Strategy Design Pattern;

**Other name (if any)**

The Strategy Pattern is also known as Policy.

**What it does**

If classes follow parent child relationship then we will call it as "is a" relationship. Let's say there is a situation where sibling classes has the same functionality/ piece of code but their parent/base class does not contain that functionality. In that situation to reduce code redundancy we can use strategy design pattern. The Strategy Design Pattern defines a family of algorithms, encapsulates each one, and makes them interchangeable, allowing clients to switch algorithms dynamically without altering the code structure.

**Where to use**

1. We can use when we need different variations of an algorithm.  
2. When you need to dynamically select and switch between different algorithms at runtime based on user preferences, configuration settings, or system states.  
3. When the behavior of an application depends on external factors such as user preferences, system configuration, or environmental conditions, and you want to manage these dependencies in a modular and adaptable way.

**Steps**

1. We have to create strategy/functionality interface.  
2. Based on the category of the functionality other functionality specialised classes will be created which will extends this interface. The actual functionalities will be written in these classes.  
3. Now create a parent class. In this class we will declare an object of the parent interface. But the object will not be created in this class.  
4. Create child classes. This class will pass the functionality specialised classes from the constructor through super. So their parent will create the object based on this passing argument. Child will pass the argument with new operator.

**Special cases (if any)**

placeholder

**Advantages**

1. A family of algorithms can be defined as a class hierarchy and can be used interchangeably to alter application behavior without changing its architecture.  
2. By encapsulating the algorithm separately, new algorithms complying with the same interface can be easily introduced.  
3. The application can switch strategies at run-time.  
4. Encourages the reuse of algorithms or strategies across different parts of the codebase, reducing redundancy and promoting consistency.  
5. the Open/Closed Principle by allowing new algorithms or strategies to be added without modifying existing code, promoting code stability and minimizing the risk of unintended side effects.  
6. Decouples the selection and execution of algorithms or strategies from the client code, promoting loose coupling and improving code maintainability and flexibility.

**Disadvantages**

1. The application must be aware of all the strategies to select the right one for the right situation.

**Code**

placeholder

**Difference with similar pattern**

placeholder

**Diagram**

placeholder