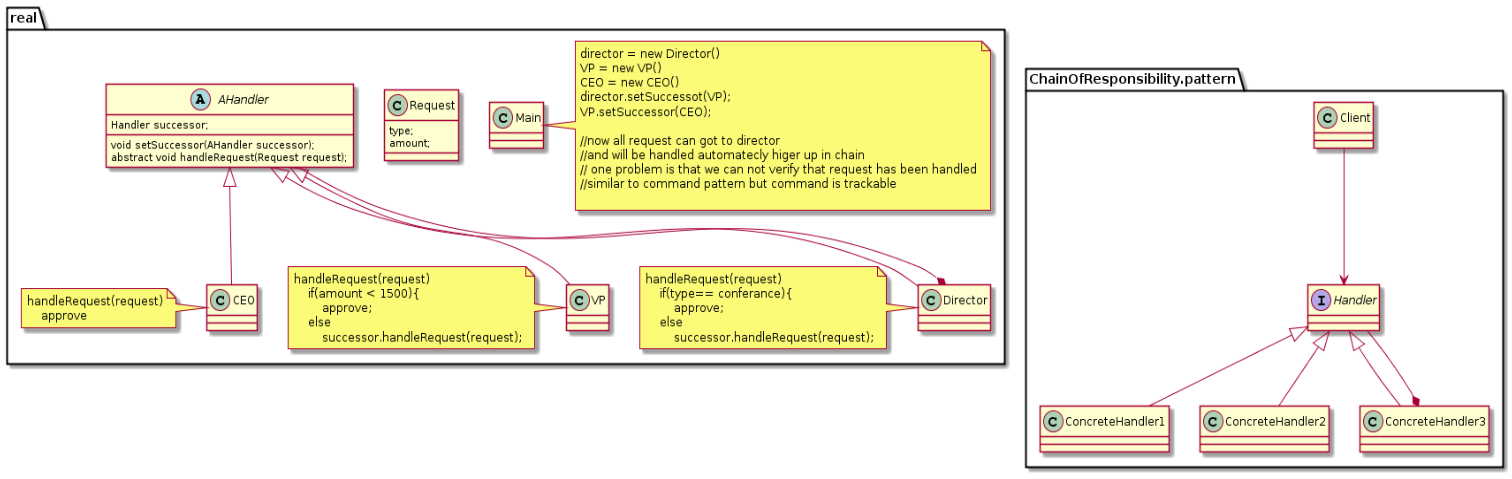
VisitorPattern



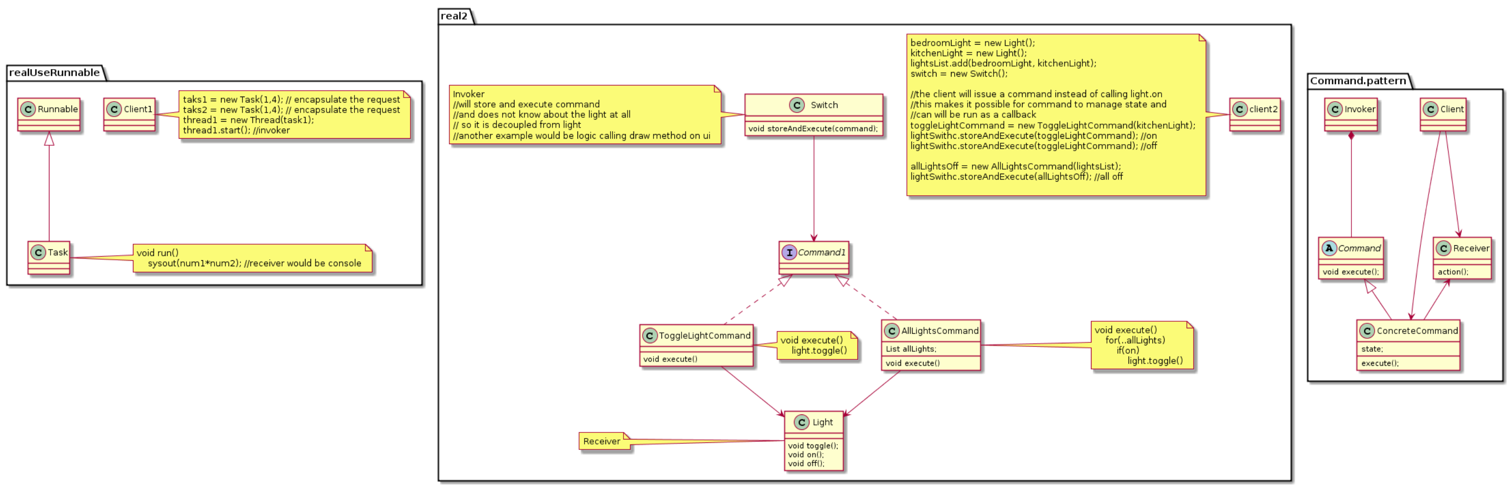
```
director = new Director()
VP = new VP()
CEO = new CEO()
director.setSuccessot(VP);
VP.setSuccessor(CEO);

//now all request can got to director
//and will be handled automatecly higer up in chain
// one problem is that we can not verify that request has been handled
//similar to command pattern but command is trackable
```

```
handleRequest(request)
    approve
```

```
handleRequest(request)
    if(amount < 1500){
        approve;
    else
        successor.handleRequest(request);
```

```
handleRequest(request)
    if(type== conferance){
        approve;
    else
        successor.handleRequest(request);
```

Command Pattern



```
taks1 = new Task(1,4); // encapsulate the request
taks2 = new Task(1,4); // encapsulate the request
thread1 = new Thread(task1);
thread1.start(); //invoker
```

```
void run()
    sysout(num1*num2); //receiver would be console
```

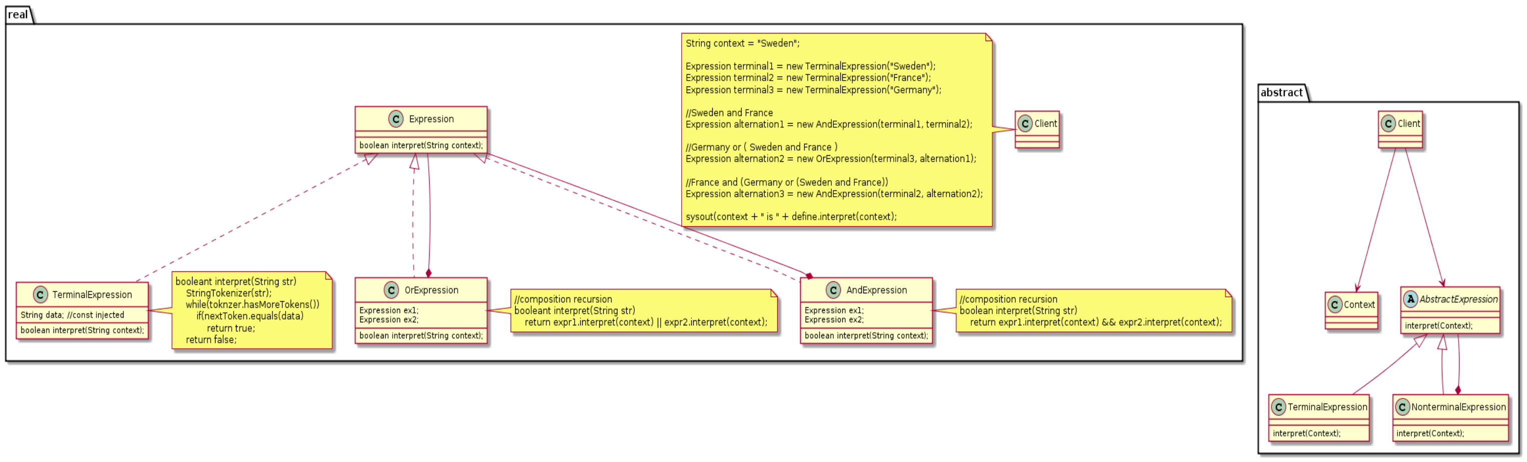
```
Invoker
//will store and execute command
//and does not know about the light at all
// so it is decoupled from light
//another example would be logic calling draw method on ui
```

```
bedroomLight = new Light();
kitchenLight = new Light();
lightsList.add(bedroomLight, kitchenLight);
switch = new Switch();

//the client will issue a command instead of calling light.on
//this makes it possible for command to manage state and
//can will be run as a callback
toggleLightCommand = new ToggleLightCommand(kitchenLight);
lightSwithc.storeAndExecute(toggleLightCommand); //on
lightSwithc.storeAndExecute(toggleLightCommand); //off

allLightsOff = new AllLightsCommand(lightsList);
lightSwithc.storeAndExecute(allLightsOff); //all off
```

```
void execute()
    light.toggle()
```

```
void execute()
    for(..allLights)
        if(on)
            light.toggle()
```

Interpreter Pattern



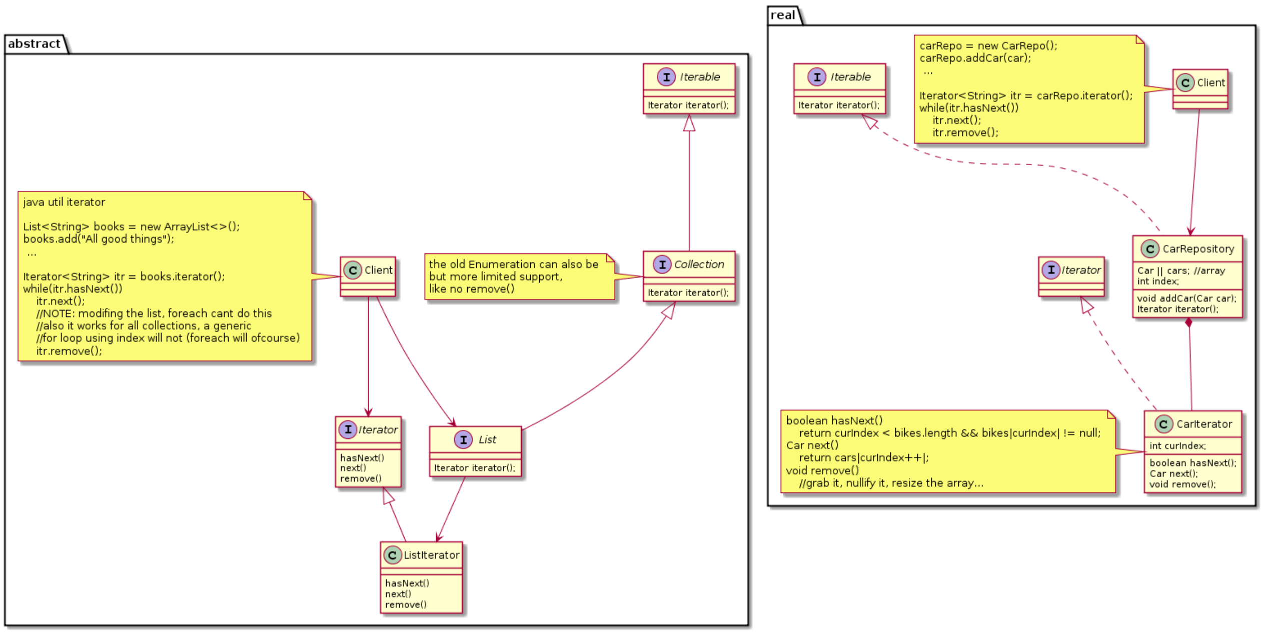
```
String context = "Sweden";

Expression terminal1 = new TerminalExpression("Sweden");
Expression terminal2 = new TerminalExpression("France");
Expression terminal3 = new TerminalExpression("Germany");

//Sweden and France
Expression alternation1 = new AndExpression(terminal1, terminal2);

//Germany or ( Sweden and France )
Expression alternation2 = new OrExpression(terminal3, alternation1);

//France and (Germany or (Sweden and France))
Expression alternation3 = new AndExpression(terminal2, alternation2);

sysout(context + " is " + define.interpret(context);
```

```
booleant interpret(String str)
    StringTokenizer(str);
    while(toknzer.hasMoreTokens())
        if(nextToken.equals(data)
            return true;
    return false;
```

```
//composition recursion
booleant interpret(String str)
    return expr1.interpret(context) || expr2.interpret(context);
```

```
//composition recursion
boolean interpret(String str)
    return expr1.interpret(context) && expr2.interpret(context);
```

Iterator Pattern



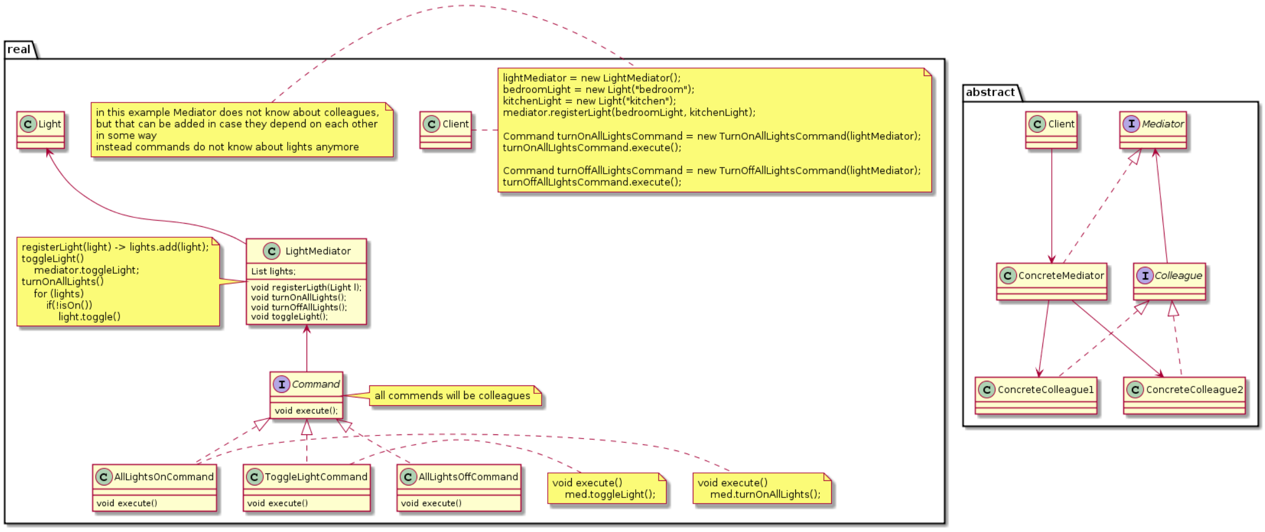
```
java util iterator

List<String> books = new ArrayList<>();
books.add("All good things");
 ...

Iterator<String> itr = books.iterator();
while(itr.hasNext())
    itr.next();
    //NOTE: modifing the list, foreach cant do this
    //also it works for all collections, a generic
    //for loop using index will not (foreach will ofcourse)
    itr.remove();
```

```
the old Enumeration can also be
but more limited support,
like no remove()
```

```
carRepo = new CarRepo();
carRepo.addCar(car);
 ...

Iterator<String> itr = carRepo.iterator();
while(itr.hasNext())
    itr.next();
    itr.remove();
```

```
boolean hasNext()
    return curIndex < bikes.length && bikes[curIndex] != null;
Car next()
    return cars[curIndex++];
void remove()
    //grab it, nullify it, resize the array...
```

Mediator Pattern



```
in this example Mediator does not know about colleagues,
but that can be added in case they depend on each other
in some way
instead commands do not know about lights anymore
```

```
lightMediator = new LightMediator();
bedroomLight = new Light("bedroom");
kitchenLight = new Light("kitchen");
mediator.registerLight(bedroomLight, kitchenLight);

Command turnOnAllLightsCommand = new TurnOnAllLightsCommand(lightMediator);
turnOnAllLightsCommand.execute();

Command turnOffAllLightsCommand = new TurnOffAllLightsCommand(lightMediator);
turnOffAllLightsCommand.execute();
```

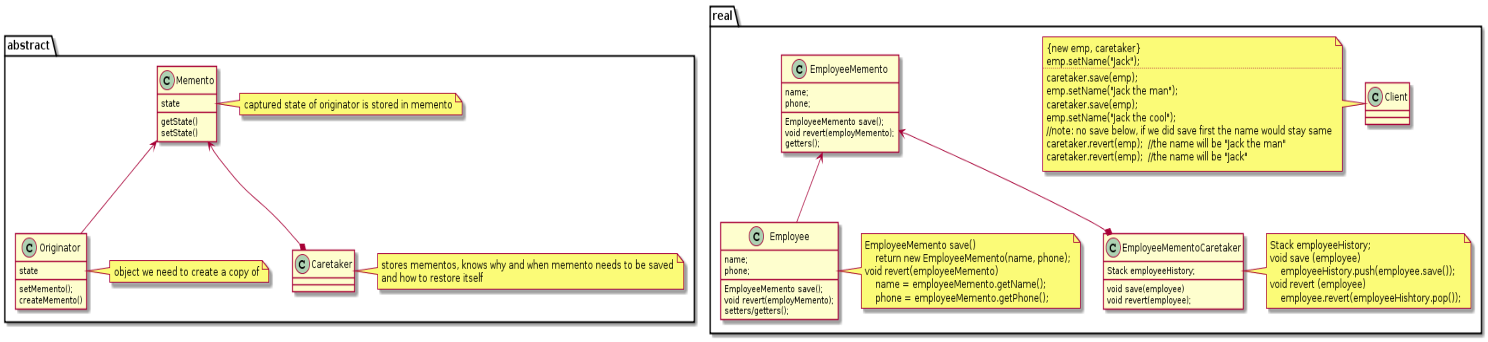
```
registerLight(light) -> lights.add(light);
toggleLight()
    mediator.toggleLight;
turnOnAllLights()
    for (lights)
        if(!isOn())
            light.toggle()
```

all commends will be colleagues

```
void execute()
    med.toggleLight();
```

```
void execute()
    med.turnOnAllLights();
```

Memento Pattern

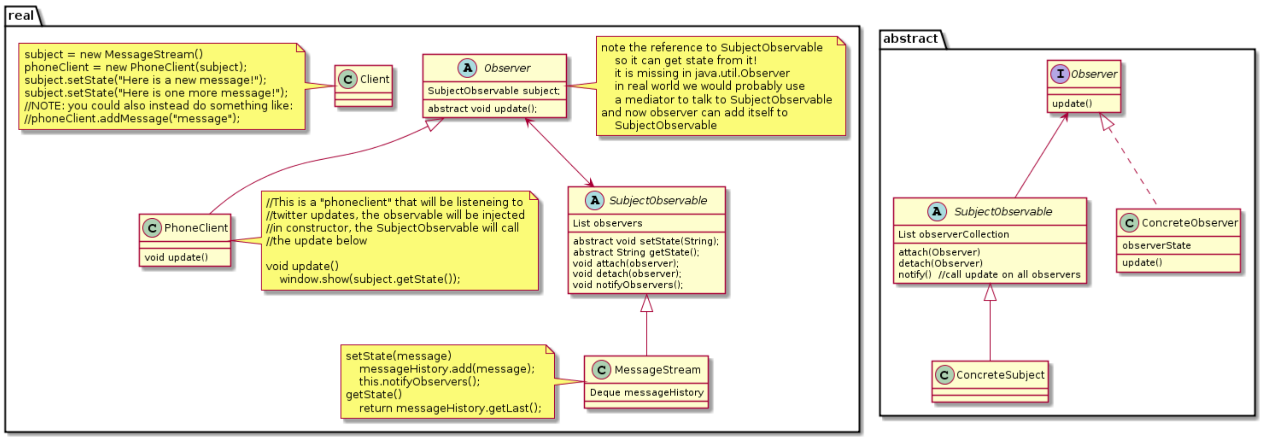


```
{new emp, caretaker}
emp.setName("Jack");
caretaker.save(emp);
emp.setName("Jack the man");
caretaker.save(emp);
emp.setName("Jack the cool");
//note: no save below, if we did save first the name would stay same
caretaker.revert(emp);  //the name will be "Jack the man"
caretaker.revert(emp);  //the name will be "Jack"
```

```
EmployeeMemento save()
    return new EmployeeMemento(name, phone);
void revert(employeeMemento)
    name = employeeMemento.getName();
    phone = employeeMemento.getPhone();
```

```
Stack employeeHistory;
void save (employee)
    employeeHistory.push(employee.save());
void revert (employee)
    employee.revert(employeeHishtory.pop());
```

Observer pattern



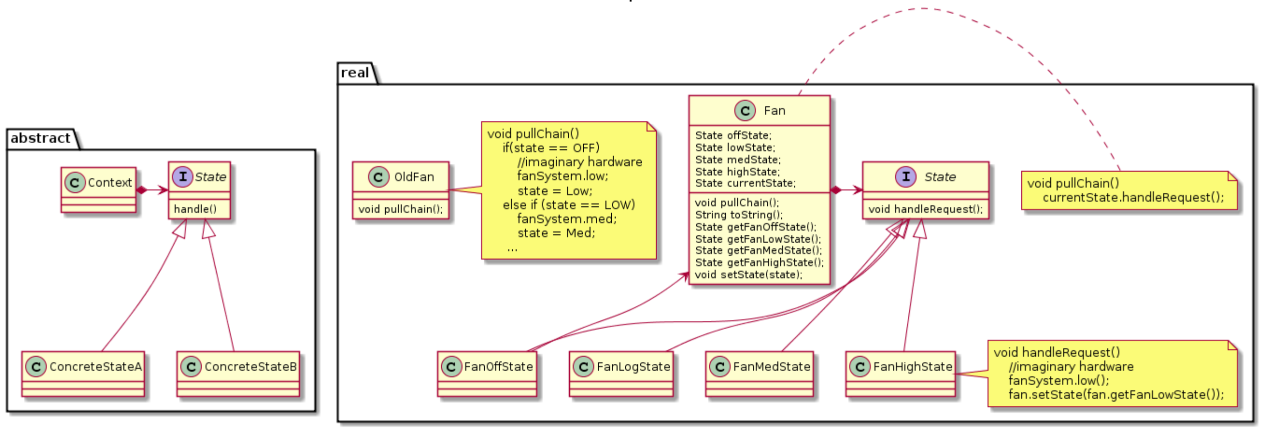
```
subject = new MessageStream()
phoneClient = new PhoneClient(subject);
subject.setState("Here is a new message!");
subject.setState("Here is one more message!");
//NOTE: you could also instead do something like:
//phoneClient.addMessage("message");
```

```
note the reference to SubjectObservable
    so it can get state from it!
    it is missing in java.util.Observer
    in real world we would probably use
    a mediator to talk to SubjectObservable
and now observer can add itself to
    SubjectObservable
```

```
//This is a "phoneclient" that will be listeneing to
//twitter updates, the observable will be injected
//in constructor, the SubjectObservable will call
//the update below

void update()
    window.show(subject.getState());
```

```
setState(message)
    messageHistory.add(message);
    this.notifyObservers();
getState()
    return messageHistory.getLast();
```

state pattern

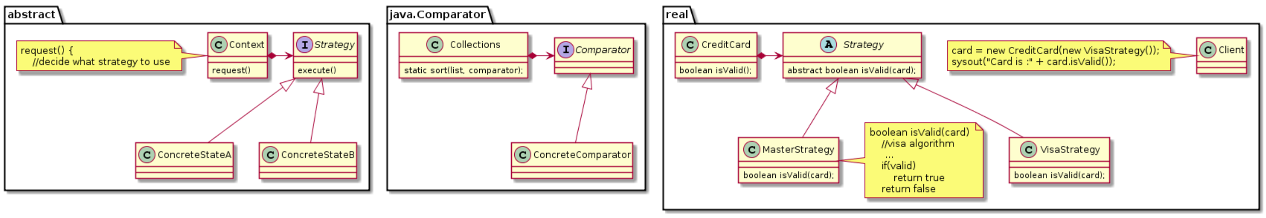


```
void pullChain()
    if(state == OFF)
        //imaginary hardware
        fanSystem.low;
        state = Low;
    else if (state == LOW)
        fanSystem.med;
        state = Med;
    ...
```

```
void pullChain()
    currentState.handleRequest();
```

```
void handleRequest()
    //imaginary hardware
    fanSystem.low();
    fan.setState(fan.getFanLowState());
```

strategy pattern

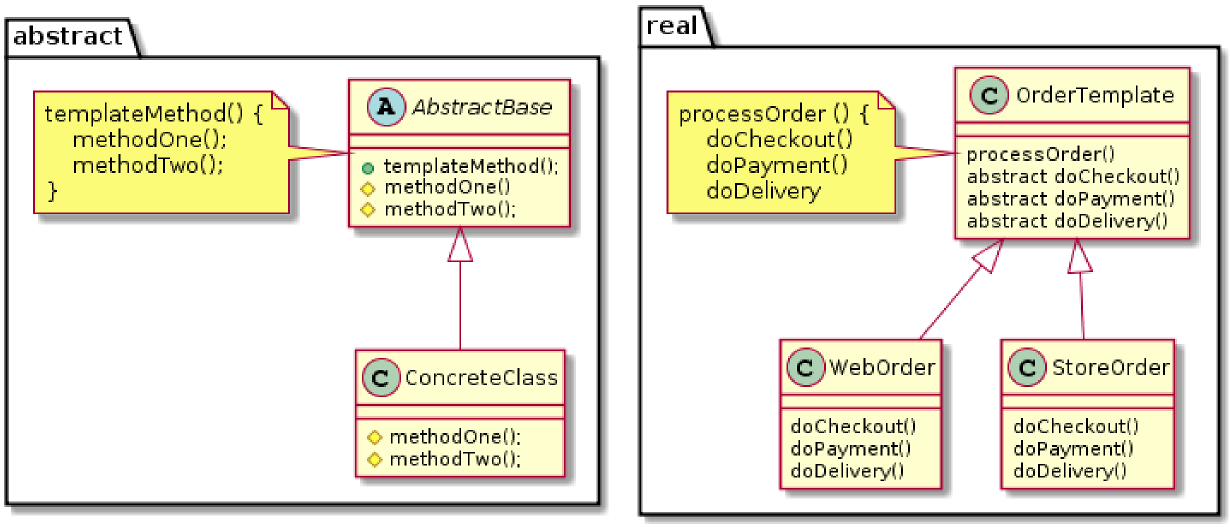


```
request() {
    //decide what strategy to use
```

```
card = new CreditCard(new VisaStrategy());
sysout("Card is :" + card.isValid());
```

```
boolean isValid(card)
    //visa algorithm
     ...
    if(valid)
        return true
    return false
```

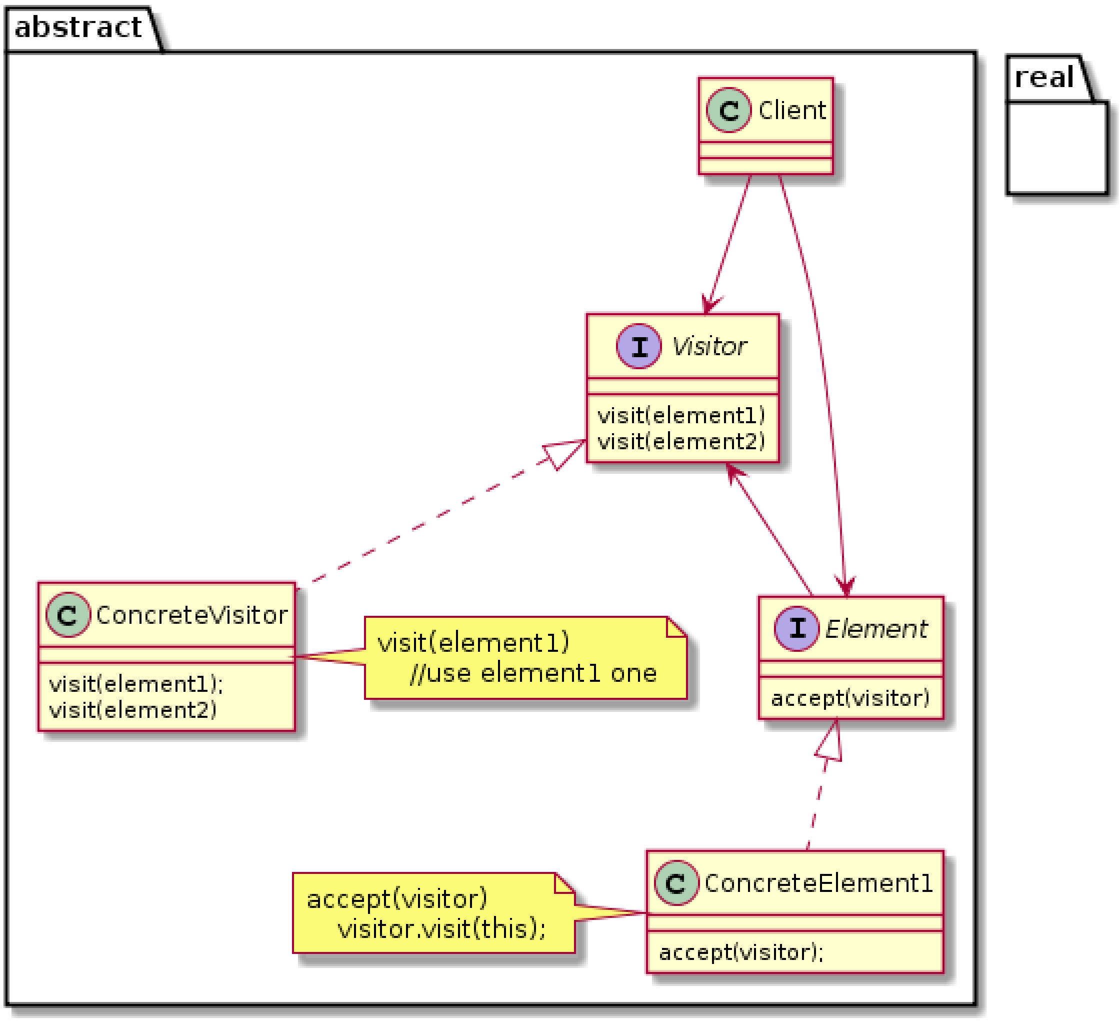
template pattern



```
templateMethod() {
    methodOne();
    methodTwo();
}
```

```
processOrder () {
    doCheckout()
    doPayment()
    doDelivery
```

visitor pattern


abstract
Client
Visitor
visit(element1)
visit(element2)
ConcreteVisitor
visit(element1);
visit(element2)
visit(element1)
//use element1 one
Element
accept(visitor)
ConcreteElement1
accept(visitor);
accept(visitor)
visitor.visit(this);
real