

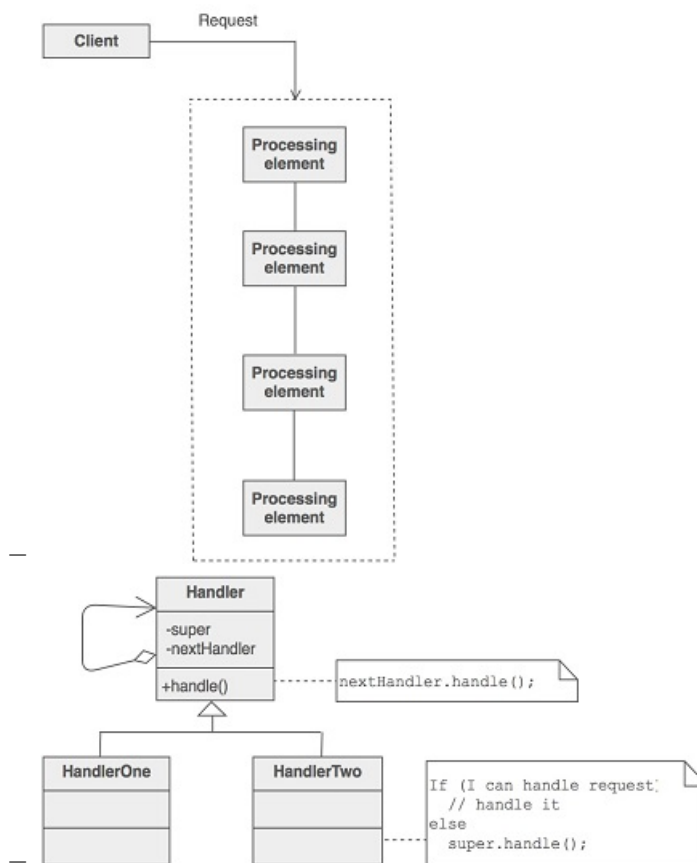
Behavioral

General:

Identify common communication patterns between objects and realize these patterns

Chain of Responsibility:

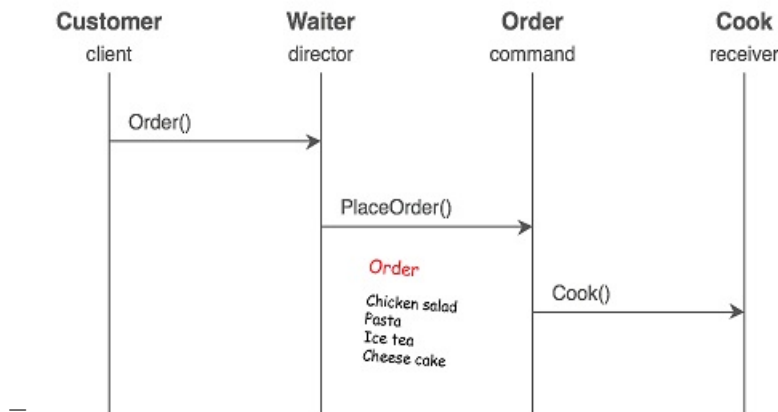
- Definition/Use
 - command objects are handled or passed on to other objects by logic-containing processing objects
 - avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request
- Structure



- Notes
 - base class maintains a next pointer
 - if the request needs to be "passed on", then the derived class "calls back" to the base class, which delegates to the "next" pointer

Command:

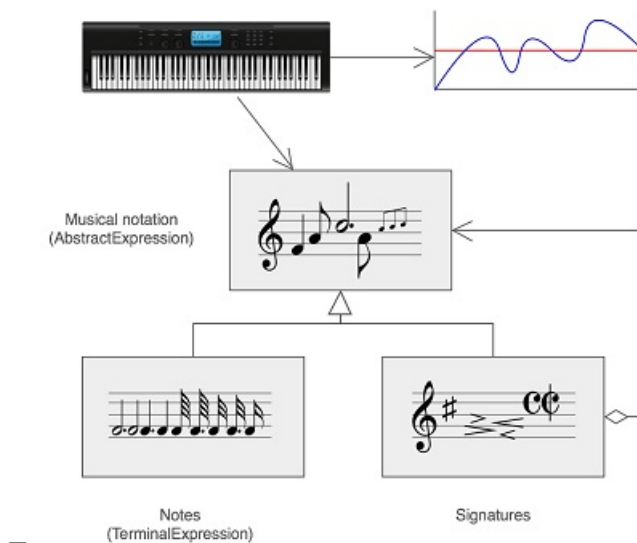
- Definition/Use
 - command objects encapsulate an action and its parameters
 - Need to issue requests to objects without knowing anything about the operation being requested or the receiver of the request
 - separation provides flexibility in the timing and sequencing of commands
 - command objects can be thought of as "tokens", created by one client that knows what need to be done, passed to another client that has the resources for doing it
- Structure



- Notes
 - define a Command interface with a method signature like `execute()`

Interpreter:

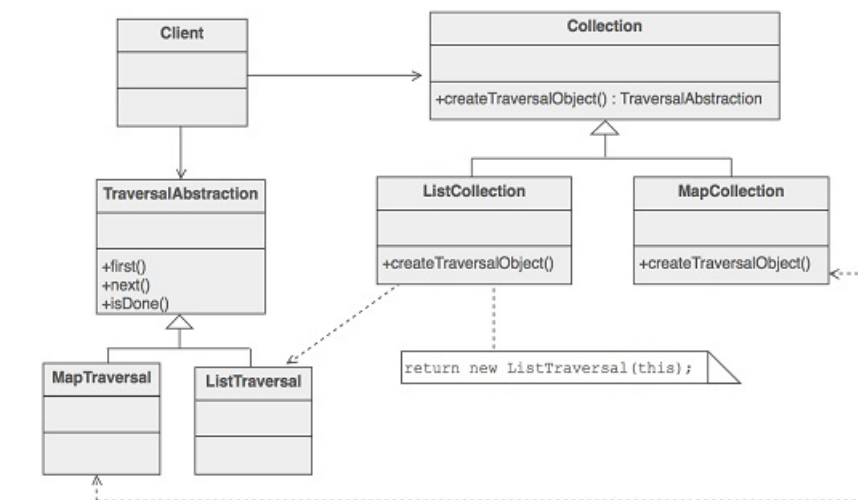
- Definition/Use
 - implement a specialized computer language to rapidly solve a specific set of problems
 - map a domain to a language, the language to a grammar, and the grammar to a hierarchical object
 - oriented design
- Structure



- Notes
 - the pattern doesn't address parsing. When the grammar is very complex, other techniques (such as a parser) are more appropriate

Iterator:

- Definition/Use
 - iterators are used to access the elements of an aggregate object sequentially without exposing its underlying representation
 - need to "abstract" the traversal of wildly different data structures so that algorithms can be defined that are capable of interfacing with each transparently
- Structure



- Notes

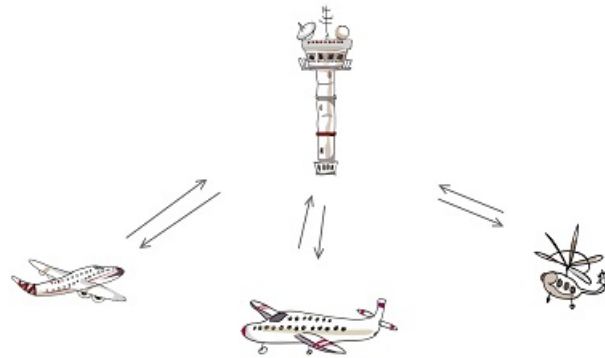
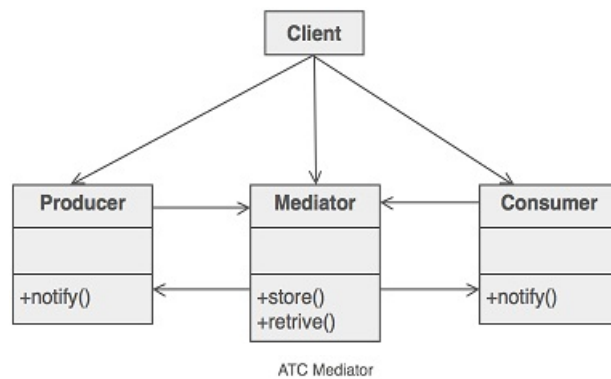
- clients use the `first()`, `is_done()`, `next()`, and `current_item()` protocol to access the elements of the collection `class`

Mediator:

- Definition/Use

- provides a unified interface to a set of interfaces in a subsystem
 - promotes loose coupling by keeping objects from referring to each other explicitly

- Structure



- Notes

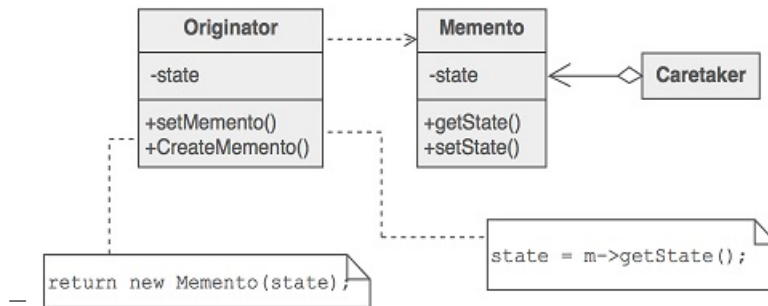
- be careful not to create a "controller" or "god" object

Memento:

- Definition/Use

- provides the ability to restore an object to its previous state (rollback)
- pattern defines three distinct roles
 - * originator : the object that knows how to save itself
 - * caretaker : the object that knows why and when the originator needs to save and restore itself
 - * memento : the lock box that is written and read by the originator, and shepherded by the caretaker

- Structure



- Notes

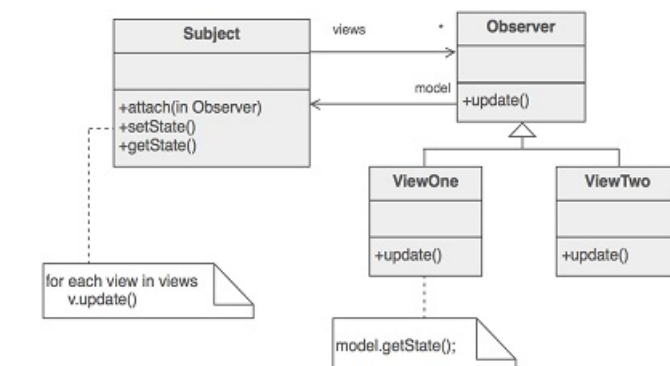
- identify the roles of “caretaker” and “originator”

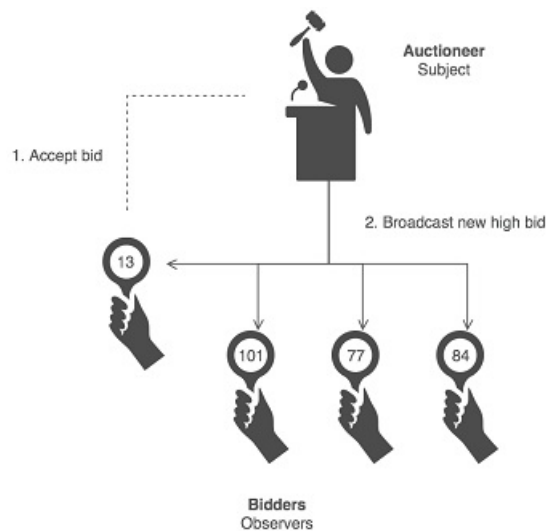
Observer(Publish/Subscribe or Event Listener):

- Definition/Use

- objects register to observe an event that may be raised by another object
- defines a one
- to
- many relationship so that when one object changes state, the others are notified and updated automatically

- Structure

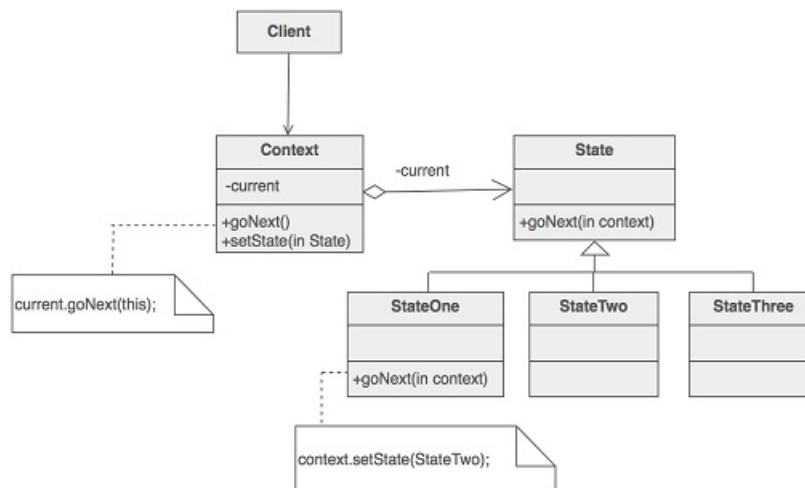


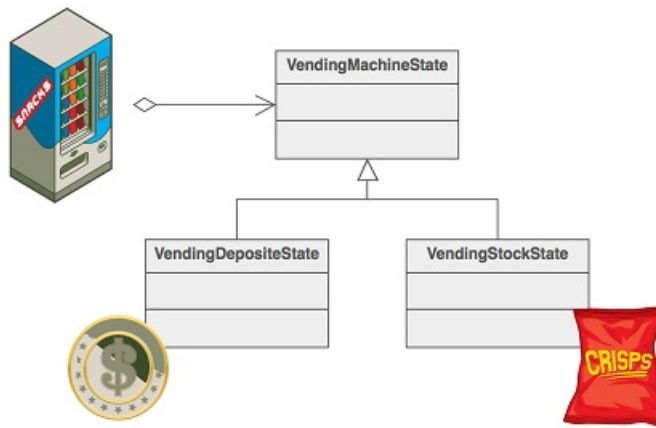


- Notes
 - subject broadcasts events to all registered observers

State:

- Definition/Use
 - a clean way for an object to partially change its type at runtime
 - a monolithic object's behavior is a function of its state
- Structure





- Notes

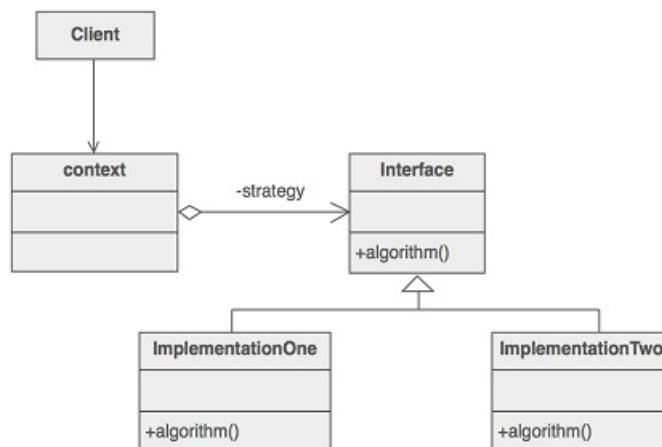
- pattern does not specify where the state transitions will be defined
 - * the "context" object
 - * each individual State derived class
 - advantage is ease of adding new State derived classes
 - disadvantage is each State derived class has knowledge of (coupling to) its siblings, which introduces dependencies between subclasses

Strategy:

- Definition/Use

- algorithms can be selected on the fly
- defines a set of algorithms that can be used interchangeably

- Structure



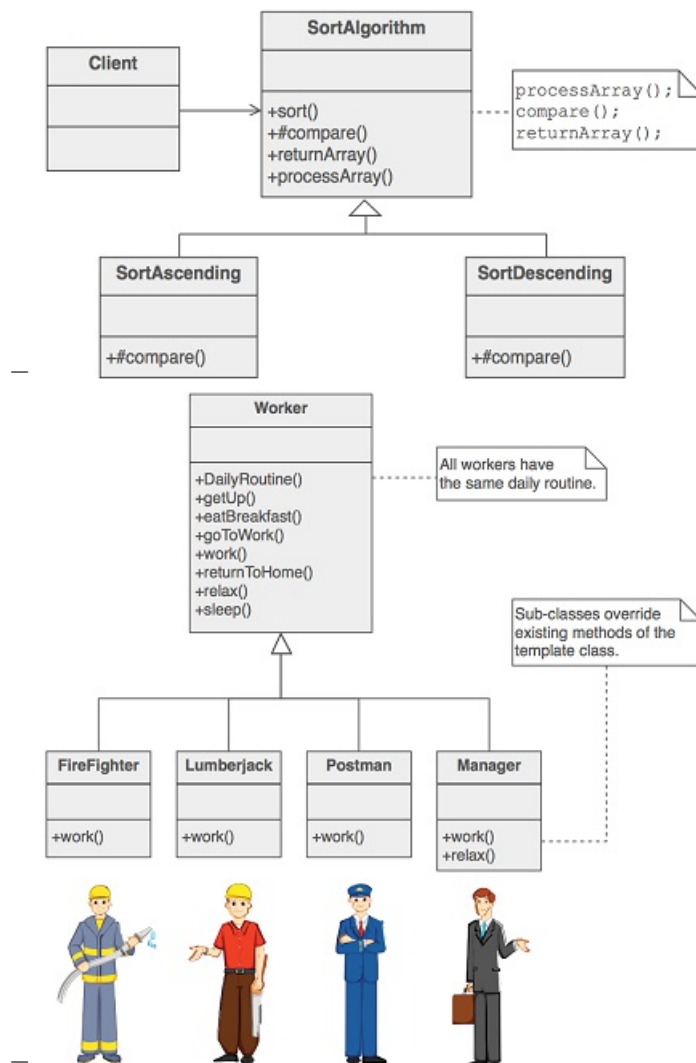
- Notes

- identify an algorithm (i.e. a behavior) that the client would prefer to access through a "flex point"

Template method:

- Definition/Use
 - describes the program skeleton of a program
 - component designer decides which steps of an algorithm are invariant (or standard), and which are variant (or customizable)

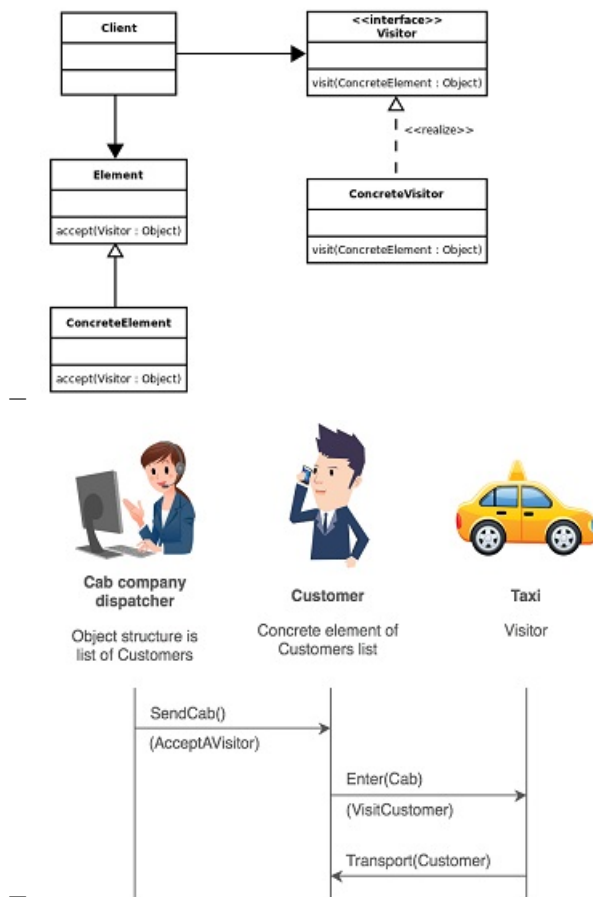
- Structure



- Notes
 - examine the algorithm, and decide which steps are standard and which steps are peculiar to each of the current classes

Visitor:

- Definition/Use
 - a way to separate an algorithm from an object
- Structure



- Example

```

public interface CharacterVisitor {

    public void visit(char aChar);

}

public class MyString {

    // ... other methods, fields

    // Our main implementation of the visitor pattern
    public void foreach(CharacterVisitor aVisitor) {
  
```

```
        int length = this.length();
        // Loop over all the characters in the string
        for (int i = 0; i < length; i++) {
            // Get the current character, and let the visitor visit it.
            aVisitor.visit(this.getCharAt(i));
        }
    }

    // ... other methods, fields

} // end class MyString

public class MyStringPrinter implements CharacterVisitor {

    // We have to implement this method because we're implementing the
    // CharacterVisitor
    // interface
    public void visit(char aChar) {
        // All we're going to do is print the current character to the standard
        // output
        System.out.print(aChar);
    }

    // This is the method you call when you want to print a string
    public void print(MyString aStr) {
        // we'll let the string determine how to get each character, and
        // we already defined what to do with each character in our
        // visit method.
        aStr.foreach(this);
    }

} // end class MyStringPrinter
```

- Notes

- if you have and will always have only one visitor, you'd rather implement the composite pattern

Comparison:

- chain of Responsibility, command, mediator, and observer, address how you can decouple senders and receivers, but with different trade
- offs
 - chain of Responsibility passes a sender request along a chain of potential receivers
- command and memento act as magic tokens to be passed around and invoked at a later time
 - in command, the token represents a request

- in memento, it represents the internal state of an object at a particular time
- polymorphism is important to command, but not to memento because its interface is so narrow that a memento can only be passed as a value