1 - Command and Composite Patterns in Visual Studio Code Explorer Video - <https://web.microsoftstream.com/video/224547f6-9d10-44a0-a375-41cc5a2a5485>

2 - Freetube vs Minitube: Proxy Pattern Analysis - <https://web.microsoftstream.com/video/49367e39-d39f-4d5c-b8e3-95829f6aead5>

3 -Use of Factory and Observer in Chromium - <https://web.microsoftstream.com/video/babcb0ca-80ce-45c6-837d-0a4763ea4b45>

4 - Why patterns are used in Chromium - <https://web.microsoftstream.com/video/b1c8c25e-dc0a-4e75-a163-e39bfb59daac>

5 -GDevelop: Patterns found in Events functionality - <https://web.microsoftstream.com/video/8b312f12-e234-43ad-829e-9a1eb0b293e2>

6 - Deep Dive Into Observer Pattern - <https://web.microsoftstream.com/video/9ac70dc1-baa6-4bd1-ba2c-5d6a420c96e5>

7 -Design Patterns in React Front-End Apps - <https://web.microsoftstream.com/video/0c68b6bd-dc50-462b-a615-eab460a63082>

8 - Analysis of Factory Method and Singleton in Chromium Utilities (Bookmark, History, and Downloads) - <https://web.microsoftstream.com/video/8f6d2d45-e508-44c7-8523-f85e540cd6e5>

9- Dependency Injection and Command Patterns usage in Visual Studio Code - <https://web.microsoftstream.com/video/48744801-0c23-4bea-991c-1d33ae934598>

10 - Analyzing implementation of design patterns in Facebook-React/Jest - <https://web.microsoftstream.com/video/8c1604ad-1fc7-44b8-9bca-92bad1c158b5>11 - Implementation of the Strategy Pattern within Krita - <https://web.microsoftstream.com/video/8cc3c14e-f7e0-4210-93e8-599eaf740329>

12 - Command Behavioral Pattern in Angular Component Routing - <https://web.microsoftstream.com/video/ba2a3205-086e-46bb-b5c7-5010c357e2d8>

**Command and Composite Patterns in Visual Studio Code (Explorer Tool)**

* Mainly use for JS, or Web development IDE but does have an ecosystem that provide library and add-on for other languages.
* Focusing on Composite and Command Design pattern. Some other patterns can be found are Chain of Responsibility and Decorator. Visual Studio Code is written in TypeScript.
* Most of the Explorer Tool and found in Composite pattern and the reason is that every class they are using, there is always another class extend from it (ex. ExplorerView extends ViewPane, ViewPane extends Pane, Pane extends Disposable, Disposable is an abstract class).
* RegisterAction2 extends Action2 as an abstract class, which is also a Command class. RegisterAction class use the Command pattern when adding the Command into the Explorer. Register function inside RegisterAction class send an ID as a string to the command. If the command is already existing, it will ignore the process.
* The abstract class Action2 is a Command as it inherits as register action to create each tool in the explorer.
* The registerAction2 is a ConcreteCommand class as it is binding between a receiver object and action. In the app, it will be between the commandService (receiver) and the abstract class, the command.
* The client, which is the Explorer class set the commandService (receiver) with the registerAction2 (concreteCommand).
* The invoker, which is executeCommand is executed only when the run function is running and one of the tools is clicked.
* The receiver, which is commandService, knows how to perform operations associated with carrying out the request. commandService is just an object of command that know how to perform all the request.
* To add a tool into the Explorer tab using command design pattern, user will need to follow similar implementation's structure just as the other tools, create a new object using registerAction2 to register the tool which is an object of Action2. The tools must have an id, title and set f1 (if the tool is an event) to true or false, icon, etc., in the constructor. After that, create an object that inherits from interface commandService (receiver) and call executeCommand (invoker) to run it.
* If user wish to add a tool or features but doesn’t know where to start, best is to read the code and understand how they implement it. Since the format for each tool/feature are similar (only needs to change the entity), user can follow what is already implemented and create a new one.
* When dealing with collections of objects, there are often operations that appropriate for both single object and the entire collection. Considering the delete operation, where user can delete a single file or delete an entire folder, or many files at the same time. When that happen, composite pattern exists to deal with inner file side the folder.

**Freetube vs. Minitube: Proxy Pattern Analysis**

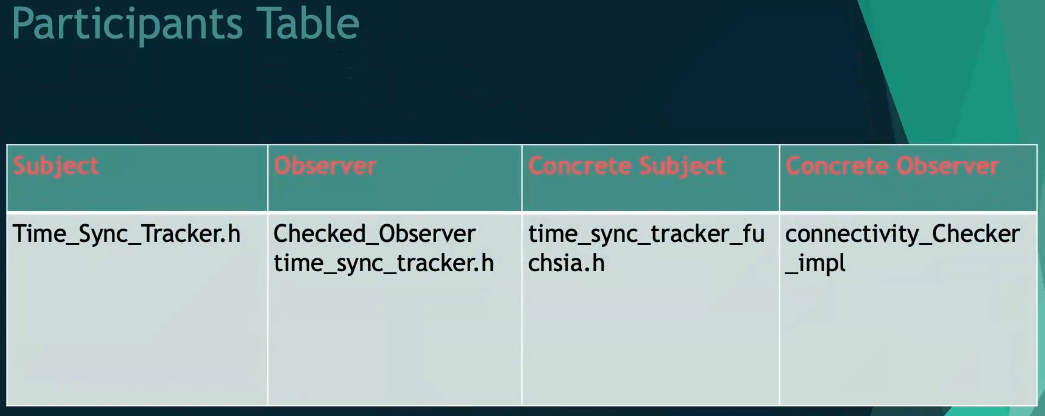
* **Remote Proxy:** the RealSubject that client is trying to access is not locally available
* **Virtual Proxy:** take place of a large object and loads it when requires
* **(05:05) Data Scraping**: copying specific data from the web into central local database for later retrieval or analysis. Typically, these 2 sites are pulling data information from Google’s YouTube API and sometimes directly call on the API for things such as channel info.
* **(06:00) Minitube:** client request a video to watch on Minitube, the Minitube then request the YouTube video via a ProxyClass. Most of the time, the ProxyClass will only return the YouTube video as “on demand”, or like when the client request, then it will fetch and load. Because the Real YouTube video we are requesting is not store locally, this makes the ProxyClass a **Remote Proxy** and **Virtual Proxy** at the same time.
* **Constructor**: no data is initialized except videoID as it doesn’t store the video locally
* **GetVideoInfo**: send an HTTP request to the actual video on YouTube and pass the data to the local member. Note that the function is using HttpUtils:stealthAndNotCached() as it will just get the video data and doesn’t allowed cached to the local machine. Because of that, whenever the user wants to watch the same video again, the same step applies.
* **LoadStreamURL**: like GetVideoInfo, except it is taking the webpage of YouTube video, not the data anymore and scrapes the video itself from the webpage.
* **StoreInfo:** User can subscribe to a channel because as soon as user click on the subscribe, it will save the channel data locally to a database which the application will be using when it’s load on the next time.
* **(08:45) Freetube:** doesn’t directly use the API like Minitube, but it does use some of the open-source work for each of the task such as download, subscribe, … where it can get the video information and store it locally, ready to be played “on demand”. Each of the open-source data kind of work as a proxy in the app. When the user subscribes to a channel, a channel ID will be store locally and the same process just happen the same with MiniTube. The open-source data that does the subscribe will take the channelID stored locally and compare with the channel online ID. If they are match, it means the user has subscribed.
* **Similar:** Both are using combination of remote and virtual proxy. They use proxy to store and retrieve the data base on the demand of user. Then they recreate the different user views on demand from that representation.
* **Differences:** MiniTube using direct call to the Google’s YouTube API while FreeTube use an extra step to conceal the information, where it is hidden behind a proxy agent to make a call via ytdl, where it will use another proxy to make a call with Google’s YouTube API
* **Consequences:** allows the app to load faster as it doesn’t have to load anything, the data can be load when user make a request. Because of that, many variants of YouTube exist because they can access the YouTube’s database via remote proxy implementations.

**Use of Factory and Observer in Chromium**

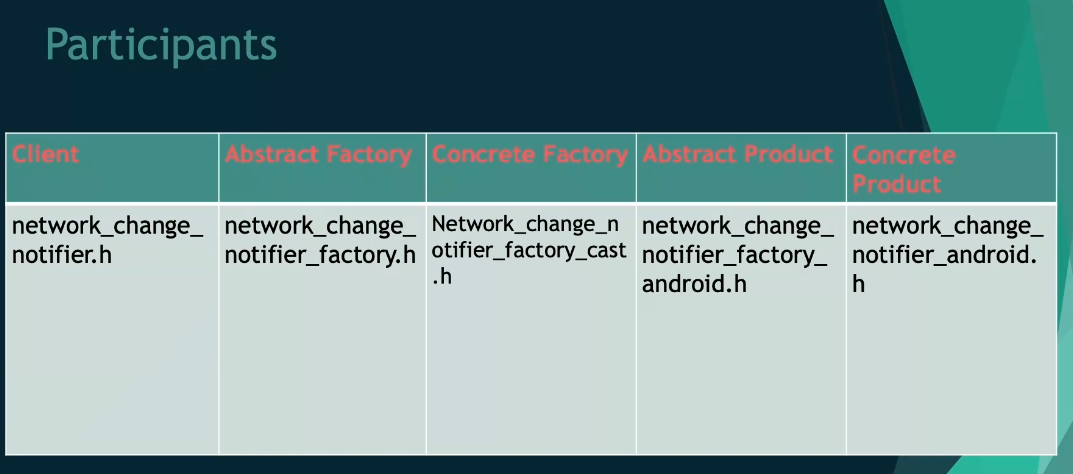
* Chromium is a Google-sponsored project, which is widely used by many browsers including Google Chrome itself and Microsoft Edge.
* **Simple Factory:** an object creational pattern, call by client through a static function. Client will pass a parameter for object creation, which it’s already know what kind of object is creating and expect a response from the factory function.
* **Factory Method:** in charge of object creation and management, use when the program has multiple families of objects.
* **(05:45)** **Factory Pattern:** Client will send a parameter for object creational though PlatformNotificationServiceFactory class. When the PlatformNotificationServiceFactory class receive the parameter, which is a BrowserContext object, the Client is expecting to receive a response object in KeyedService type. The class PlatformNotificationServiceFactory is a Singleton.
* **(09:20) Observer Pattern:** also known as Publish-Subscribe. The Observer class is a NotificationObserver class, and the Concrete Observer class is the NotificationSystemObserver class. The Subject class will be NotificationServiceImpl, which has its Notify type, AddObserver or RemoveObserver. The Concreate Subject class is Notification Source.

**Why Patterns are used in Chromium**

* Chrome is an improved version of Chromium, which contain many features such as Automatic Updates, unique and licensed media format (AAC, H.264, MP3...), Flash API plugin, ...
* Chromium is a based, or foundation for many open-source browsers to use, including MS Edge, Brave, Vivaldi, …
* Design Pattern using: Factory, Observer and Singleton.
* **(04:35) Observer Pattern:** The implementation is done using abstract observer, which make the observer can be use anywhere an observer is needed. The benefit of using this is the observer pattern allows to create many observers that can view multiple values and essentially be used to observe any object, where Chromium and Chromecast are using. It can also adapt to any interface and create multiple objects of itself to attach to different values and allows it to be used without the need of creating multiple observer class.



* **(09:00) Abstract Factory Pattern:** use in creating notification for client when there is a change in the network. The client will call the factory to create a product, then goes to abstract network change notifier factory. Then the network change notifier inherits its based class and create its own factory to create a product. Client will call upon the Abstract Factory to deal with the creation of Client. From there, the Abstract Factory decide which concreate factory to go to. The benefits are using this can make it easier to create many objects with similar functionality, hidden implementation from clients, promotes consistency between products.



* **(11:50) Singleton Pattern:** used in Chrome cast base. It was implemented in Java while GoF used C++. It could be useful to create a unit object that use for checking/testing and reduces to be used as 1 instance and control the global variables in the code.

**GDevelop: Patterns found in Events functionality**

* An open-source, cross-platform game engine. Allow users to create their own game without programming language knowledge.
* Initially developed in 2008 using C++. In 2013, the web version released, with the language porting from C++ to JavaScript. In 2018, GDevelop 5 is released with the implementation of IDE to use Pixi.JS and React. Currently, it is still using GDJS engine but including the Pixi.JS for graphics support (shaders, 2D rendering) and material-UI (React Framework and UI Design)
* Event Driven is to define all functionality in the game, including logic of the game
* **(04:20) Singleton Pattern:** exist in EventsCodeNameMangler.h and in the .cpp, it will check if the instances have been created. The benefits of using this is to avoid multiple calling creation, control the access and avoid the use of global variables to store an instance.
* **(07:15) Prototype Pattern:** implemented a virtual clone() in the base class and allow its child class to modified in their own way. Some of the child class include: WhileEvent, GroupEvent, ForEachEvent, … That is useful when you have multiple While/ForEach loop in the program and doing this can avoid multiple creation/define. The class are added to the project dynamically at run time and cloned whenever needed, this can avoid of getting another class to run when the clone process is need which will take more time and resources. It is also flexible when creating a new event, with new conditions and new actions, however, it must have a clone() function as other events.
* **(14:20) Composite Pattern:** used in BaseEvent class and its concrete class, such as StandardEvent, ForEachEvent, GroupEvent, WhileEvent, RepeatEvent,… All the concrete class of BaseEvent will include an object of BaseEvent.

**Deep Dive into Observer Pattern**

* Tachiyomi is a free open-source manga reader for mobile devices. Allows to track, read and download the manga from multiple sources.
* **(04:40) RxJava Implementation of Observers:** Rx is stand for ReactiveX in Java and it is using Reactive pattern, an upgrade of Observer pattern. Because of that, the Subjects are now called Observables. Using RxJava when there is a constant source of data coming to the environment (GPS data, RSS feed, etc.,). The Observers dynamically react to change this changing data and create a unifying interface for composing data. Almost any data type can be used as an Observable. With the RxJava, instead of notifying the other observers like what Observer design pattern should do, it will not be doing that and instead call the onNext() method to handle the next event. It does also have an onError() and onComplete() method.
* Download Manager is responsible for building a list of pages that its need to download. Consider that Download Manager is creating an array of work. It will create an observable list of page or chapter to an Observer and manage the downloaded status to either load the downloaded chapter or begin the download. When there are multiple downloads, the download service is interacting with the download controller in order to know when its turn is. The downloader subscribes to the downloads relay and iterates over the list of observables that were admitted to it and initiated for every chapter emitted in the relay. The downloader will then go over the list again to ensure the chapter that need to be downloaded has been downloaded and mark the status.
* The Download Presenter is a Model View presenter for the download controller. The Download controller subscribes to the download progress observables through this Presenter. The Presenter get a list of the Observables downloads from the download queue that allow the controller to view any state changes of downloads. The controller subscribes to the download service to keep track of the status of each download. The controller also subscribes to the observables handed down through the Presenter. By doing this, the Download View, or Presenter can see the progress and status of each download and display that to user.
* The Download Queue retrieves observable download from the download queue. Using the getProgressObservable() to iterator over all the download observables in the queue and filter what is left to be downloaded to display.
* (13:33) Misuse of Observables: The Page Holder use the Observables to track the download progress of an image. Instead of using the observables to track for the percentage of the image that has been loaded, the observable they are implementing is checking for every 100ms. Doing this is defeating the purpose of Observe design pattern in the way that they are implementing an observable to track the download %, but instead of getting the information from the download, the page is manually checking.

**Design Patterns in React Front-End Apps**

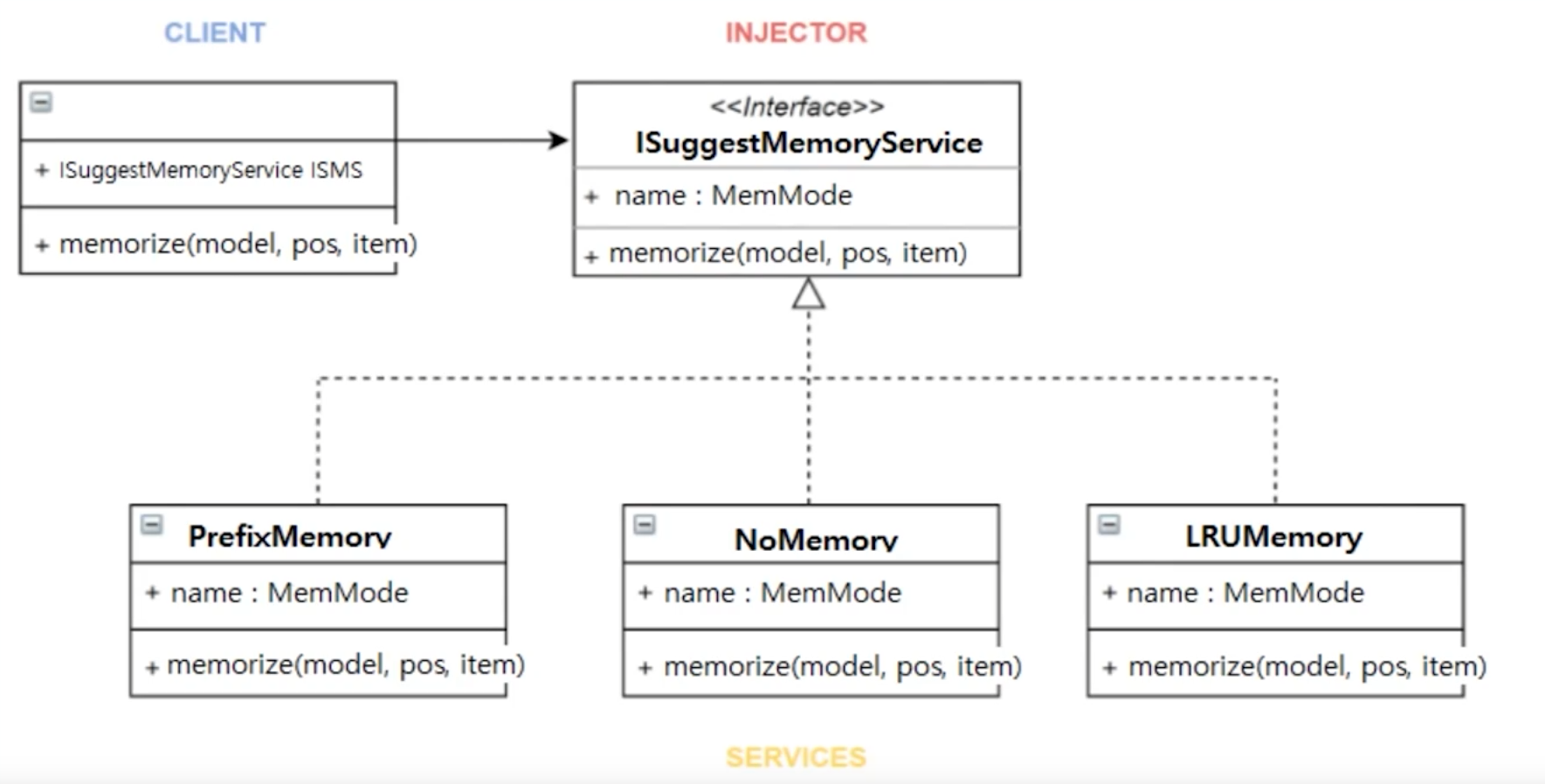
* Decorator pattern is replicated by a built-in feature of the framework. Higher Order Components are functions that take a component as a parameter and return a component with new behavior. Since React is a JS library, the participants involved in the pattern can be vary, unlike C++ where all member such as AbstractComponent or AbstractDecorator are required.
* The ConcreateComponent still defines the base object which will have new behavior added to it. The ConcreteDecorator are the Higher Order Components and define the new functionality that will be added. Applying multiple Decorator to one component requires passing it through multiple higher order components.
* (05:00) Memento Pattern: Mostly use to store user’s input so in case where user perform an undo action, it can restore the data. In the React case, when the user clicks on the create Memento, it will send what is currently in the text area to the createMemento function and then temporarily put the data to savedState. When the user wants to restore the data, simply take whatever is currently in savedState and put it into the text area
* (08:30) Bridge and Mediator Pattern: The functionality of a button can be considered as the abstraction and the implementation is how the button looks. The mediator functionality is to register and established communication with the colleague that is currently in the list of user’s arrays. In order to achieve that, they must use JS prototyping to add methods to every instance of mediator. In each of the “colleague”, there will be a function to update the receiving message, the send message function…. So that when 1 colleague send a message to the other, the other can receive and show the result right away. Everything will be done through a Mediator object.

**Analysis of Factory Method and Singleton in Chromium Utilities (Bookmark, History and Downloads)**

* Chromium use Factory Method to allow subclasses (provide more utilities), increase code efficient (reusage enable) and easy to maintain structure between utilities. By providing a base class, it can ensure every utility can do the work correctly, or not, closely to correct because it is ensuring all function is required is implemented.
* Chromium use Factory Method to control the Singleton class for each of the concrete class. The interface here is the ProductDefaultSingletonTraits, which include Singleton concrete subclasses of DownloadCoreServiceFactory, TopSitesFactory and BookmarkModelFactory. Moreover, the DownloadCoreServiceFactory and BookmarkModelFactory is grouped by the Creator BrowserContextKeyedServiceFactory and the TopsitesFactory is grouped by the Creator of RefcountBrowserContextKeyedServiceFactory.
* Chromium use Singleton to limit the factory method creation to a single instance and ensure access is enable to client side. Because of that, all concreate class we see used in Factory Method above is instantiate to a unique factory instance.

**Dependency Injection and Command Patterns usage in Visual Studio Code**

* Code is organized around services in Platform layer. The services are injected via a technique called Constructor Injection.
* (02:10) Dependency Injection Design Pattern: outsources the creation of dependent objects to an external class. There is an Injector class that handle the incoming data types and instantiate the object of whatever the Client, the class that is request data is looking for. The Injector class can have multiple concrete class to handle the requesting type, and the data of the required class can be passing through a method called constructor injection where it would pass the parameters to the Client’s constructor to initialize. In order to achieve that, the Client class must be generic enough so that it doesn’t depend on any other class when requesting data. Benefits of using this is decouples dependent classes, easier to extend the concrete class (adding more options) and allow the developer to modify the class work easily. However, the downside of that is the code will be harder to read by another person
* The class ISuggestMemoryService is export as an external class, following the Dependency Injection design pattern rules. ISuggestMemoryService is also an interface in which other class can inherits and follow. Because of that, the concrete class who is following it must be injected by creating a pulling on object outside.



* Depends on which class Memory are using, the memorize function can be different to accommodate the memory request usage. NoMemory is use when there is no word/function to suggest, while PrefixMemory will understand the writing language and suggestion according from that.
* Normally, the command design pattern requires the concrete class to have an object of the required call function to be calling, according to GOF. However, in VSCode, the implementation is a bit different. The concrete class, which is known as CompletionAcceptedCommand doesn’t have an object of CompletionItemProvider, which contain the onCompletionAccepted() function to be execute. The process is done through a Command Manager class which will instantiate the CompletionAccptedCommand and bind that to the onCompletionAccepted() function. The reason for doing this is when using the Command design pattern, the concrete class usually need to use 1 or 2 function that is required to execute from a list of many functions in one of the external class. Using Command Manager allows the concrete class to know and only use the required function without touching any other function, which sometimes can cause problems.

**Analyzing implementation of design pattern in Facebook-React/Jest**

* Facebook – React -> It is a javascript library that is used for building user interfaces. It is great to use as it encapsulates states then allows you to create complex UIs
* The team focuses on the component details :

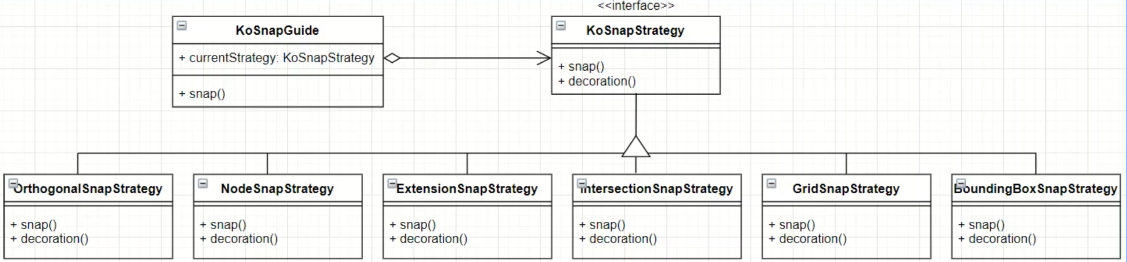
Config.build.js : Controls the default features flags that are used by bundle tests -> Contains many observers,

Don’t-run-jest-directly.js : Forces any react project to test scripts -> Proxy pattern, decorator pattern

SetupTests.js : Used to decode error messages -> Displays a state-like pattern, proxy pattern, singleton pattern,

**Implementation of the Strategy Pattern within Krita**

* Krita is an open-source software for graphic editor, including painting, sketching, and create 2D animation that is written in C++ (back-end) and Qt (front-end).
* Krita uses the Strategy design pattern to implement for snapping and object alignments features. KoSnapGuide class being the Client, where it has an object of KoSnapStrategy class, which is an interface class for the concrete class functionality. The snap() function in KoSnapGuide will be execute based on the KoSnapStrategy class snap() function, or any override concrete class. The KoSnapStrategy class will know which concrete class need to be call to fulfill the request from KoSnapGuide. There is multiple snapping method to use such as OrthogonalSnap, NodeSnap, ExtensionSnap, IntersectionSnap, GridSnap and BoundingBoxSnap.
* Since there are many snapping types, it does make sense to use Strategy as it will group the algorithms into certain type, in this case, SnapStrategy. Although they are performing different operations, the result shouldn’t be much of a difference, or it shouldn’t have any difference. When developers are doing Strategy, they give themselves more room in the future when they can add a new snapping method to the existing list. Since the KoSnapStrategy is starting along with the KoSnapGuide, which is the Client, the Client is able to see what snapping method is existing and change that, if client desire at the runtime. The concrete class will only run when the selected snapping types is selected.
* Another potential method for this implementation type is to use Template Method. This method is almost similar, when the template is carrying auto common code and have multiple algorithms placed in its subclass. However, one limitation of Template method is that it is hard to control when user is using multiple snapping method at the same time.
* Enforces Open/Closed Principle: Locks code for context modification and Open for extension when desire (adding more options).



**Command Behavior Pattern in Angular Component Routing**

Angular is a component based web application framework made by google that uses typescript and css.

This feature allows for a series of different type of commands to be run at any moment, and also allows them to be stopped to run a different command.

Invoker : Starts routing process through the html/ts file.

Receiver : app-routing.module.ts perfoms the routing operation -> Contains an array that has different types of routes to run

Command : RouterEvent carries out the request and initializes the concretecommand -> Contains 9 different types of events that can be triggered

On the client side -> We now see the new route on the screen.