# **2.7 Static Properties & Methods In Object Oriented PHP**

we covered static variables in the first section of the course and i will leave the link to that lesson in the description if you want to refresh your mind on static variables.

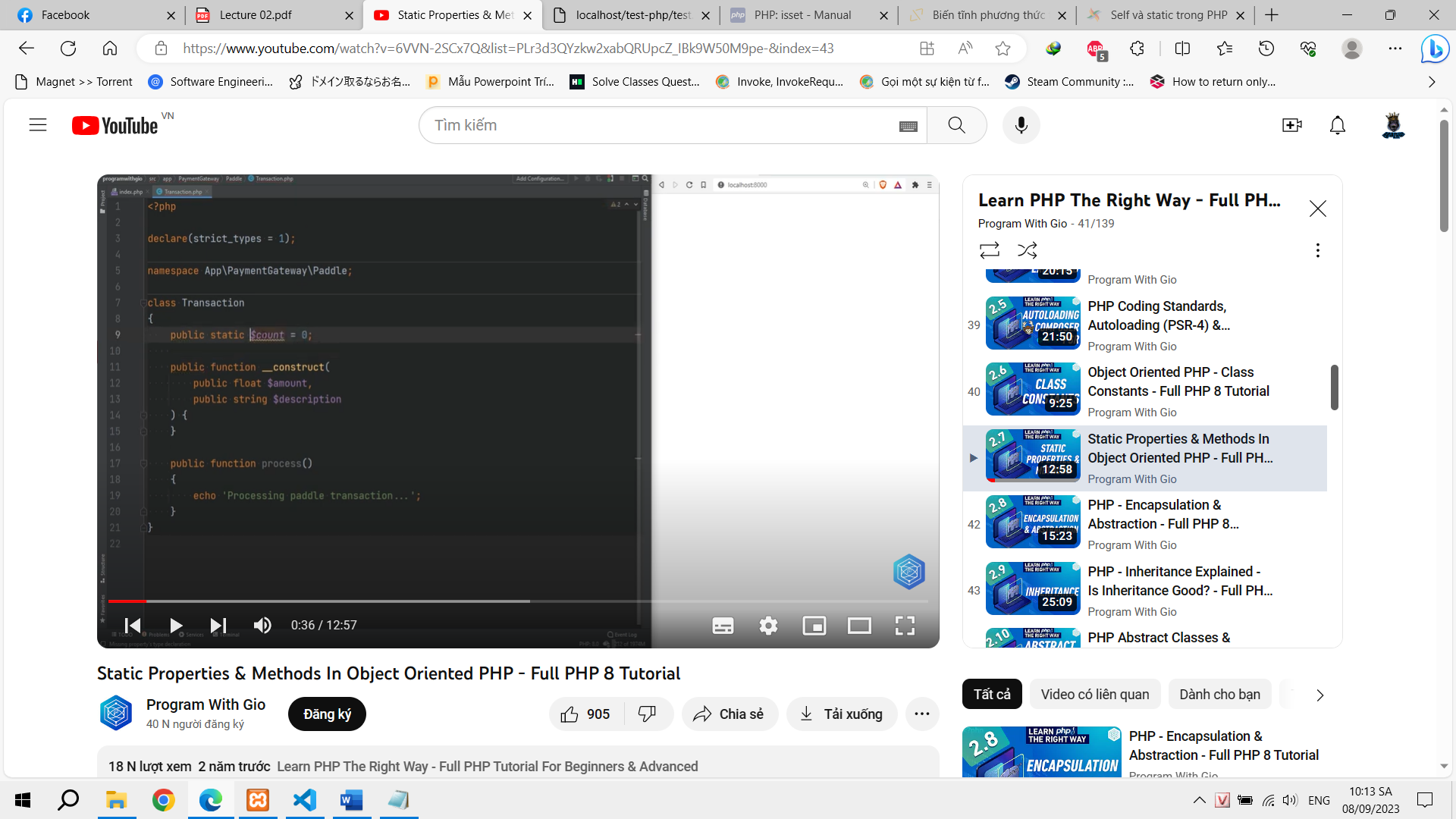
in this lesson we're going to be talking about static properties and methods of the class.

so let's add a static property and a method to this transaction class

you can define your properties and methods to be static using the static keyword

(Có thể định nghĩa các thuộc tính và phương thức tĩnh bằng bằng việc sử dụng từ khoá “static” )

so let's add a count property to this transaction class



and let's make it static so we could do

public static count and we can set the default value to zero and let's make this an integer now you could put the static keyword before the access modifier or public in this case and it would still work but if you're

following standards you should put it after the access modifier but it is upto you as long as you stay consistent

( public static int $count = 0;)

you can access static properties and methods the same way you could access constants using the scope resolution operator

so if we go to index.php, we could do var dump transaction object scope resolution operator and count

($transaction = new Transaction(1, “Test”);)

($transaction::$count;)

if we refresh the page we get zero the thing is you don't actually need the object to access the static methods and properties of the class because they're not created and they're not associated per object they are created and are associated on class basis so they belong to the class itself and not to any

particular object or an instance and here as you can see this is perfectly valid and it will work you can access static properties and methods on the objects but you don't have to you could access them on the class itself the same way you were accessing class constants

(Thuộc tính tĩnh hoặc phương thức tĩnh có thể truy cập trực tiếp thông qua tên lớp và toán tử phân giải phạm vi truy cập. VD: Transaction::$count;)

A screenshot of a computer

Description automatically generated

so if we do a transaction count and we refresh the page we get zero now i mentioned that

they are not associated per object what that means is that the regular class properties like amount and

description in this case every time you create an instance these properties get created and get associated to that specific object so you could have multiple different transaction objects and each one can have different values for those properties

you can think of static properties as global variables in a way their values are shared across the objects because they're not tied to a particular object or an instance they're tied to the class itself so let me show you an example of that let's increment count every time a new instance is created when referencing

( các thuộc tính tĩnh có thể được hiểu như các biến toàn cục vì chúng không bị rằng buộc với một đối tượng cụ thể hoặc ‘instance’ chúng được gắn cụ thể ở lớp đó)

static properties or methods from within the class you could use either the self keyword or you could use the class name as well so we could the self count and we could simply do plus plus now if we refresh the page as you notice it changed from zero to one now if we duplicate this say five times and we refresh the page as you can see now it's printing five

A screenshot of a computer

Description automatically generated

if we comment all of them out and we don't have any object

now it's going to be zero which is the default value so as you can see the value is shared and it's kind of

globally accessible right you don't need an instance of the class you could simply just access them using the class itself

now let's change the access modifier to private here and now this property is only accessible within the class and it's no longer accessible outside of that class

(nếu đổi phạm vị truy cập là Private thì thuộc tính chỉ truy cập trong class và không thể truy cập từ bên ngoài)

public static int $count = 0; 🡪 private static int $count = 0;

so if you refresh the page we're going to get a fatal error to fix this we need to add a static method so we could do something like get count and we could create this method with the static keyword and simply return self count

(Thêm một phương thức tĩnh trả về giá trị của $count để có thể truy cập tới $count từ bên ngoài phạm vị của lớp)

public static function getCount() : int {

return self::$count;

}

if we refresh the page everything is still working if i go ahead and uncomment these it will still print five because the access of static methods does not require an instance of the class and because they belong to the class itself and not to a particular object the variable

this is not available within the static method so you cannot do something like echo this amount you're going to get a fatal error which makes sense right

A screenshot of a computer

Description automatically generated

(Không thể dùng $this để truy cập hoặc thay đổi giá trị của thuộc tính trong phương thức static, thay vào đó phải dùng từ khoá ‘self’)

because this variable refers to the calling object but in the case of statics we don't really have an object so instead of using the this variable you would have to use the self keyword and there are two other keywords that you could use like parent and static and we'll talk about those later in the course once we get to the inheritance and late static binding so don't worry about those two keywords right now I don't want to overwhelm you with too much information another thing you should be aware of is that if you try to use the scope resolution operator on a non-static property you will also get an error so for example we have the amount and description properties here that are public

if we go to index.php and we did something like transaction scope resolution operator amount this is not going to work let me delete the other four transactions and

if i refresh page we're going to get fatal error because the static property amount does not exist we only have the non-static property amount and you cannot access non-static properties statically and same applies to methods

if you try to access a non-static method statically you would also get an error since php8 and prior to phva you would get a deprecated message so if ichange this to process which is a non-static method this one

and i refresh the page we get the fatal error because i am on php8 but prior to php8 you would get

the deprecated message all right now that we've covered the basics of the statics let's actually talk about the use cases for static properties and methods the use of static methods and properties is generally considered a bad practice

but there are some use cases where the use of statics can come in handy one use case for static properties is to have some sort of counter as i did in this example or to cache values so it can often be

used to implement a technique called memoization which can speed up expensive operations by caching the results for later access and i covered an example of that in the static variables lesson for which the link is in the description and you could check it out

if you want to you might also come across a code that makes a use of static properties to implement a single tone pattern

we'll talk about design patterns later in the course but singleton pattern maintains a single instance of a given class throughout your app execution. i'll show you the most simplest example of the singleton pattern implementation using the static properties and methods

so i have a db class created here and this is the most basic example of the singleton pattern now

this is not exactly fully singleton because it does not contain things like clone and other magic methods and the main reason for that is because we haven't covered those magic methods and i decided not to include them here

(Pattern Singleton)

but this is the most basic one and i'll explain what this is so we have a static property instance that is nullable but its type is the current db class and it's set to null by default then in the constructor we are accepting configuration or maybe db credentials or something like that and that's the property right that is

non-static property and notice that the constructor access modifier is set to private this means that we cannot directly instantiate this class

(Hàm khởi tạo đang là private thì ta sẽ không thể trực tiếp khởi tạo class đó ở bên ngoài )

(VD: $db = new DB(); // get error)

if i go to index.php and i tried to get the instance of the db by simply instantiating the db class this is not going to work if I refresh the page we are going to get a fatal error and because of that we havethis get instance method which is a static method and we can call this to get the instance of the db class this again accepts the same arguments as the constructor and checks

if the instance is set to null if it is set to null it will set that property to the new instance of the db class and then simply return that instance so in here for example if we did db get instance and we passed in some

config here no matter how many times we call that get instance it will still get the same instance and i can show that to you very easily if i go to db.php i could simply echo instance created with a break line and

if i refresh the page we see that it's only printed one time

if however we change this to public and instead of db instance we did the new db to create the new instance of the db class this way if i refresh the page we see that it's being printed five times but in the other case where we have the get instance and when we have this set to private we're only getting a single instance i'm only showing this to you

A screenshot of a computer

Description automatically generated

because you will come across some code that might implement singleton pattern this way

and it does not mean it's the right way dependency injection is the right way and we'll talk about it

in more detail and about singleton pattern later in this course so don't worry about it right now another use case for static methods would be to create some kind of utility methods that don't really need instance of the class so maybe you had some kind of formatter class that had bunch of methods to format your currency or format the amount or format the percent value and so on and it doesn't really depend on the

object

(class DB

{

    public static ?DB $instance = null;

    private function \_\_construct(public array $config)

    {

    }

    public static function getInstance(array $config) : DB

    {

        if(self::$instance === null)

        {

            self::$instance = new DB($config);

        }

        return self::$instance;

    }

}

(Lazy initialization)

so if those methods are not tied to the object and they don't need the instance of the class and it makes it convenient for the developer to access them then you could use the static methods in that way so you could have something like format amount and pass your amount and that would work in that case you don't need the instance of format because this method is not tied directly to that object

another use case that you might see with static methods is with the factory pattern and we'll talk about the factory design pattern later as well but basically factory class is responsible for creating objects of other classes and often developers implement it using the static methods so for example in here we are instantiating this transaction class but

if we wanted the factory to handle the instantiation and creation of this object we could do something like

transaction equals to transaction factory make and passing the necessary arguments here and this static make method would be responsible for creating the transaction object don't worry about this for now

we'll cover the factory patternin this course and this is not the rightway either the better way is to use dependency injection again but i'll show you the proper way to implement the factory pattern without using the static methods the main reason why the use of static properties and methods is discouraged

is that they represent what's called a global state which means that you can modify the data or call the static function from anywhere in the code which can make things harder to maintain and harder to test there are some very small niche use cases where the use of status could make sense but a lot of

times statics are misused and dependency injection should be used instead also some might argue that the use of statics is faster than creating instances of objects which can be true

but it's a micro-optimization that in most cases would not matter before we wrap up this video i want to

mention and show you a couple of things one of them is that you could access static methods non-statically so for example if we did var dump transaction scope resolution operator get count this would work right this prints one because getcount is a static method

you could also use the object operator to access that static method and it would still work though i would not recommend doing this

because it's not clear that this method is static and plus you cannot access this variable within this method so it could cause confusion and problems down the road

another thing i wanted to mention is that you could have static callbacks or static closures for example

if we had something like array map and we passed the callback here right function and we passed some kind of array and you do whatever you need to do in this array map function you could set this function to be static and the main reason why you would see somebody do this is because when you set the function to static you can no longer access this variable so if we move this inside the class like transaction class for example

if we did this somewhere maybe within the process method when you define the callback or the closure as static you can no longer access this variable so therefore this amount would not work so if we did something like bar dump this amount and let's make this array of size one and let's call transaction process here and we refresh the page we're going to get the fatal error because that this variable is not available if we remove the static keyword from here we refresh then everything is working we get the

text printed and we also get that amount of are dumped from the arraymap function so that's one of the reasons why somebody would have a static callback or a closure to avoid having access to this variable because from within the callback you can technically change this value right I could do something like

this amount equals to 35 and now if you go to index.php and rebar dump the transaction amount we passed 25 but when we refresh now it's set to 35 and this can cause some unexpected bugs and hard to debug issues and if you were to add this static keyword here now you can be sure that this kind of code will throw an error and not let the developer proceed

so if i refresh we'll get that fatal error also another reason why somebody would have static closures or callbacks is that it can be a micro micro optimization over the regular function

but don't use it only for that micro optimization i use static closures when i want to make sure that this variable is not available and accessible in that function this is it for this

# **2.8 Encapsulation & Abstraction**

let's talk about the four principles of

object-oriented programming which are encapsulation – obstruction - inheritance and polymorphism

A screen shot of a computer

Description automatically generated

(Lập trình hướng đối tượng bao gồm tính kế thừa, tính đa hình, tính đóng gói, tính trừu tượng)

A screen shot of a data

Description automatically generated

(Tính đóng gói: data sẽ được để phạm vi truy cập ở private hoặc protected, hạn chế việc truy cập từ bên ngoài phạm vi lớp, thay vào đó khi cần truy cập tới phải thông qua các phương thức ở phạm vi public)

it hides the internal representation or the state of the object which protects the integrity of that object encapsulation ensures that your object manages its own state and nothing can change that unless it's explicitly allowed let's take this transaction class as an example here

i have a single property called the amount which is currently set to public and as you know we have three visibility access modifiers public private and protected

public simply means that the property or the method is accessible outside of this class by anyone interacting with the object private means that the property and method can only be accessed from within the class

itself and protected is pretty much the samething as the private but it extends tothe child or thesubclasses which we'll cover once we get to the inheritance so don't worry about that for now setting properties to public like this can break the encapsulation principle

(thuộc tính nếu để phạm vi truy cập là public thì sẽ phá vỡ quy tắc đóng gói)

A screenshot of a computer

Description automatically generated

(khởi tạo 2 phương thức set và get. Set method để set giá trị cho thuộc tính. Get method sẽ lấy giá trị của thuộc tính. Chúng đều có phạm vi là public để có thể truy cập từ bên ngoài.)

we're able to change the state of our object by simply changing the amount property to something like 125. now if we refresh the page we see that now it says that it's processing 125 transaction which is a problem right we've passed the 25 in the constructor and then we're allowed to explicitly override

that and change the amount properly which directly affects how the process method is handled and even though this might look harmless to you now and you might be thinking that yeah i would never do this i would never pass a 25 here and then right after change it to 125 but you cannot guarantee that when you're working in teams also think about it in a different way what if the transaction object was given

to you in a method so let's say you were inside a method and it accepts an argument where this transaction object is passed to you and then if you have ability to override and change the amount within that

bad or whether you should use it or you should not use it i'm going to leave that up to you to decide

(getter và setter có thể phá vỡ tính chất đóng gói, có hại nhiều hơn lợi)

A screen shot of a computer

Description automatically generated

(Như ở đây, khi khởi tạo 1 đối tượng từ Transaction, ta đang muốn làm việc với giá trị 100, nhưng nếu ta dùng setter và truyền vào giá trị =! 100 thì ta làm mất đi công việc đang làm hiện tại)

but the way i go about this is that if I have a property that represents the state of the object

i make it private and i make sure that I don't have another method that changes that state of the object

instead i accept the value through the constructor or through the method that

we call instead of the setter so in this case i would simply remove the set amount and we're already accepting the amount in the constructor and we're setting it there and if for whatever reason we needed to

change the amount then we should simply just create a new transaction object instead of changing the state of this object if we wanted to process a 125 transaction then we shouldn't be mutating and changing this object we should be processing a different transaction right and about the getter method here which gets the amount property i would simply leave that in if i need to access the amount

but if I don't really need to access the amount i'm not going to leave that in just

because i have the amount property and maybe down the road i might need to access the amount property so i'm going to remove that for now because we don't need it and let's change this back to being passed in the constructor and let's move on there are cases of course where the getters and setters make sense in some cases you might have some additional logic that you perform before

you set the property or before you get the property like some validation or formatting for example

and so on in those cases then setters and getters do make sense also with setters you can do method

chaining where you kind of build up your object by calling bunch of methods which gives you this fluent interface and you'll see that as a common practice in some cases it's also perfectly fine to have public properties and you'll see them be used in popular frameworks so there are definitely some use cases for them one use case is for the data transfer objects or in short dtos

which we'll cover later in this course as well my advice here would be to not define getter and setter methods for every single property unless and until you actually need them the point is don't just create setters and getters just

(Chỉ định nghĩa getter và setter khi thực sự cần thiết)

because you have a property in addition to protecting your properties you might also have some methods that are not meant to be exposed or publicly available they might be internal implementation so you would

want to hide those as well so for example within the process method we might be calling some other methods right we might be calling something like generate receiptn which maybe just generates the receipt of the transaction and then maybe we have a method that sends the email now we can create these methods here just to demonstrate but these methods are internal implementations and they

should not be set to public if you set this to public that means that this method can be called without

ever calling process outside of this class which can cause some issues right you might be generating some receipt

here or if you set this to public as well you might be sending an email before actually generating the

transaction and so on that's why because these are internal implementations you would want to set these to either private or protected and we'll talk about the protectedly in the later lesson but you would want to restrict direct access or public access to these methods and speaking of restricting access to private and protected properties and methods i want to show you a quick trick on how you could actually break the encapsulation and access private and protected properties and methods

while it is true that it will throw a fatal error if you try to access restricted properties and methods you

can actually still get the access to those restricted properties and methods through something

called php's reflection api so for example if i go over to index.php and try to access amount property which is private i would get the fatal error right

(Có thể truy cập các thuộc tính và phương thức private thông qua Relection API)

so let's now try to access this private property using the reflection api and we'll cover the reflection api in

more detail later in the course just bear with me here reflection api basically just adds the ability to

introspect your classes so we could do something like reflection property equals new reflection property and the first argument we pass the fully qualified class name transaction class and the second argument is the property name and in this case that's amount and now we can simply do reflection property set accessible to true now this does not make this amount properly directly accessible on the transaction object we'll still get the fatal error but now we can do something like this let's vary dump reflection property get value and we pass the object on which we're getting that value and that is transaction object

let's refresh the page and as you can see we have accessed the private property amount and it's printing 25 right here

A screenshot of a computer

Description automatically generated

now if we can access the privateproperty can we modify that private property the answer is yes we can do something like this reflection property set value and we pass the transaction object as the first argument and second argument would be the actual amount so that's 125.

if i refresh the page we have changed the 25 which was the original amount to 125. so as you can

see we're not really restricting anything by setting properties and methods to private or protected but the

point of private and protected access modifiers is to let the one know interacting with the object that certain properties and methods are meant to be kept private or protected they are internal implementations and they are meant to be used within that class

don't worry too much about this because in most cases it would not matter i just wanted to share this trick with you to show you how you could actually access private and protected properties and methods you probably wouldn't actually do this in a real application all right so let's remove this and let's move on to obstruction there are different ways to interpret the term obstruction and some refer to obstruction

as abstract classes and methods but the obstruction principle from the four principles of object-oriented

programming refers to a different concept and is not the same as abstract classes and methods

yes you could implement obstruction using abstract classes and methods

but it's not the same thing obstruction is actually closer and goes hand in hand with encapsulation instead

of abstract classes and methods we'll cover abstract classes and methods later

but obstruction simply means that internal implementation details of an object are hidden from the user

you could call a method on the object maybe provide some input and maybe get an output but you don't care how the method is actually implemented and how it works behind the scenes

all you care about is that you just call it and you expect some kind of output or some kind of result for example when we call the process method here we don't care what goes on behind the scenes we don't care what payment gateway this process method connects to we don't care how it's saved in the database we don't care whether it sends an email or not and we don't care how it generates the receipt all we care about is that we process the transaction this is what obstruction is in simple terms you're basically hiding the implementation details from the user you could change the way transaction is

processed maybe you could change the payment gateway or maybe you now decide that you don't want to send the email here or something like that or maybe you change the way the receipt is generated

and so on the point is that the place where you call the process method does not and should not care what goes on within that process method and any changes to that process method should not affect

this part of the code here setting properties to public breaks the encapsulation as we mentioned

before right but if you set this to public it also breaks the abstraction

because for example we have this amount property set to public and we have used this amount property in maybe 100 other places in our code base let's say that later we decided that we wanted to change the type of this property to something like integer

because maybe we're now expecting it in cents instead of in dollars or something or maybe you changed it into string or whatever basically you made some kind of change in the definition of this property now your code will blow up in 100 different places or you will have some unexpected results because you're directly accessing this property so basically if you try to build your classes in a way that does not break

encapsulation and does not break obstruction you will be on track on achieving a good object-oriented design there is a question that you could ask yourself when building your classes can you make changes to this class without breaking too many things outside of this class if the answer is yes then you have a

well-designed class and maintaining and extending that code base and that class

will be much easier so what's the difference between encapsulation and obstruction

because they have some things in common right we are working with the public and private properties here we've talked about that setting public breaks the encapsulation but now it also breaks the

obstruction and so on right so the encapsulation hides the internal state or the information while the obstruction hides the actual implementation of it i know i said a lot and don't worry if something sound a bit confusing and some things don't make sense take your time and watch this video as many times as needed also as i mentioned before there are certainly some use cases for public properties and setters

and getters just keep encapsulation and abstraction principles in mind when designing and building out your classes before we wrap up this video i want to show you one last thing as a bonus in this lesson

A screen shot of a computer

Description automatically generated

let's create a new transaction and pass hundred as the amount and let's comment out the process from

here and it's refresh and as you can see

it prints hundred and it prints true which means that we're able to access the private and protected properties and methods of that object

A screen shot of a computer

Description automatically generated