Strategy design pattern

- In Strategy pattern, a class behavior or its algorithm can be changed at run time. This type of design pattern comes under behavior pattern.
- In Strategy pattern, we create objects which represent various strategies and a context object whose behavior varies as per its strategy object. The strategy object changes the executing algorithm of the context object.
- Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from the clients that use it.
- Capture the abstraction in an interface, bury implementation details in derived classes.
- A strategy is an algorithm. Strategies often have internal variables that record the state of the algorithm. At the end of the algorithm, the variables might record the result, but in general they are only meaningful during the execution of the algorithm. A strategy is either selected by an outside agent or by the context. A strategy tends to have a single "start" method, and it calls all the rest. There is a lot of cohesion between the methods of a strategy.
- In contrast, a state usually has no variables. ("State has no state", is what I say.) A state usually selects the next state of its context. A state tends to have lots of unrelated methods, so there is little cohesion between the methods of a state.
- Strategy - defines an interface common to all supported algorithms. Context uses this interface to call the algorithm defined by a ConcreteStrategy.
    - ConcreteStrategy - each concrete strategy implements an algorithm.
    - Context
        - contains a reference to a strategy object.
        - may define an interface that lets strategy accessing its data.

  The Context objects contains a reference to the ConcreteStrategy that should be used. When an operation is required then the algorithm is run from the strategy object. The Context is not aware of the strategy implementation. If necessary, addition objects can be defined to pass data from context object to strategy.

- The context object receives requests from the client and delegates them to the strategy object. Usually the ConcreteStartegy is created by the client and passed to the context. From this point the clients interacts only with the context.
- Participants
    - Strategy (Compositor)
        - declares an interface common to all supported algorithms. Context uses this interface to call the algorithm defined by a ConcreteStrategy.

    - ConcreteStrategy (SimpleCompositor, TeXCompositor, ArrayCompositor) ◦
        - implements the algorithm using the Strategy interface.
    - Context (Composition)
        - is configured with a ConcreteStrategy object.
        - maintains a reference to a Strategy object.
        - may define an interface that lets Strategy access its data.
- Collaborations
    - Strategy and Context interact to implement the chosen algorithm. A context may pass all data required by the algorithm to the strategy when the algorithm is called. Alternatively, the context can pass itself as an argument to Strategy operations. That lets the strategy call back on the context as required.
    - A context forwards requests from its clients to its strategy. Clients usually create and pass a ConcreteStrategy object to the context; thereafter, clients interact with the context exclusively. There is often a family of ConcreteStrategy classes for a client to choose from.