**Strategy Pattern**

**Strategy pattern** is a behavioural design pattern that allows the behaviour of an object to be selected at runtime.

The Strategy pattern is based on the idea of encapsulating a family of algorithms into separate classes that implement a common interface. The pattern consists of three main components: the Context, the Strategy, and the Concrete Strategy.

The Context is the class that contains the object whose behaviour needs to be changed dynamically. The Strategy is the interface or abstract class that defines the common methods for all the algorithms that can be used by the Context object. The Concrete Strategy is the class that implements the Strategy interface and provides the actual implementation of the algorithm.

**The Strategy pattern provides several advantages, including:**

1. Flexibility: The Strategy pattern allows the behaviour of an object to be changed dynamically at runtime by selecting different algorithms.
2. Modularity: The pattern encapsulates the algorithms in separate classes, making it easy to add or remove algorithms without affecting other parts of the code.
3. Testability: The pattern makes it easy to test the different algorithms separately, without affecting the overall behavior of the code.
4. Open-Closed Principle: The Strategy pattern follows the Open-Closed Principle, which states that a class should be open for extension but closed for modification.
5. However, the Strategy pattern can also have some drawbacks, including increased complexity due to the use of multiple classes, and potential performance issues if the selection of algorithms is done frequently at runtime.