* - Now we get to the really interesting part of objects .
* Creating objects blueprints or templates using classes .
* Side notes, classes are relatively new introduction to JavaScript but they've been around in other programming languages for a long time .
* So if you have previous experience with classes in other languages, this will all look very familiar to you .
* Okay, say you have a series of objects and you want to make quick work of setting them up .
* Verbosely declaring each object the way we've done so far would produce a lot of code and a lot of repetition since each property and method would have to be defined for each new object .
* A better solution is to build some sort of template .
* And for that, we have classes .
* **Classes work as templates for an object type .**
* And anytime we create a new object based on a class, that object automatically gets all the properties and the methods from that class .
* That means we can change the properties of the class or the methods of the class .
* And those changes apply to every single instance of that class that we've created .
* In the exercise files for this movie, I've made some significant changes .
* There's a new file called backpack .
* js that exports a new backpack class .
* You can see the class here, it's called backpack .

Graphical user interface, text, application

Description automatically generated

* And at the very bottom, it says export default backpack .

Graphical user interface, text, application, email

Description automatically generated

* Inside script .
* js, we are importing that backpack class from backpack .
* js .

Text, application

Description automatically generated

* And then we use it to create a new backpack object here .
* And output the values of that object in the browser just like we have before .
* Finally, in index .
* html, I'm importing both of these JavaScript files, backpack .
* js and script .
* js and setting them up as **modules** .

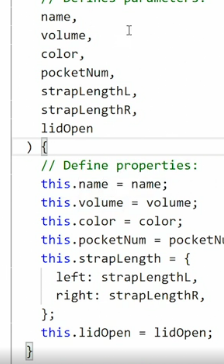
Text

Description automatically generated

* So remember from the beginning of the course that we talked about using **modules to have files depend on each other, that's what we're doing here .**
* And we can do it because a class is something that is only used internally in the JavaScript .
* We don't need to gain access to the class from the browser through the console or anywhere else .
* So class is a perfect candidate for using modules .
* Let's see how all this fits together .
* **To create a class**, we start with the **class** keyword followed by a capitalized name .
* This is a naming convention to ensure we know we're now looking at a class instead of just a regular object .
* There are actually two ways of declaring a class .
* You can either do this, what you see here .
* Which is called a **class declaration .**
* Class followed by a name .
* Or you can set up a **class expression** where you create a constant with a name again, capitalized and then you set it equal to class .

A picture containing text

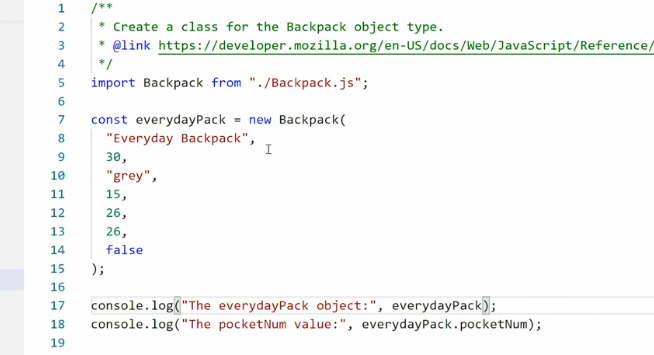
Description automatically generated

* And then the curly brackets .
* These two are just two different ways of doing the same thing .
* And the choice between a declaration and an expression is mainly one of preference .
* The trend currently seems to be, to use an expression but there's no reason to use an expression over a declaration .
* They are the same thing .
* Inside the class, ***we use a constructor method to literally construct the object created from that class .***
* *The constructor method first defines the parameters for each of the properties that's done inside the parentheses here .*
* So you see here's a list of all our properties .
* Then inside the curly brackets, it defines all the properties and set their values to the parameters have been passed in from the class .
* 
* This constructor is a method inside the class and it uses these parameters to populate these properties down here .
* Everything here works the same way as what we've seen before .
* And we use this keyword to point to the current object .
* Then we use dot notation to go to the property in question and set the property value equal to whatever P value was passed in through the parameters .
* We can also **add methods to classes .**
* So they are applied to all objects created from that class .
* To do so, we add the methods after the constructor method .
* So this curly bracket here wraps around the constructor method .

Graphical user interface, text, application

Description automatically generated

* After that curly bracket, we then declare the toggle lid method and the new strap length method .
* These methods will then be appended to the object we've created once we use the class to create an object .
* Now that we have a class in place, **we can use it to create those new objects** and that's done the following way .
* First, we create a new variable .
* In this case, a **constant** called *everyday pack .*
* Then we fill it or set it equal to a new backpack .
* So this is how we use a class to create a new object .
* We say the word ***new***, this is a keyword .



* And then we point out whatever constructor we want to use .
* In this case, the backpack class .
* And then we just pass in the values for each of those properties as parameters .
* So here we have the name, the volume, the color, the number of pockets, the length of the straps .
* And finally, the lid opens status .
* And these match up to these properties .
* Name, volume color, pocket number, strap length, and lid open .
* This piece of code here creates a new object for us .
* And that object becomes available just like the object we've worked with so far .
* That means we can now console log out the everyday pack object entirely .

Graphical user interface, text, application

Description automatically generated

* Or we can go in and get the everyday pack pocket number properties specifically from this object .
* So if I take this and run it in my browser, you'll see here we have the everyday pack objects .

Graphical user interface, text, application, email

Description automatically generated

* This is a full object again .
* Only this time it tells me this was generated from the backpack class .
* You can see it says backpack at the top .
* And then down here, we have the pocket number value .
* 15 from that object .
* To create new objects, all you have to do now is just create a new constant with a new name and then say new backpack and pass in the values and you'll create a whole new backpack .
* So now you can generate many different objects without having to verbosely declare those objects in your main file .
* One thing to note here is ***you can only use a class after it has been declared .***
* Remember, JavaScript reads from the top down .
* You have to make sure the class is declared before you start using it .
* *A good way of avoiding this problem altogether is to place your classes in a separate file and importing them because all imports need to happen at the top of the main file where it's being used .*
* So that way, we know the backpack class has been declared before we use it because the import happens at the very top .
* Using a class to create an object template allows us to define the property and methods structure for all the objects created with that class without having to redeclare them over and over .
* And if a property or method needs to be changed, you can change it in the class .
* And then all of the ancestors .
* So all the objects that have been created from that class will also update automatically .