## **OOP PRINCIPLES**

### **Introduction**

By this point, you will have learned and had a chance to practice the most common object-creation and organization patterns in JavaScript. But that is just the *tip* of the iceberg. More important than learning the syntax for factory functions or modules is figuring out how to use them effectively.

This whole series of lessons has been about the “Object Oriented Programming” paradigm (OOP). The basics of creating objects and classes are relatively straightforward. But it is not straightforward to decide what to put in each object, or when to make a new object, or when to let an object ‘inherit’ from another one.

### **Learning Outcomes**

By the end of this lesson, you should be able to do the following:

* Explain the “Single Responsibility Principle”.
* Briefly explain the additional SOLID principles.
* Explain what “tightly coupled” objects are and why we want to avoid them.

Luckily there are several concepts and principles that can guide us into making good decisions when it comes to our objects. This lesson is an introduction to the most important of those concepts. Keep in mind that there is not usually a very clear answer to your application design questions. Some patterns and ideas are obviously better than others, but there is often some trade-off when deciding where to put a specific function. In other words.. These principles are not *rules*- they’re helpful guidelines.

As you read these resources, it might help to go back to some projects you’ve already done and think about how what you’ve written measures up to the examples you see. And of course, as you move on, keep these things in mind when crafting new projects.

### **Single Responsibility**

As you craft your objects, one of the most important things to remember is the Single Responsibility Principle which states that a class (or object or module.. you get the point) should only have *one* responsibility. This doesn’t mean that an object can only do one thing, but it does mean that everything an object does should be part of one responsibility.

Here’s a really common example. Most of our code has functions to update and write things to the DOM in addition to our application logic. It’s a *really* good idea to separate your DOM stuff from the application logic.

So instead of this:

function isGameOver() { // game over logic goes here! if (gameOver){ const gameOverDiv = document.createElement('div') gameOverDiv.classList.add('game-over') gameOverDiv.textContent = `${this.winner} won the game!` document.body.appendChild(gameOverDiv) } }

You should extract all the DOM manipulation into its own module and use it like so:

function isGameOver() { // game over logic goes here! if (gameOver){ DOMStuff.gameOver(this.winner) } }

In fact - the function isGameOver shouldn’t be calling the DOM function anyway. That should go elsewhere (directly in the game-loop).

Another way to think about the Single Responsibility Principle is that a given method/class/component should have a single reason to change. Otherwise, if an object is trying to have multiple responsibilities, changing one aspect might affect another.

The Single Responsibility Principle is the first of a commonly found set of 5 design principles called the SOLID principles. You will read more about these principles in the assignment articles below.

### **Loosely Coupled Objects**

Obviously, all of our objects are intended to work together to form our final application. You should take care, however, to make sure that your individual objects can stand alone as much as possible. Tightly coupled objects are objects that rely so heavily on each other that removing or changing one will mean that you have to completely change another one - a real bummer.

This one is related pretty strongly to ‘Single Responsibility’ but takes a different angle. As an example, if we were writing a game and wanted to completely change how the User Interface worked, we should be able to do that without completely reworking the game logic. So we should be able to start off writing our game using primarily console.logs() and then add in a bunch of DOM functions later without touching the game logic.

### **Assignment**

1. The following articles mention the acronym SOLID before going on to talk about Single Responsibility. Single Responsibility is definitely the most relevant of the 5. Feel free to dig into the rest of the SOLID principles if you like.. but pay special attention to Single Responsibility.
   1. Read [SOLID JavaScript: The Single Responsibility Principle](http://aspiringcraftsman.com/2011/12/08/solid-javascript-single-responsibility-principle/). NOTE: This article does make use of JQuery, one of the earliest and most popular JavaScript libraries prior to the ES6 standard. While The Odin Project does not teach JQuery and you are not expected to understand the example, be sure to focus less on the code itself and more on the SOLID concepts being expressed.
   2. [5 Principles that will make you a SOLID JavaScript Developer](https://thefullstack.xyz/solid-javascript/) hits the same topic, and also covers the rest of ‘SOLID’ concisely.
   3. And read [S.O.L.I.D. The first 5 principles of Object Oriented Design with JavaScript](https://medium.com/@cramirez92/s-o-l-i-d-the-first-5-priciples-of-object-oriented-design-with-javascript-790f6ac9b9fa) for good measure.
2. [How to Write Highly Scalable and Maintainable JavaScript: Coupling](https://medium.com/@alexcastrounis/how-to-write-highly-scalable-and-maintainable-javascript-coupling-c860787dbdd4) explains loosely coupled objects pretty well.

### **Additional Resources**

* The best book we’ve ever read on the subject of loose coupling is [Practical Object-Oriented Design In Ruby](http://www.poodr.com/). Unfortunately, it is not free.. and not JavaScript. We feel confident in recommending it anyway. If you don’t know Ruby, it is a clear enough language that you don’t really need to learn it to follow the examples and the content of the book is sincerely fantastic. Alternatively, [99 Bottles of OOP](https://sandimetz.com/products) is written in both JavaScript and Ruby. It is written by the same author and may be a better option if you are brand new to OOP (it is not free either).
* A nice summary of [OOP’s most common principles using JavaScript](https://medium.com/better-programming/object-oriented-programming-in-javascript-b3bda28d3e81)