other people. Chances are (as mentioned above), you've used plenty of design patterns without even knowing what they are. The reason I felt the need to make this clear is that a lot of people learn about design patterns, and attempt to chain a million of them together for no reason. Yes, you should know about them, and you will think about them while programming, but you should not focus on implementing a million of them. You should use them when it makes sense.

Yes, planning is a somewhat important part to having a clean project from the start. You're never going to be able to plan far enough ahead in the real world though, chances are your application will always be changing.

(Jan 8, 2017 01:45 AM)Dostoevsky Wrote: [\[CITE\]](#) As a general rule, you should employ dependency injection into most things you do (for the most part, anyway).

Just so we're clear, anyone writing OO code is already utilizing dependency injection. People should look into DI frameworks though.

Sorry, I was going to reply more, but apparently google decided to interview me half way into this post..

Spoiler ([Click to View](#))

Haha xd

randomized patterns - January 10, 2017, 12:30 pm

(Jan 10, 2017 01:02 PM)Demmonic Wrote: ([CITE](#)) Spoiler ([Click to View](#))

Haha xd

What extension is that?

clrlslwi - January 10, 2017, 12:41 pm

(Jan 10, 2017 01:30 PM)lolxd123 Wrote: ([CITE](#))

What extension is that?

Dark skin for youtube, although I think Youtube Plus has one too.

Fizzix - January 10, 2017, 6:36 pm

Not even going to argue with the retards in this thread. You should be trying to fit pretty much everything into a well established pattern. It makes it so much easier for other to understand what the hell you're doing, and it's easier to catch/not create bugs in the first place.

You can think of patterns as building blocks for your application. Use them as much as you can, and think of not using one as a lot of overhead cost (so they should be avoided).

GenericSkid - January 11, 2017, 6:03 pm

(Jan 7, 2017 04:50 PM)Demmonic Wrote: ([CITE](#)) If you're constantly thinking about design patterns while you're programming, you might want to reevaluate your decisions.

Is it a problem to think about how to best approach a logical problem?

Dostoevsky - January 11, 2017, 9:00 pm

(Jan 10, 2017 01:02 PM)Demmonic Wrote: ([CITE](#)) Just so we're clear, anyone writing OO code is already utilizing dependency injection. People should look into DI frameworks though.

Dependency injection isn't an inherent trait of OOP.

You can write really shitty OO code and really good OO code. Good OO code would be utilising DI but it isn't there by default (as your rather patronising response seems to imply is the default - no). So, I assume you have too much faith in the skiddos of HF.

DI goes above and beyond passing around classes.

So, "just so we're clear", you may have a little inherent DI usage but that doesn't mean your project's overall design couldn't do with better DI.

ASLR - January 13, 2017, 8:41 pm

(Jan 8, 2017 01:45 AM)Dostoevsky Wrote: [\[CITE\]](#) Without going into much esoteric detail, if you're doing Java professionally (this actually also applies to pretty much all OOP languages), you should know:

- SOLID principles
- a bit about GRASP
- common idioms that are language agnostic e.g. DRY - mostly so you can be pretentious on HF.
- design patterns

In C++, I often find myself using the "factory" pattern a lot to control the instantiation of what I call "pseudo-singletons" - probably a really bad naming strategy.

I go by my own rule that is: an object should never manage its own creation, lifetime, and destruction. It's all part of a bigger system. E.g. defer the strategic responsibility of handling these things to other, single purpose, classes with often vague names.

I often find myself using the visitor pattern (I do a lot of file parsing stuff where it's really useful - e.g. I ripped off OW2's ASM library design to make my own C++ version) - obviously how I store the other shit like the constant pool etc. differs greatly.

But yeah, to list the common ones I use that I know the name for: factory, singleton, bridge, adapter, visitor, and mediator. They're probably the most common ones I use. You'll find yourself reading up on design patterns and being like "oh, I kinda did this instinctively a while back" and seeing where they get their ideological roots.

As a general rule, you should employ dependency injection into most things you do (for the most part, anyway).

I'm with him. Factory is the most common pattern I use and dependency injection is a life saver when it comes to writing unit tests