# What are the four pillars of Object-Oriented Programming?

I had a feeling this would come up later on in the course, so I actually wrote down the four pillars when they were referenced earlier in the week: Abstraction, encapsulation, inheritance, and polymorphism.

Abstraction and encapsulation kind of go hand in hand in my head. It's basically a way of blocking the user from editing data we don't want them to touch. As I recall, the video described it sort of like not needing to know how an engine runs in order to drive a car. In my mind it might be for security reasons, like you're handling sensitive data and you need some layer of protection (abstraction) to prevent someone on the outside from modifying things they aren't supposed to have access to.

Doing some reading on the matter, this page gives a good example of abstraction that I think reaches beyond what the lesson had to say about it. It's targeted more at Javascript than Java, but it's still about OOP overall, so I think it has important things to say:

https://www.freecodecamp.org/news/four-pillars-of-object-oriented-programming/

It focuses specifically on API usage and how there's a lot of functionality you'd have to duplicate in your code in order to satisfy the needs of logging into and accessing an API. Or, instead, you write an abstraction that quietly handles all of that behind the scenes and all you have to do is pass it some basic credentials to keep everything simple and easy to understand. It also cuts down on debugging time, too.

Encapsulation is closer to what I was talking about earlier. Basically, only exposing the absolute bare minimum of functionality to the user so they can't modify things that could cause an exception (or worse).

Inheritance is a key element that keeps everything readable. In order to support concepts like abstraction and encapsulation, which require lots of surrounding concepts like Getters and Setters in order to maintain modularity and privacy, it can require big chunks of (simple, easily repeated, even auto-generated) code. Inheritance allows us to break everything out into different class files that all talk to each other, both for organizational reasons and, again, limiting repeat code. We inherit data and functionality we've already written somewhere else.

Polymorphism ties it all together. Sort of like inheritance, polymorphism is about accessing the same method across a range of different classes, cutting down on reused code. Multiple sites I found just straight up included the definition of polymorphism, which is “the condition of occurring in several different forms.” Or, in other words, you write a method in one file and reference it in another, making everything easier to work with, easier to organize, and easier to debug.

## What is an exception?

An exception happens when something in the code is out of range of what the computer expects. Code typically allocates specific blocks of memory, and when you try to access something that is not within the allocated memory, it causes an exception which crashes the software or otherwise causes it to stop executing prematurely. It also happens if you try to do something that is mathematically impossible, like divide by zero.

## What are the differences between checked and unchecked exceptions?

A checked exception is something the compiler knows has the potential to crash the application (or "throw an exception"), and thus will not let the code be run. In order to work around this, checked exceptions must be wrapped in try/catch blocks. Basically:

```
try {
        suspect code;
} catch {
        the exception result;
}
```

"Try" is what it sounds like -- the program will try to run the code, which in some cases will probably execute correctly. But if that potential exception comes up, the "catch" portion will allow additional code to be run (like displaying an error message) without ending the program's execution.

An unchecked exception is basically the opposite of that. It's when the compiler cannot automatically detect a situation in which an exception could occur. Or in the easiest possible terms, it is an exception that cannot be checked for.

Source: https://www.geeksforgeeks.org/exceptions-in-java/

## What is unit testing and why is it important?

Unit testing is basically isolating small pieces of code and looking for what parts are causing problems. Instead of running an entire program where there could be dozens, hundreds, or even thousands of failure points, you look at specific chunks to verify they are working properly one by one. This is generally a faster, cleaner, more targeted way to diagnose and fix deeply rooted problems.

Source: I've personally done unit testing in other scripting languages. Literally all I did was google "what is unit testing?" and went, "oh yeah, of course, I've done that."