Q. What is the role of Behavioral patterns in software design? Explain with some behavioral patterns.

Role of behavioral patterns in software design :

Behavioral design patterns are concerned with algorithms and the assignment of responsibilities between objects.

Some behavioral patterns :

- Chain of Responsibility
  Lets you pass requests along a chain of handlers. Upon receiving a request, each handler decides either to process the request or to pass it the next handler in the chain.
- Command

  Turns a request into a stand-alone object that contains all information about the request. This transformation lets you pass requests as a method arguments, delay or queue a request's execution, and support undoable operations.

- Iterator
  Lets you traverse elements of a collection without exposing its underlying representation (list, stack, tree, etc.).
- Memento
  Lets you save and restore the previous state of an object without revealing the details of its implementation.
- State
  Lets an object alter its behavior when its internal state changes. It appears as if the object changed its class.
- Visitor
  Lets you separate algorithms from the objects on which they operate.

Q. Write about the intent, motivation, structure and applicability of Chain of responsibility pattern.

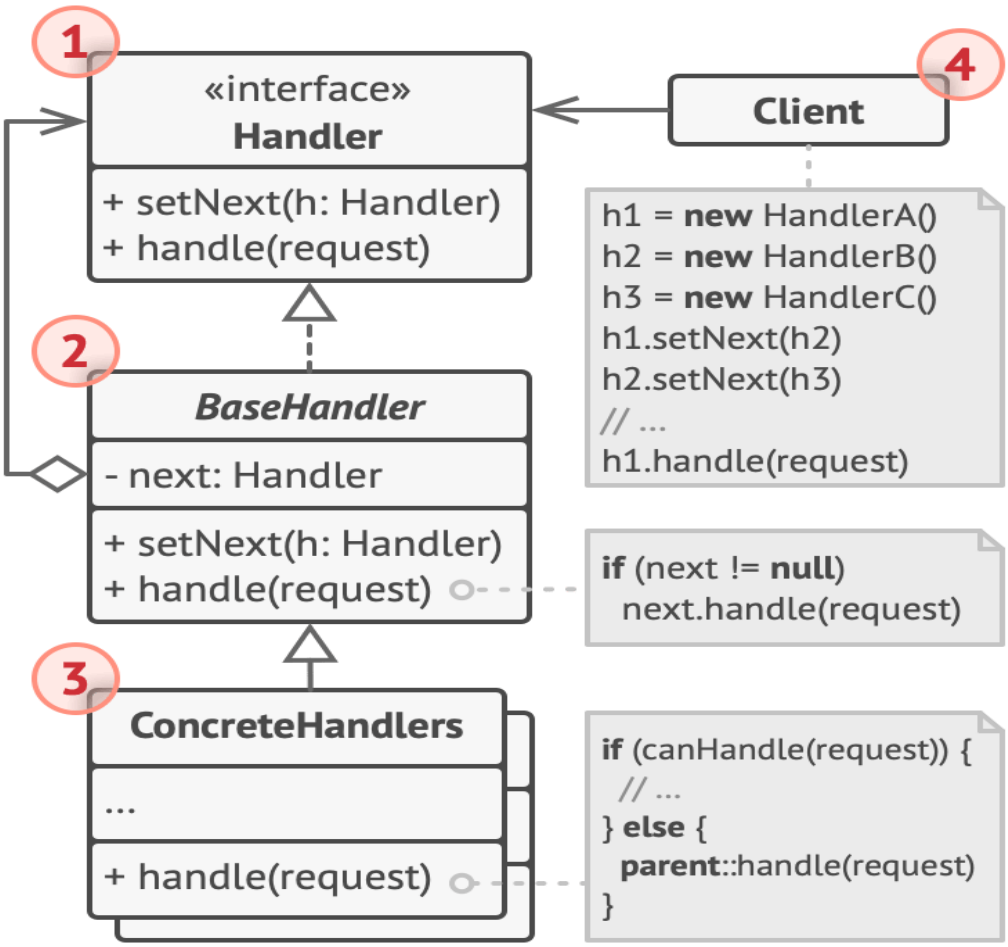
Intent

Chain of Responsibility is a behavioral design pattern that lets you pass requests along a chain of handlers. Upon receiving a request, each handler decides either to process the request or to pass it to the next handler in the chain.

Motivation

Chain of Responsibility relies on transforming particular behaviors into stand-alone objects called handlers. In our case, each check should be extracted to its own class with a single method that performs the check. The request, along with its data, is passed to this method as an argument.

Structure



Participants

The Handler declares the interface, common for all concrete handlers. It usually contains just a single method for handling requests, but sometimes it may also have another method for setting the next handler on the chain.

The Base Handler is an optional class where you can put the boilerplate code that's common to all handler classes.

Concrete Handlers contain the actual code for processing requests. Upon receiving a request, each handler must decide whether to process it and, additionally, whether to pass it along the chain.

The Client may compose chains just once or compose them dynamically, depending on the application's logic.

Applicability

- Use the Chain of Responsibility pattern when your program is expected to process different kinds of requests in various ways, but the exact types of requests and their sequences are unknown beforehand.
- Use the pattern when it's essential to execute several handlers in a particular order.

- Use the CoR pattern when the set of handlers and their order are supposed to change at runtime.

Q. Write about the intent, motivation, structure and applicability of Command pattern.

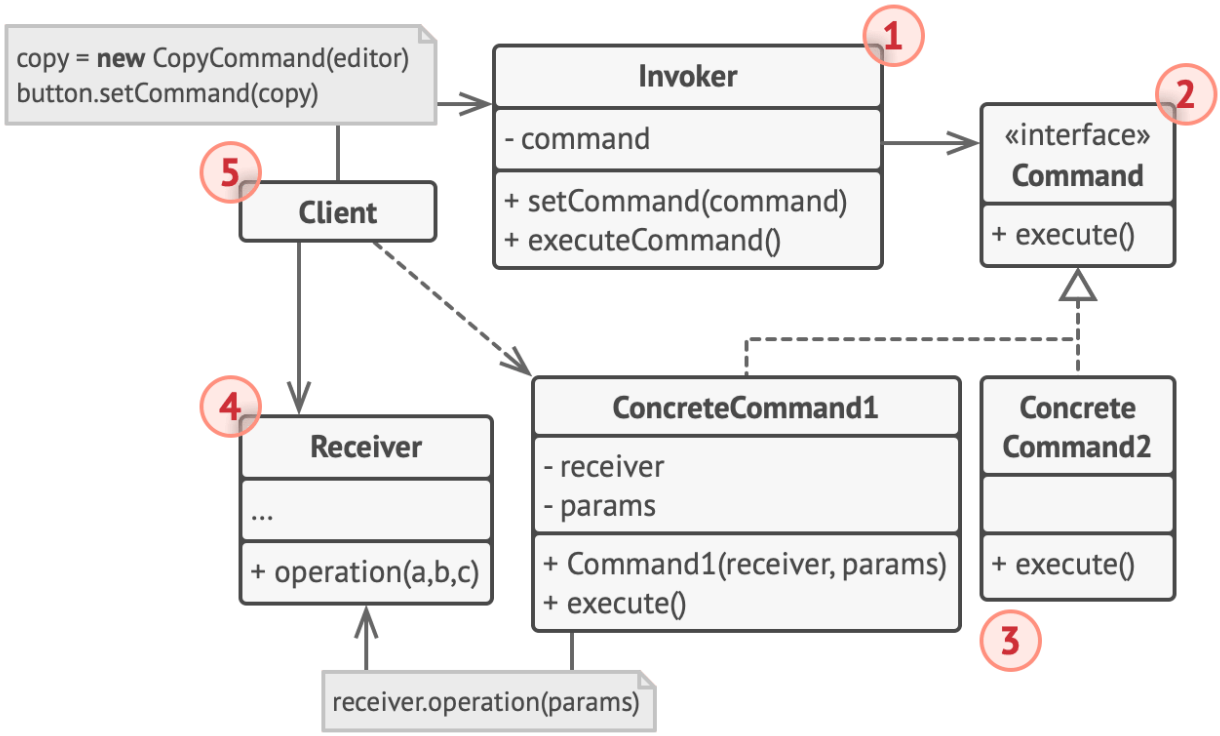
Intent

Command is a behavioral design pattern that turns a request into a stand-alone object that contains all information about the request. This transformation lets you pass requests as a method arguments, delay or queue a request's execution, and support undoable operations.

Motivation

The Command pattern suggests that GUI objects shouldn't send these requests directly. Instead, you should extract all of the request details, such as the object being called, the name of the method and the list of arguments into a separate command class with a single method that triggers this request.

Structure



Participants

The Sender class (aka invoker) is responsible for initiating requests. This class must have a field for storing a reference to a command object. The sender triggers that command instead of sending the request directly to the receiver.

The Command interface usually declares just a single method for executing the command.

Concrete Commands implement various kinds of requests. A concrete command isn't supposed to perform the work on its own, but rather to pass the call to one of the business logic objects.

The Client creates and configures concrete command objects. The client must pass all of the request parameters, including a receiver instance, into the command's constructor.

Applicability

- Use the Command pattern when you want to parametrize objects with operations.
- Use the Command pattern when you want to queue operations, schedule their execution, or execute them remotely.
- Use the Command pattern when you want to implement reversible operations.

Q. Write about the intent, motivation, structure and applicability of Iterator pattern.

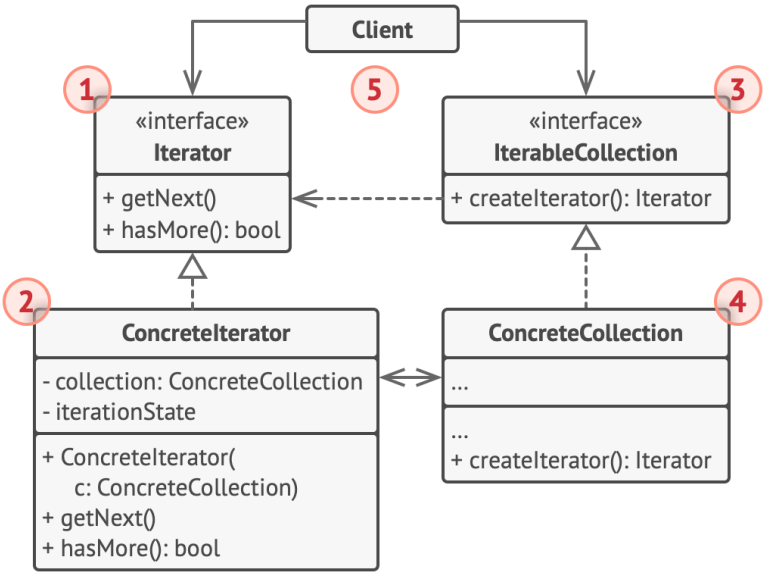
Intent

Iterator is a behavioral design pattern that lets you traverse elements of a collection without exposing its underlying representation (list, stack, tree, etc.).

Motivation

The main idea of the Iterator pattern is to extract the traversal behavior of a collection into a separate object called an iterator.

Structure



Participants

The Iterator interface declares the operations required for traversing a collection: fetching the next element, retrieving the current position, restarting iteration, etc.

Concrete Iterators implement specific algorithms for traversing a collection. The iterator object should track the traversal progress on its own. This allows several iterators to traverse the same collection independently of each other.

The Collection interface declares one or multiple methods for getting iterators compatible with the collection. Note that the return type of the methods must be declared as the iterator interface so that the concrete collections can return various kinds of iterators.

Concrete Collections return new instances of a particular concrete iterator class each time the client requests one.

The Client works with both collections and iterators via their interfaces. This way the client isn't coupled to concrete classes, allowing you to use various collections and iterators with the same client code.

Applicability

- Use the Iterator pattern when your collection has a complex data structure under the hood, but you want to hide its complexity from clients (either for convenience or security reasons).
- Use the Iterator when you want your code to be able to traverse different data structures or when types of these structures are unknown beforehand
- Use the pattern to reduce duplication of the traversal code across your app.

Q. Write about the intent, motivation, structure and applicability of Momento pattern.

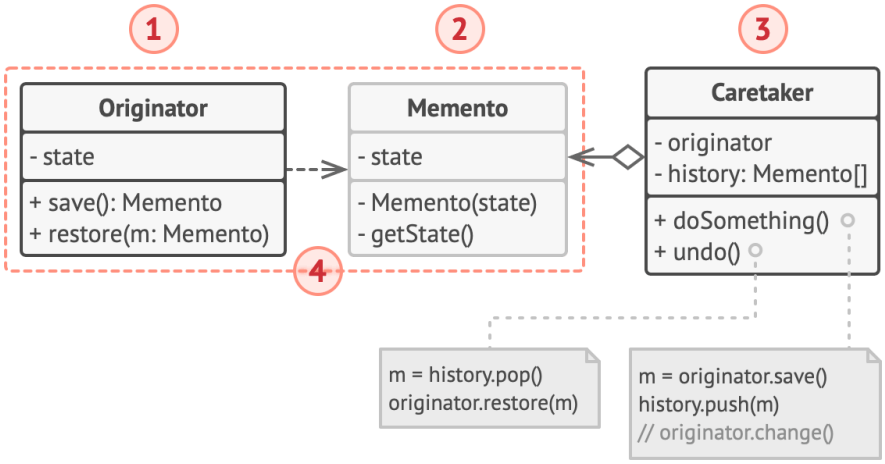
Intent

Memento is a behavioral design pattern that lets you save and restore the previous state of an object without revealing the details of its implementation.

Motivation

The Memento pattern delegates creating the state snapshots to the actual owner of that state, the originator object. Hence, instead of other objects trying to copy the editor's state from the "outside," the editor class itself can make the snapshot since it has full access to its own state.

Structure



Participants

The Originator class can produce snapshots of its own state, as well as restore its state from snapshots when needed.

The Memento is a value object that acts as a snapshot of the originator's state. It's a common practice to make the memento immutable and pass it the data only once, via the constructor.

The Caretaker knows not only "when" and "why" to capture the originator's state, but also when the state should be restored.

Applicability

- Use the Memento pattern when you want to produce snapshots of the object's state to be able to restore a previous state of the object.
- Use the pattern when direct access to the object's fields/getters/setters violates its encapsulation.

Q. Write about the intent, motivation, structure and applicability of State pattern.

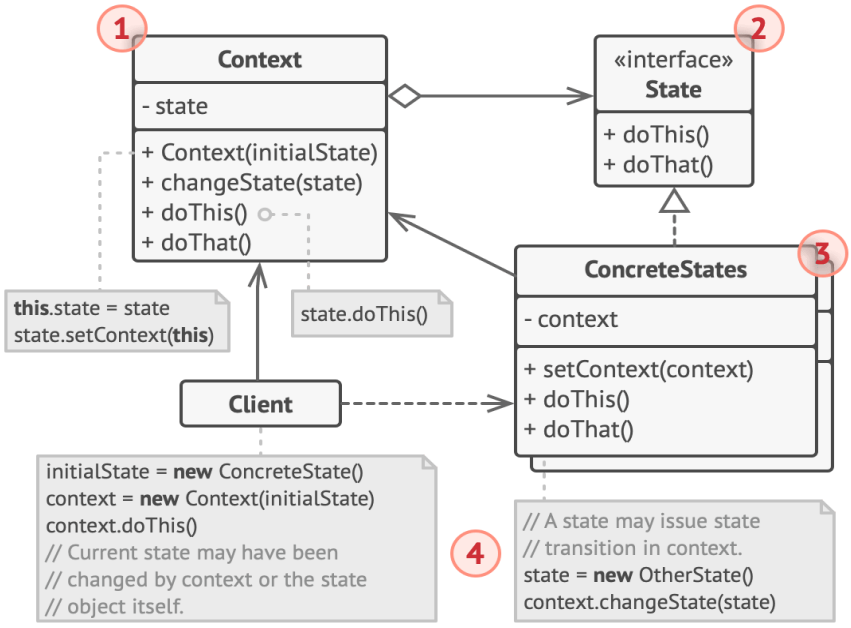
Intent

State is a behavioral design pattern that lets an object alter its behavior when its internal state changes. It appears as if the object changed its class.

Motivation

The State pattern suggests that you create new classes for all possible states of an object and extract all state-specific behaviors into these classes.

Structure



Participants

Context stores a reference to one of the concrete state objects and delegates to it all state-specific work. The context communicates with the state object via the state interface. The context exposes a setter for passing it a new state object.

The State interface declares the state-specific methods. These methods should make sense for all concrete states because you don't want some of your states to have useless methods that will never be called.

Concrete States provide their own implementations for the state-specific methods. To avoid duplication of similar code across multiple states, you may provide intermediate abstract classes that encapsulate some common behavior.

Applicability

- Use the State pattern when you have an object that behaves differently depending on its current state, the number of states is enormous, and the state-specific code changes frequently.
- Use the pattern when you have a class polluted with massive conditionals that alter how the class behaves according to the current values of the class's fields.
- Use State when you have a lot of duplicate code across similar states and transitions of a condition-based state machine.

Q. Write about the intent, motivation, structure and applicability of Visitor pattern.

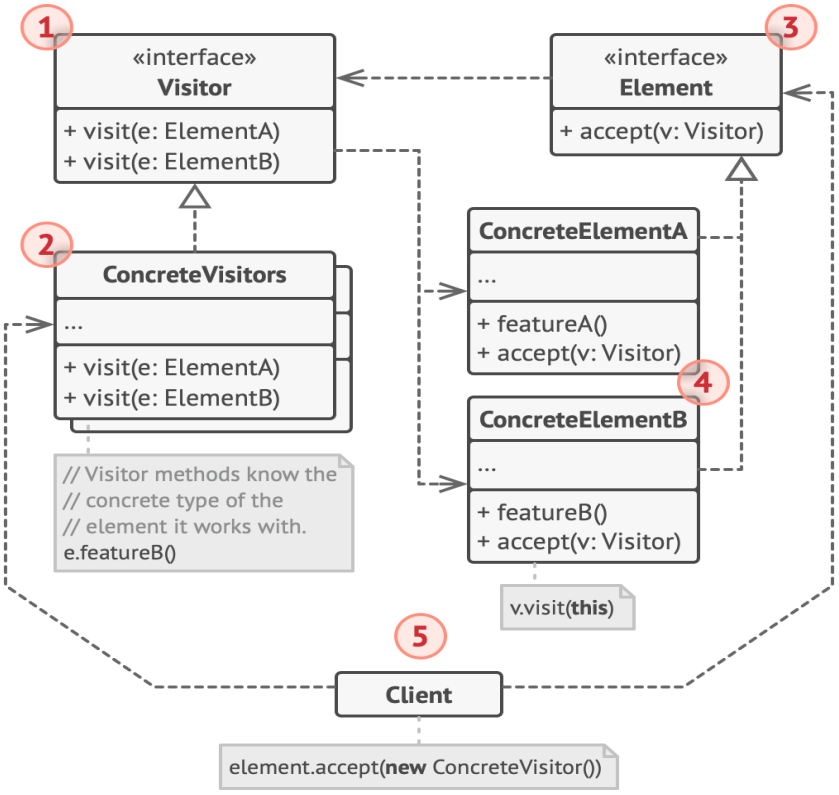
Intent

Visitor is a behavioral design pattern that lets you separate algorithms from the objects on which they operate.

Motivation

The Visitor pattern suggests that you place the new behavior into a separate class called visitor, instead of trying to integrate it into existing classes. The original object that had to perform the behavior is now passed to one of the visitor's methods as an argument, providing the method access to all necessary data contained within the object.

Structure



Participants

The Visitor interface declares a set of visiting methods that can take concrete elements of an object structure as arguments. These methods may have the same names if the program is written in a language that supports overloading, but the type of their parameters must be different.

Each Concrete Visitor implements several versions of the same behaviors, tailored for different concrete element classes.

The Element interface declares a method for "accepting" visitors. This method should have one parameter declared with the type of the visitor interface.

Each Concrete Element must implement the acceptance method. The purpose of this method is to redirect the call to the proper visitor's method corresponding to the current element class

The Client usually represents a collection or some other complex object (for example, a Composite tree)

Applicability

- Use the Visitor when you need to perform an operation on all elements of a complex object structure (for example, an object tree)
- Use the Visitor to clean up the business logic of auxiliary behaviors.
- Use the pattern when a behavior makes sense only in some classes of a class hierarchy, but not in others.