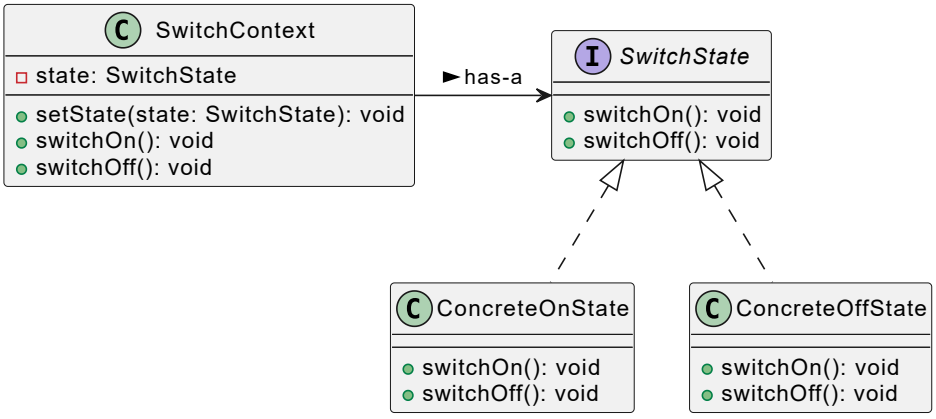
[Behavioral Patterns]

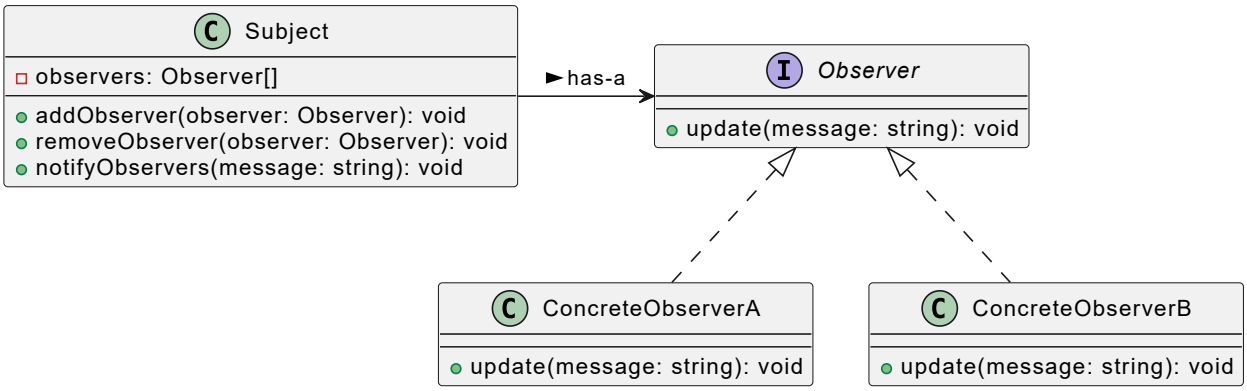
1. State Pattern

State pattern allows an object to alter its behaviour when its internal state changes.



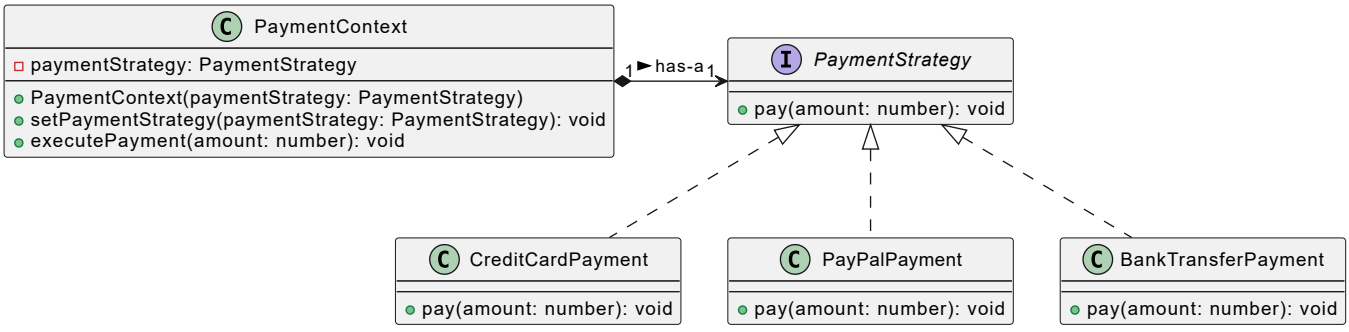
2. Observer Pattern

In **Observer Pattern**, an object known as **observable** maintains a list of its dependents, called **observers**, and notifies them of any state change, usually by calling one of their methods.



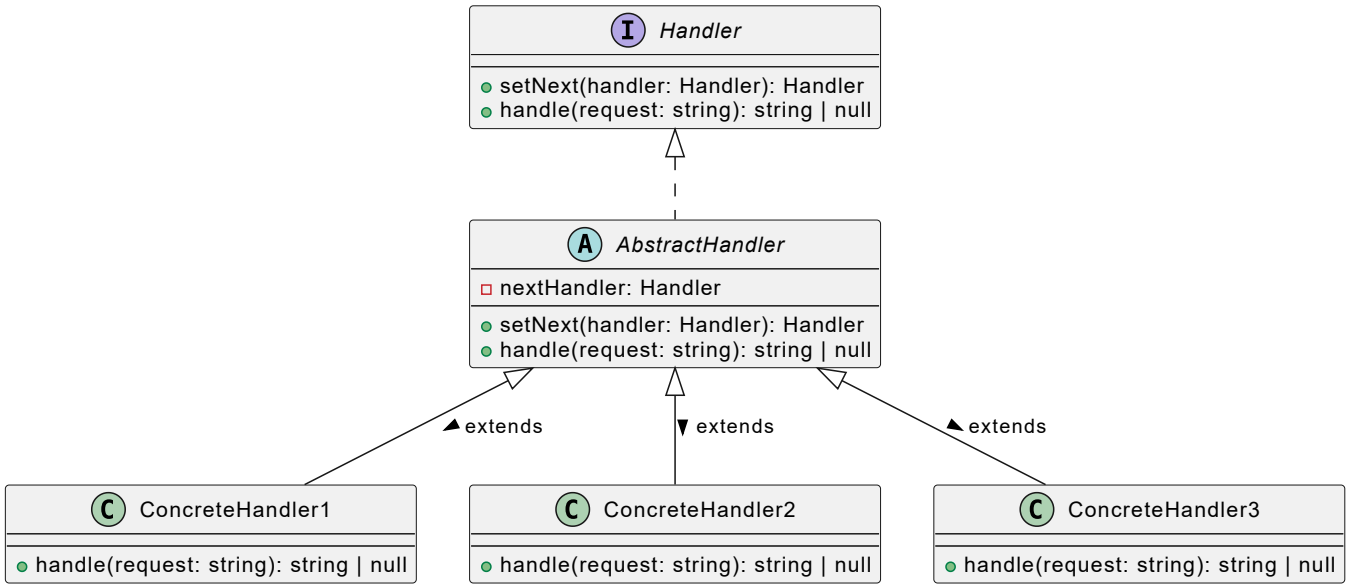
3. Strategy Pattern

Strategy Pattern allows us to define multiple algorithms to perform a specific task and select one depending on the situation or context. It encapsulates each algorithm and makes them interchangeable.



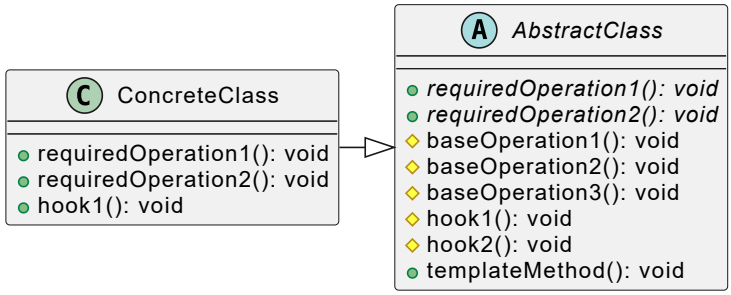
4. Chain of Responsibility Pattern

Chain of Responsibility Pattern allows multiple objects to handle a request without the sender needing to know which object will process it ultimately.



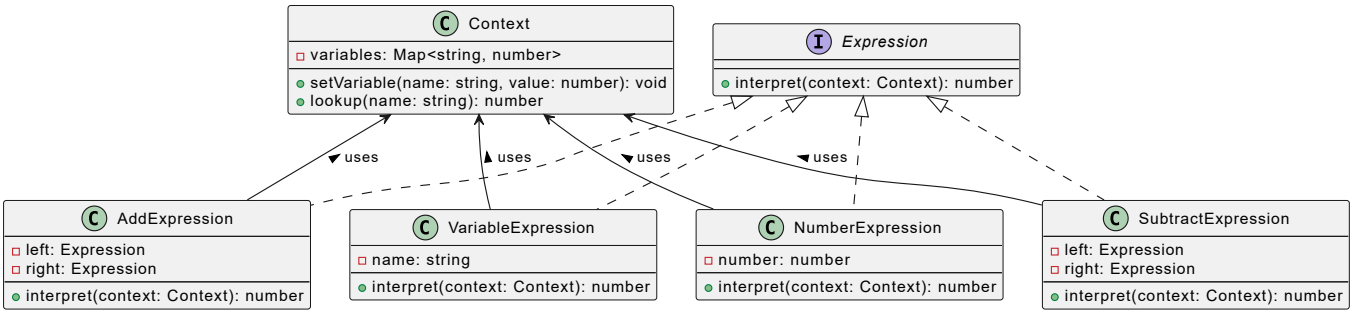
5. Template Pattern

The **Template Pattern** ensures that a number of classes follow specific steps to perform an operation but allows each step to have its own logic in that specific step.



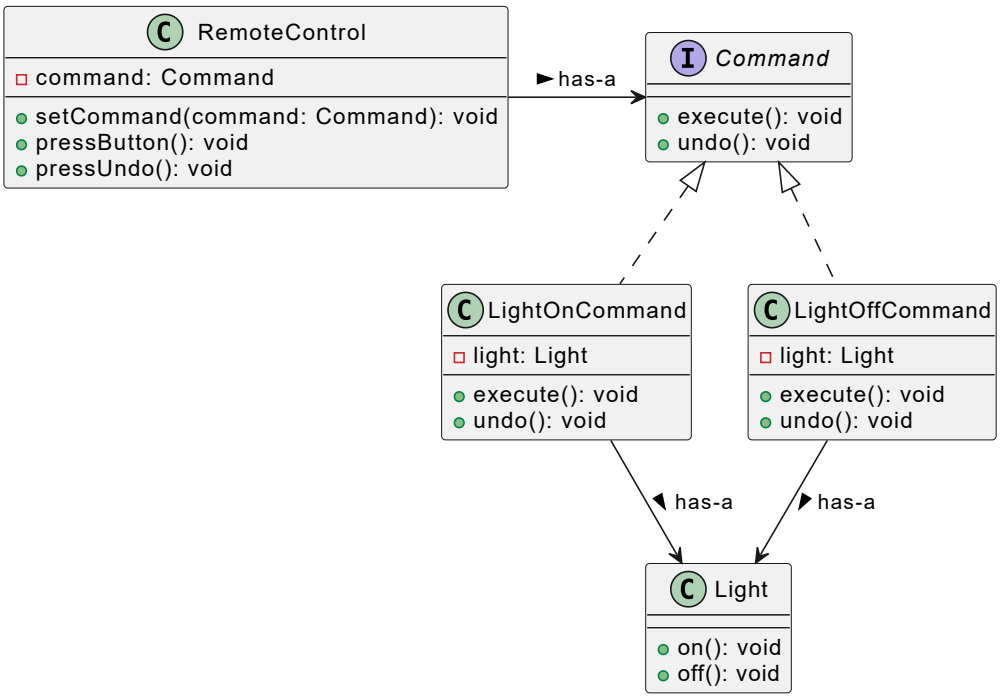
6. Interpreter Pattern

Interpreter Pattern defines a context to interpret or evaluate an expression.



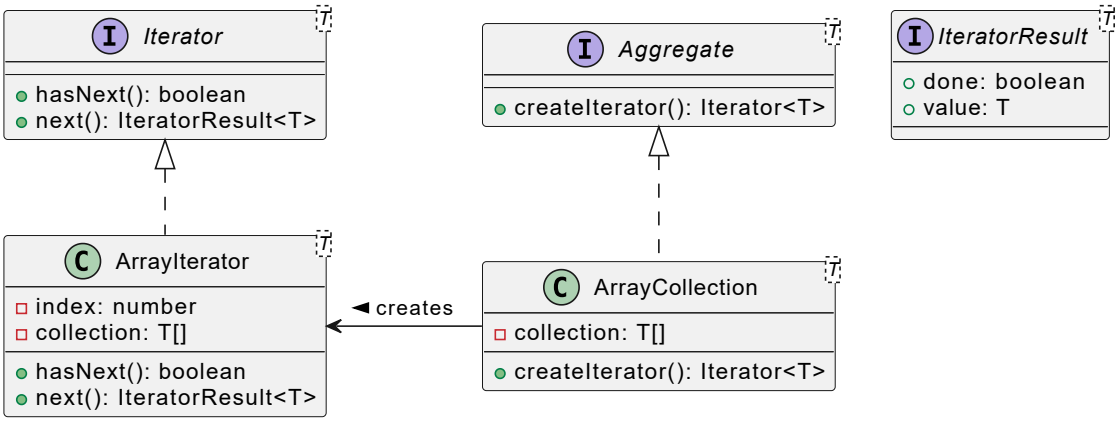
7. Command Pattern

Command Pattern turns request commands into objects, allowing us to either parameterize or queue them. This helps in decoupling the request sender and the receiver.



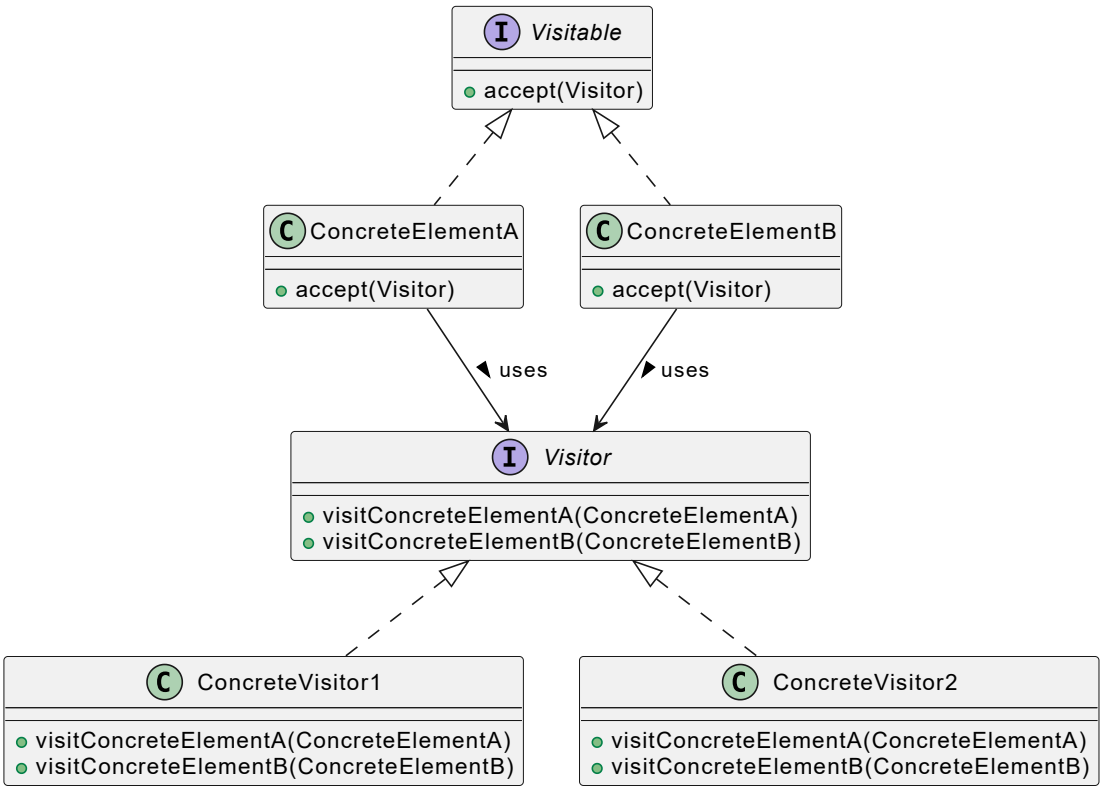
8. Iterator Pattern

Iterator Pattern provides a way to access elements of a collection sequentially without exposing the underlying representation of the collection.



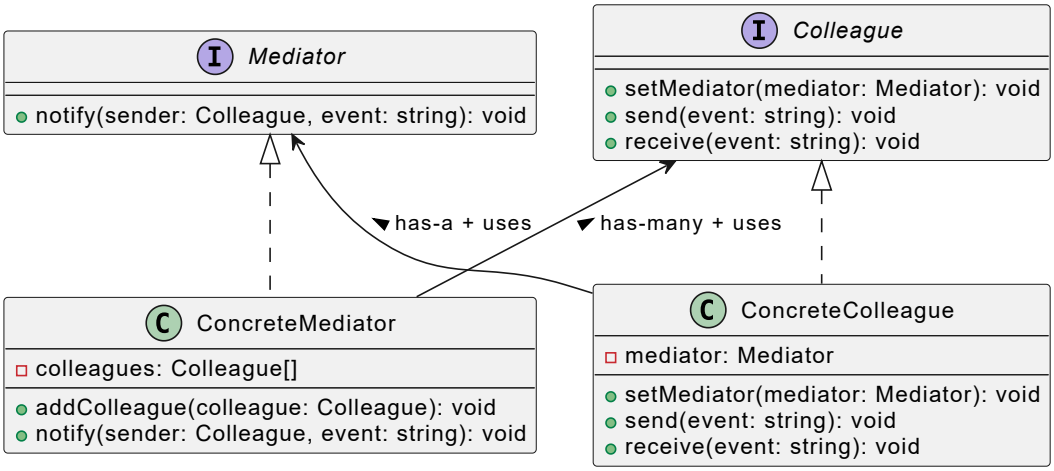
9. Visitor Pattern

Visitor Pattern allows adding operations to existing classes without changing them, encouraging the open/close principle of SOLID.



10. Mediator Pattern

Mediator Pattern encourages loose coupling by keeping two objects from referencing each other through a mediator object.



11. Memento Pattern

Memento Pattern provides the ability to revert an object to its previous state.