**STRATEGY DESIGN PATTERN**

Here's a simple explanation:

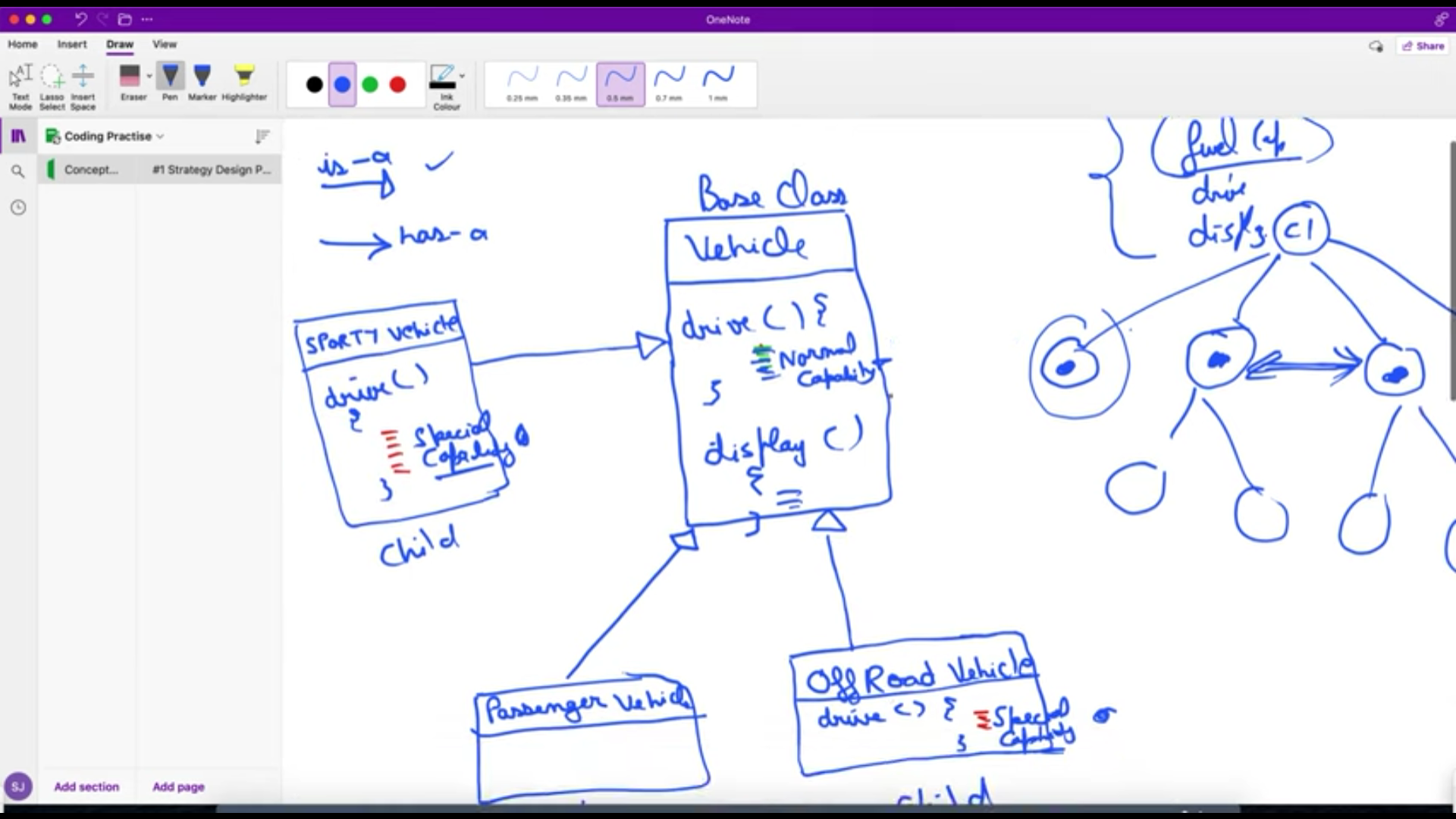
Imagine you have a task to perform, but there are different ways you could go about it. For example, let's say you need to sort a list of items. You could use various sorting algorithms like Bubble Sort, Quick Sort, or Merge Sort.

Now, instead of coding each sorting algorithm directly into your main program, you create a separate "strategy" class for each sorting algorithm. Each strategy class implements a common interface, let's call it SortAlgorithm, and provides its own implementation for sorting.

So, you have classes like BubbleSort, QuickSort, and MergeSort, all implementing the SortAlgorithm interface.

Then, in your main program, you have a context class, let's call it Sorter, which has a reference to a SortAlgorithm. When you need to sort a list, you simply pass the appropriate sorting algorithm to the Sorter, and it will use that algorithm to perform the sorting.

This way, you can easily switch between different sorting algorithms without changing the main program. You can even add new sorting algorithms without modifying existing code. This flexibility and interchangeability of algorithms is the essence of the Strategy Design Pattern.



**Case study:** In the above case, passenger vehicle child class can directly implement drive method from its parent without making another one for itself. But for sporty and off-road vehicle child classes, they need to override the parent’s drive class and make their own.

There’s a possibility that the overridden methods are same in both sporty and off-road vehicle classes. So, there’s duplicity in child classes and the body doesn’t exist in parent.

**FULL EXPLAINATION:** https://www.youtube.com/watch?v=u8DttUrXtEw&list=PL6W8uoQQ2c61X\_9e6Net0WdYZidm7zooW&index=7