Padrões de Projeto de Software Orientados a Objetos

Tecnologia em Análise e Desenvolvimento de Sistemas

Paulo Mauricio Gonçalves Júnior

Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco

7 de novembro de 2018

Parte I

Outros Padrões

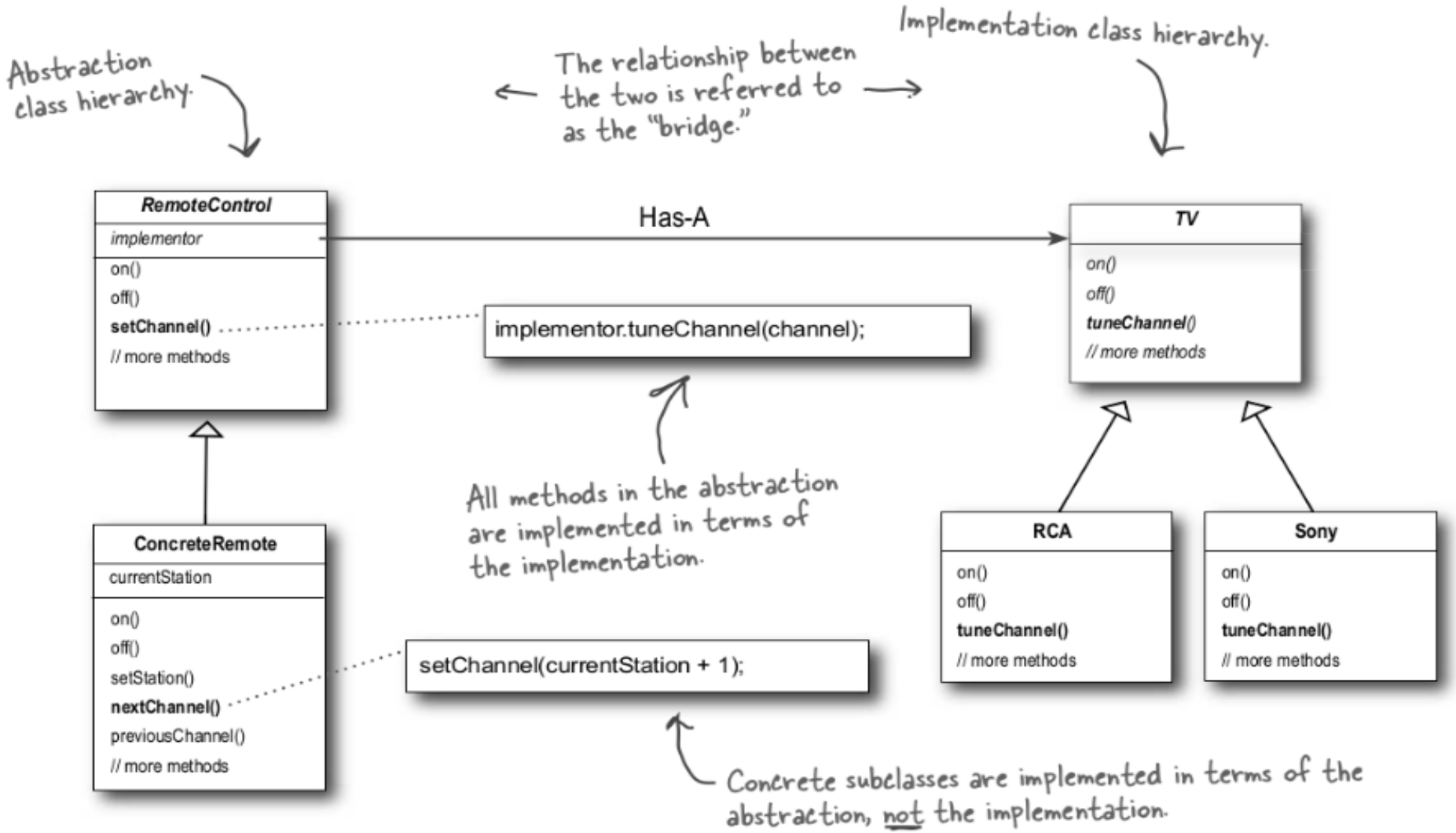
Bridge I

Definição

Use o padrão para variar não apenas suas implementações, mas também suas abstrações.

- Você vai implementar o código de um controle remoto para TVs. Você já sabe que você tem que usar boas técnicas OO porque enquanto o controle remoto é baseado na mesma abstração, existirão várias implementações – um para cada modelo de TV.
- Os controles remotos mudarão, assim como as TVs também mudarão.
- Colocaremos ambos em hierarquias de classes separadas.

Bridge II


Abstraction class hierarchy.
The relationship between the two is referred to as the "bridge."
Implementation class hierarchy.
RemoteControl
implementor
on()
off()
setChannel()
// more methods
Has-A
TV
on()
off()
tuneChannel()
// more methods
implementor.tuneChannel(channel);
All methods in the abstraction are implemented in terms of the implementation.
ConcreteRemote
currentStation
on()
off()
setStation()
nextChannel()
previousChannel()
// more methods
setChannel(currentStation + 1);
RCA
on()
off()
tuneChannel()
// more methods
Sony
on()
off()
tuneChannel()
// more methods
Concrete subclasses are implemented in terms of the abstraction, not the implementation.

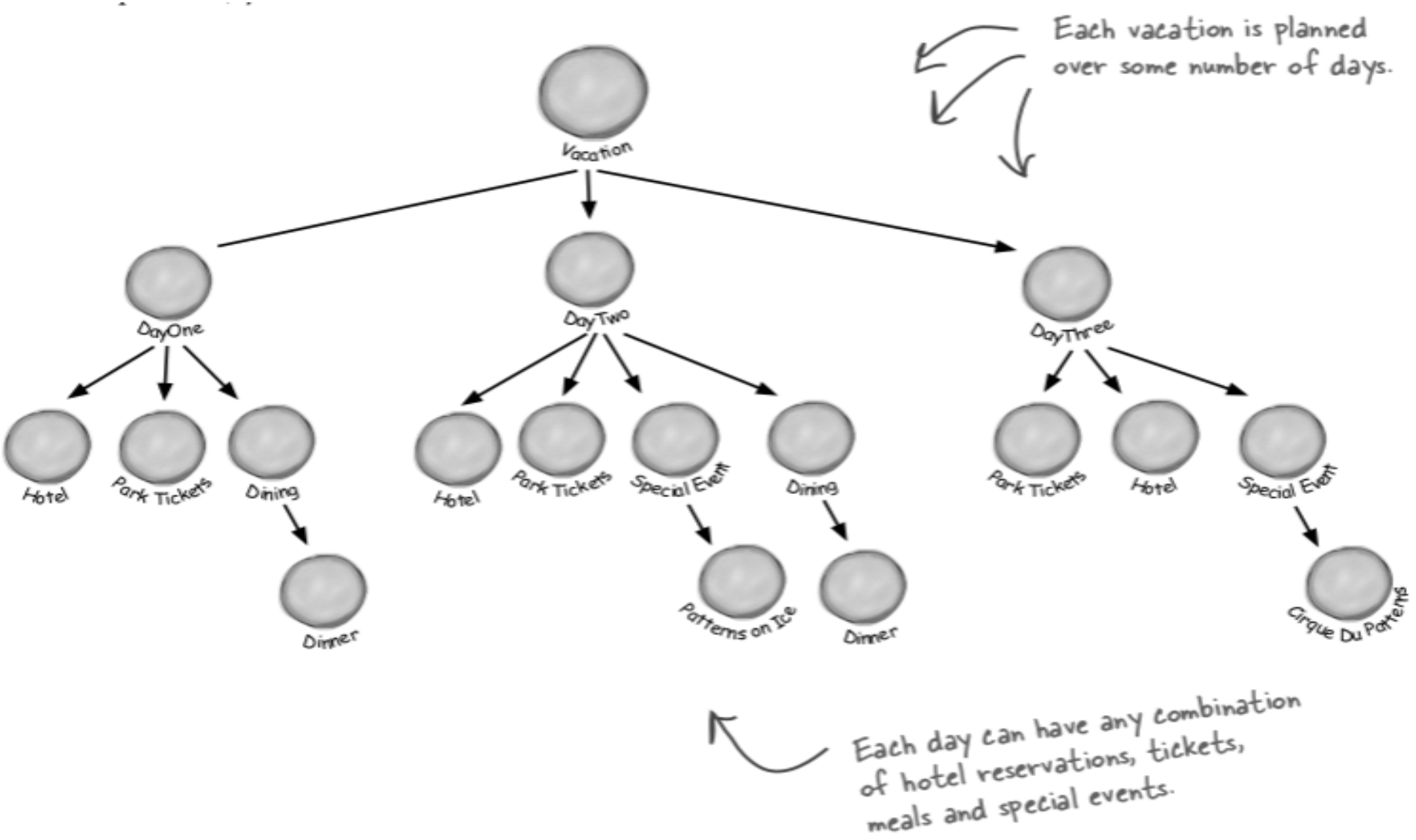
Builder I

Definição

Use o padrão para encapsular a construção de um produto e o permite ser construído em passos.

- Implementar um sistema de planejamento de viagem, podendo reservar hotel, bilhetes para parques, reservas de restaurantes, e eventos.
- O número de dias e atividades podem variar de cliente para cliente.
- Como prover uma forma de criar uma estrutura complexa sem misturar com os passos de sua criação?

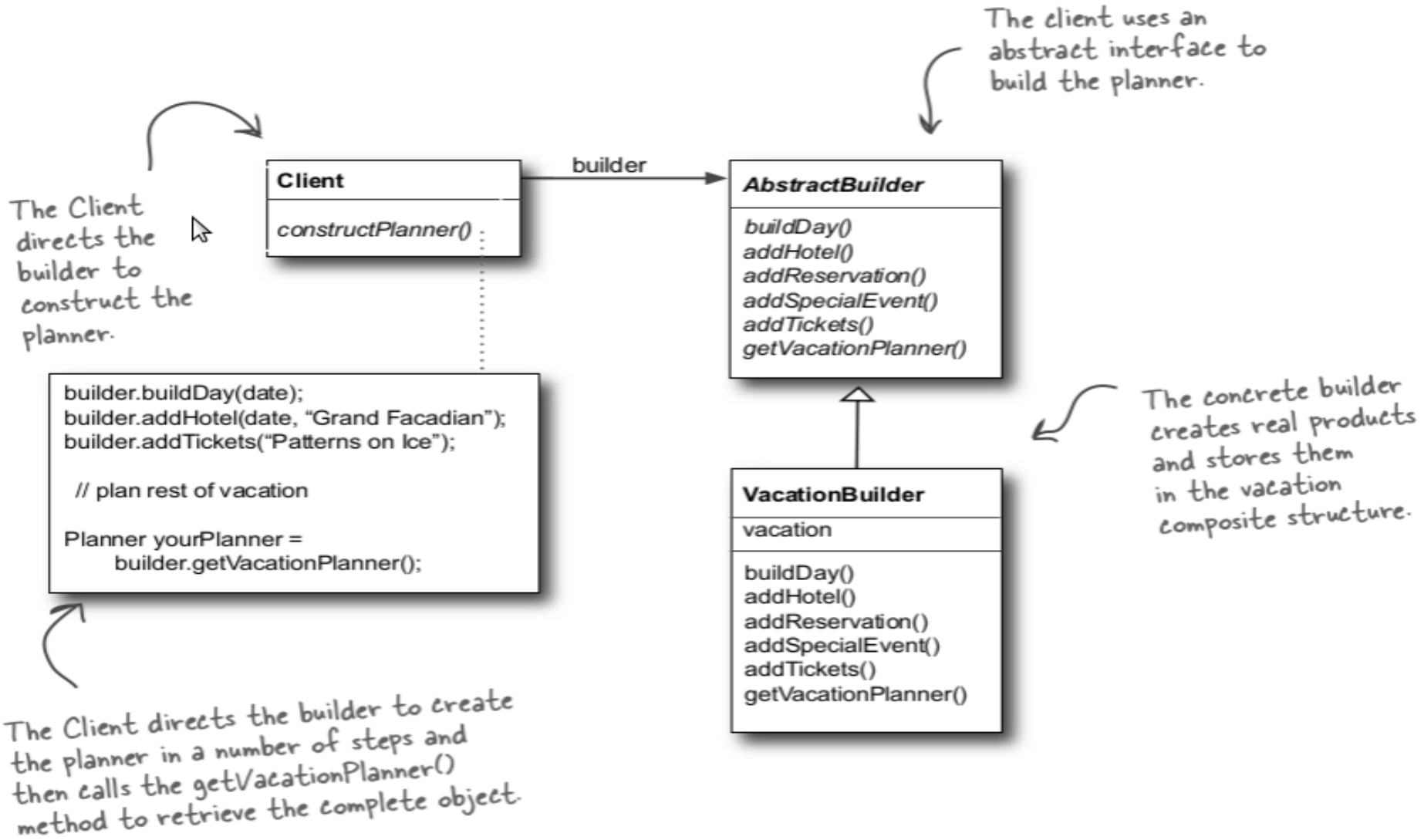
Builder II


Each vacation is planned over some number of days.
Vacation
DayOne
DayTwo
DayThree
Hotel
Park Tickets
Dining
Dinner
Hotel
Park Tickets
Special Event
Dining
Pattems on Ice
Dinner
Park Tickets
Hotel
Special Event
Cirque Du Pattems
Each day can have any combination of hotel reservations, tickets, meals and special events.

Builder III

- Encapsularemos a criação do planejador em um objeto e faremos o cliente solicitar a ele a construção da estrutura do planejamento.

Builder IV


The client uses an abstract interface to build the planner.
The Client directs the builder to construct the planner.
Client
constructPlanner()
builder
AbstractBuilder
buildDay()
addHotel()
addReservation()
addSpecialEvent()
addTickets()
getVacationPlanner()
builder.buildDay(date);
builder.addHotel(date, "Grand Facadian");
builder.addTickets("Patterns on Ice");
// plan rest of vacation
Planner yourPlanner =
builder.getVacationPlanner();
VacationBuilder
vacation
buildDay()
addHotel()
addReservation()
addSpecialEvent()
addTickets()
getVacationPlanner()
The concrete builder creates real products and stores them in the vacation composite structure.
The Client directs the builder to create the planner in a number of steps and then calls the getVacationPlanner() method to retrieve the complete object.

Chain of Responsibility I

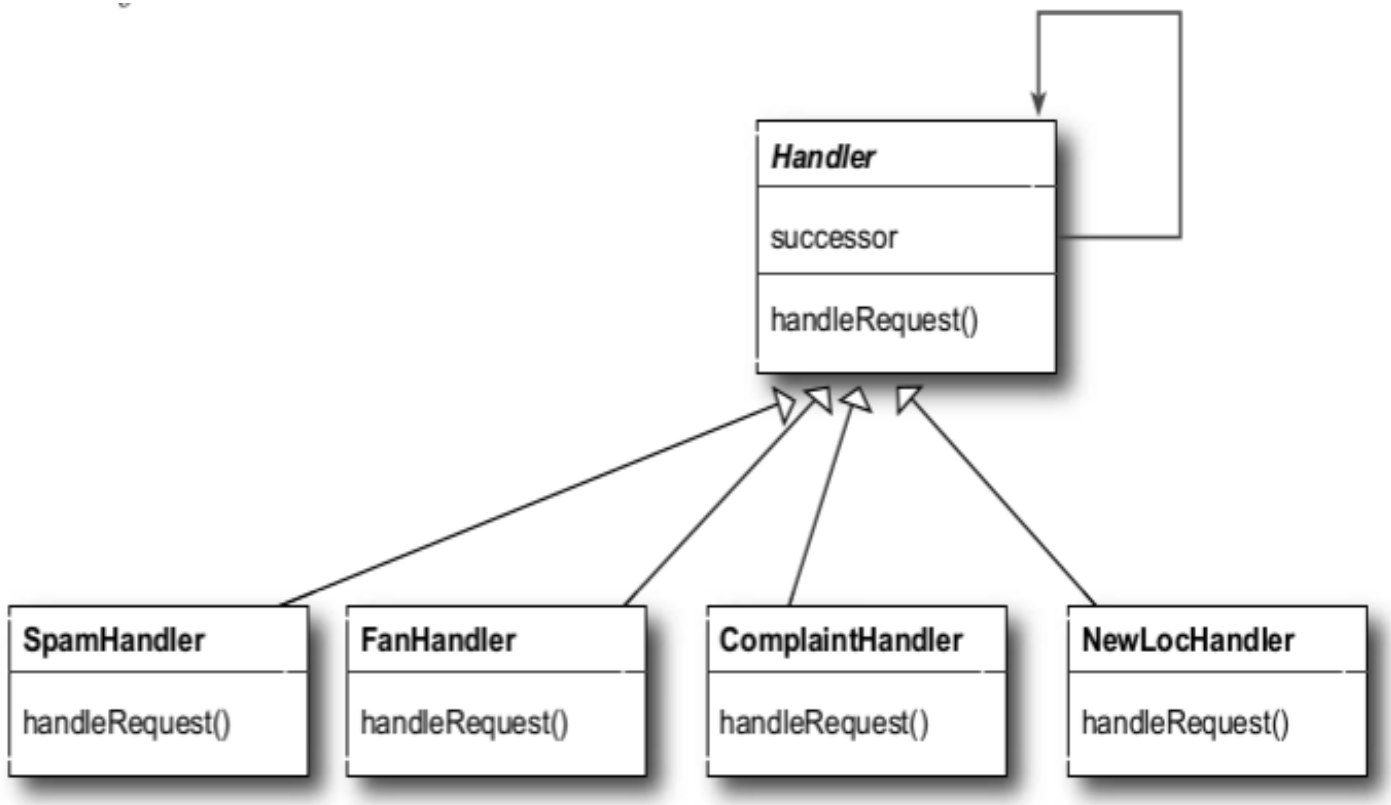
Definição

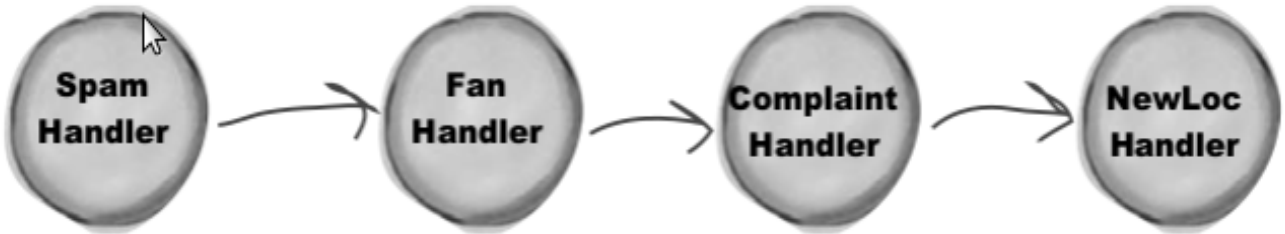
Use o padrão quando você quer dar a mais de um objeto a oportunidade de tratar um pedido.

- Implementar um sistema para, a partir de um categorizador de e-mail, enviar as mensagens automaticamente para o destinatário específico.
- Criamos uma cadeia de objetos que examinam o pedido. Cada objeto por sua vez examina o pedido e o trata, ou passa para o próximo objeto na cadeia.

Chain of Responsibility II

Each object in the chain acts as a handler and has a successor object. If it can handle the request, it does; otherwise, it forwards the request to its successor.





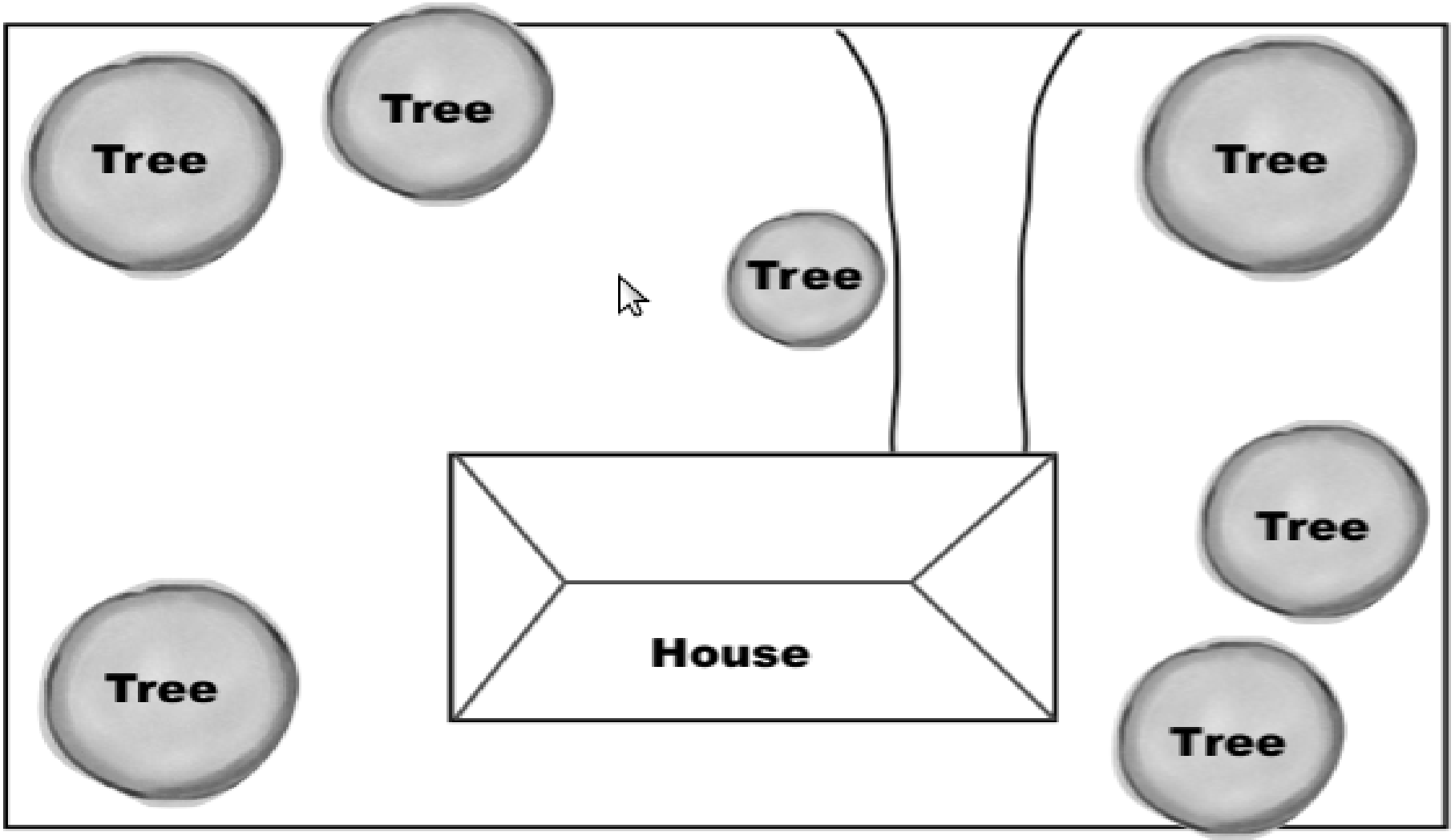
Flyweight I

Definição

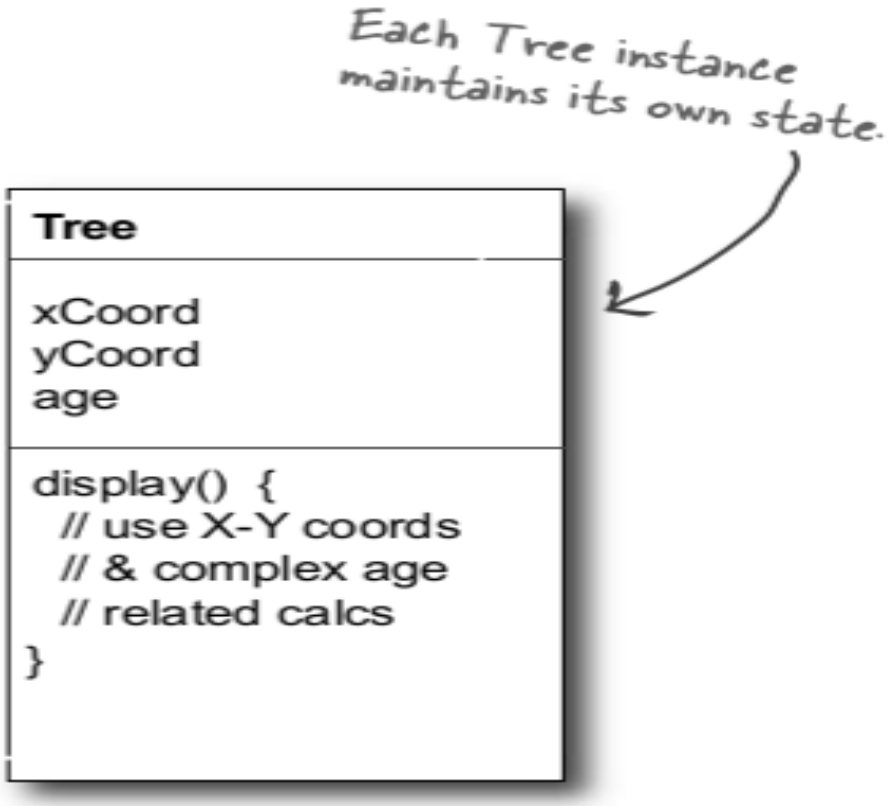
Use o padrão quando uma instância de uma classe pode ser usada para prover várias "instâncias virtuais."

- Você quer adicionar árvores em um aplicativo de projeto de exteriores. Nele, árvores não fazem muita coisa; eles possuem uma posição X-Y, e eles podem desenhar a si próprios dinamicamente, dependendo de sua idade. Várias árvores podem ser adicionadas nos projetos.

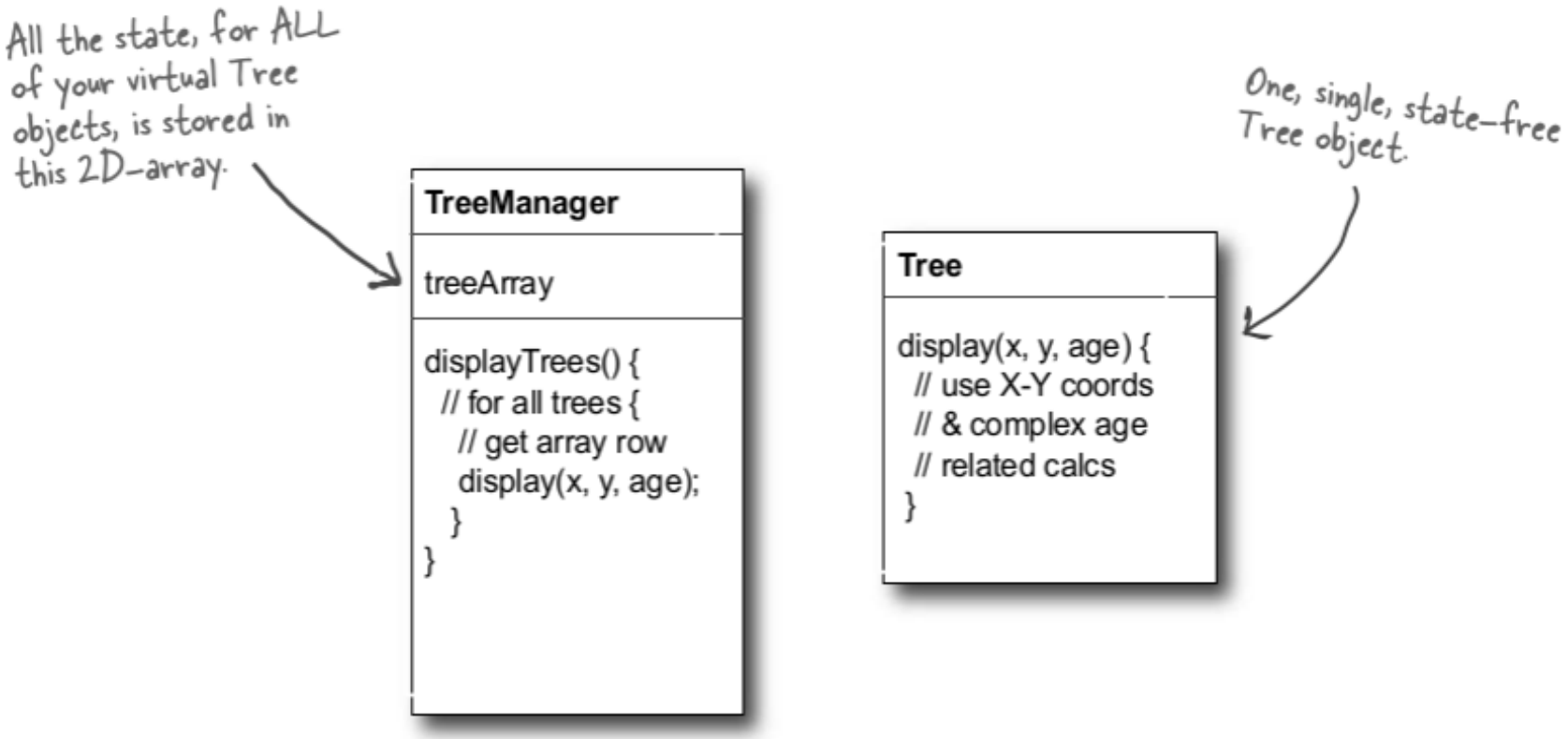
Flyweight II



Flyweight III



Flyweight IV



Interpreter I

Definição

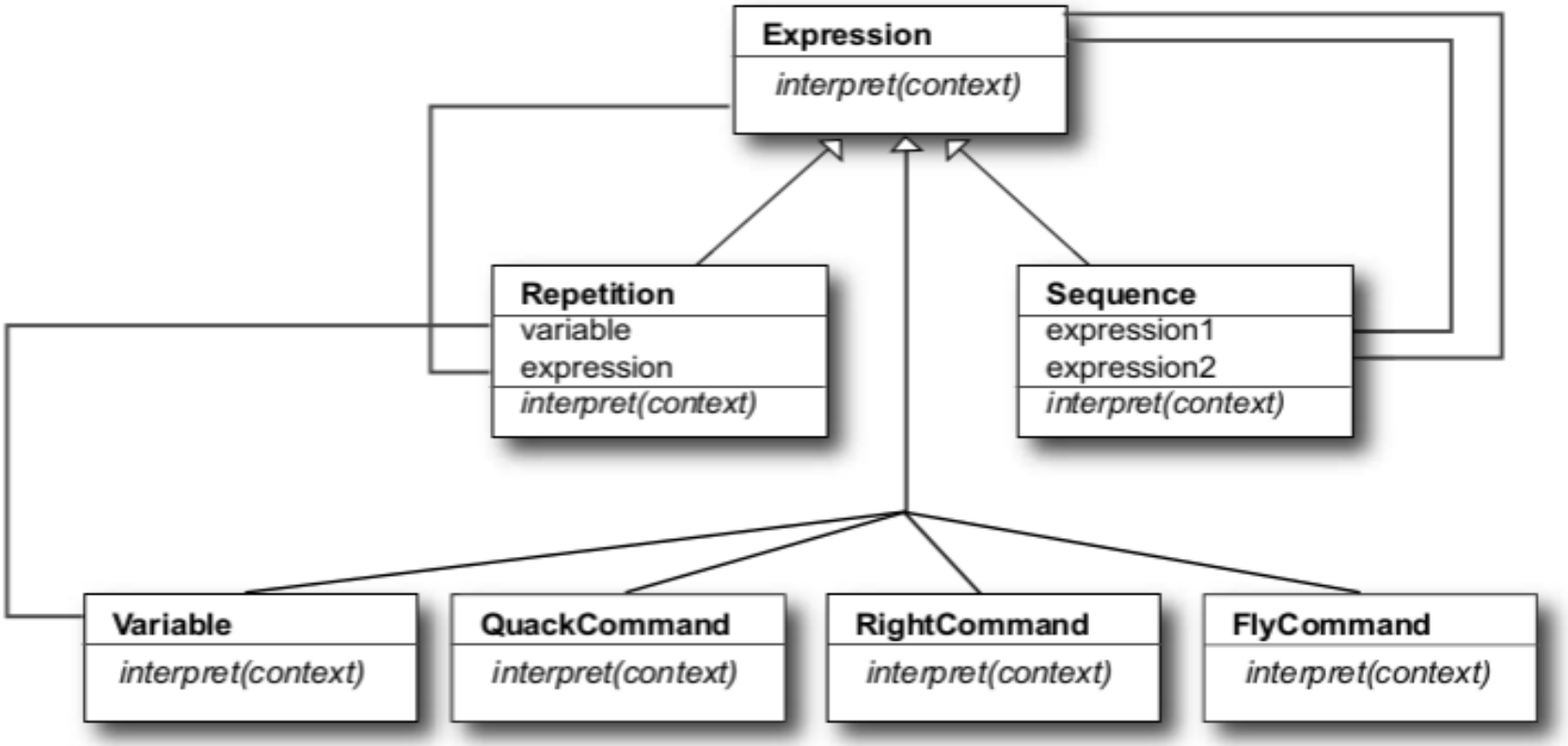
Use o padrão para construir um interpretador para uma linguagem.

- Uma classe representará cada regra da linguagem.

```
right;
while (daylight) fly;
quack;

expression ::= <command> | <sequence> | <repetition>
sequence ::= <expression> ';' <expression>
command ::= right | quack | fly
repetition ::= while '(' <variable> ')' <expresion>
variable ::= [A-Z,a-z]+
```

Interpreter II



Mediator I

Definição

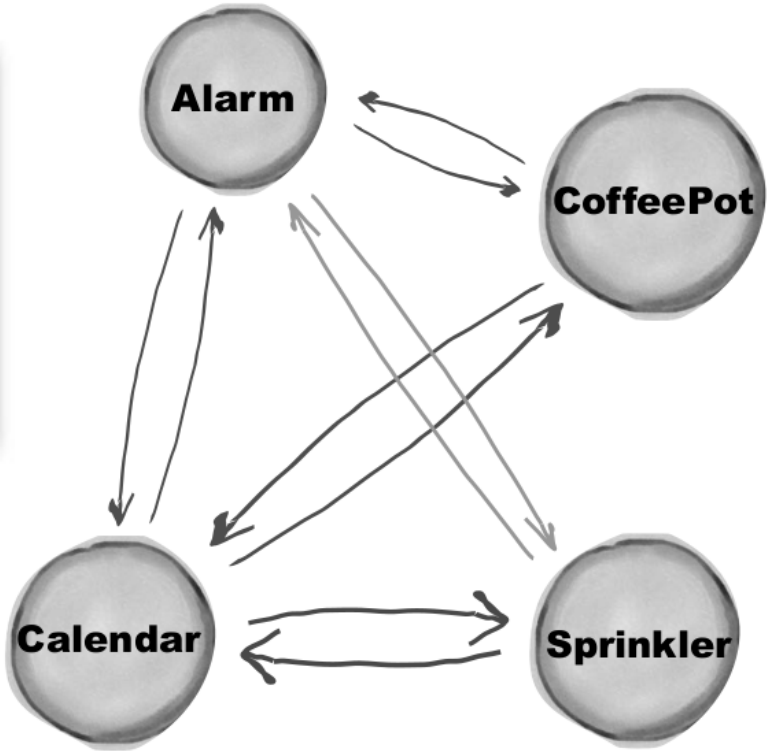
Use o padrão para centralizar comunicação e controles complexos entre objetos relacionados.

- Internet das coisas: casa do futuro
- É difícil identificar quais regras ficam em quais objetos, e como eles se relacionam entre si.

Mediator II

Alarm

```
onEvent() {
  checkCalendar()
  checkSprinkler()
  startCoffee()
  // do more stuff
}
```



CoffeePot

```
onEvent() {
  checkCalendar()
  checkAlarm()
  // do more stuff
}
```

Calendar

```
onEvent() {
  checkDayOfWeek()
  doSprinkler()
  doCoffee()
  doAlarm()
  // do more stuff
}
```

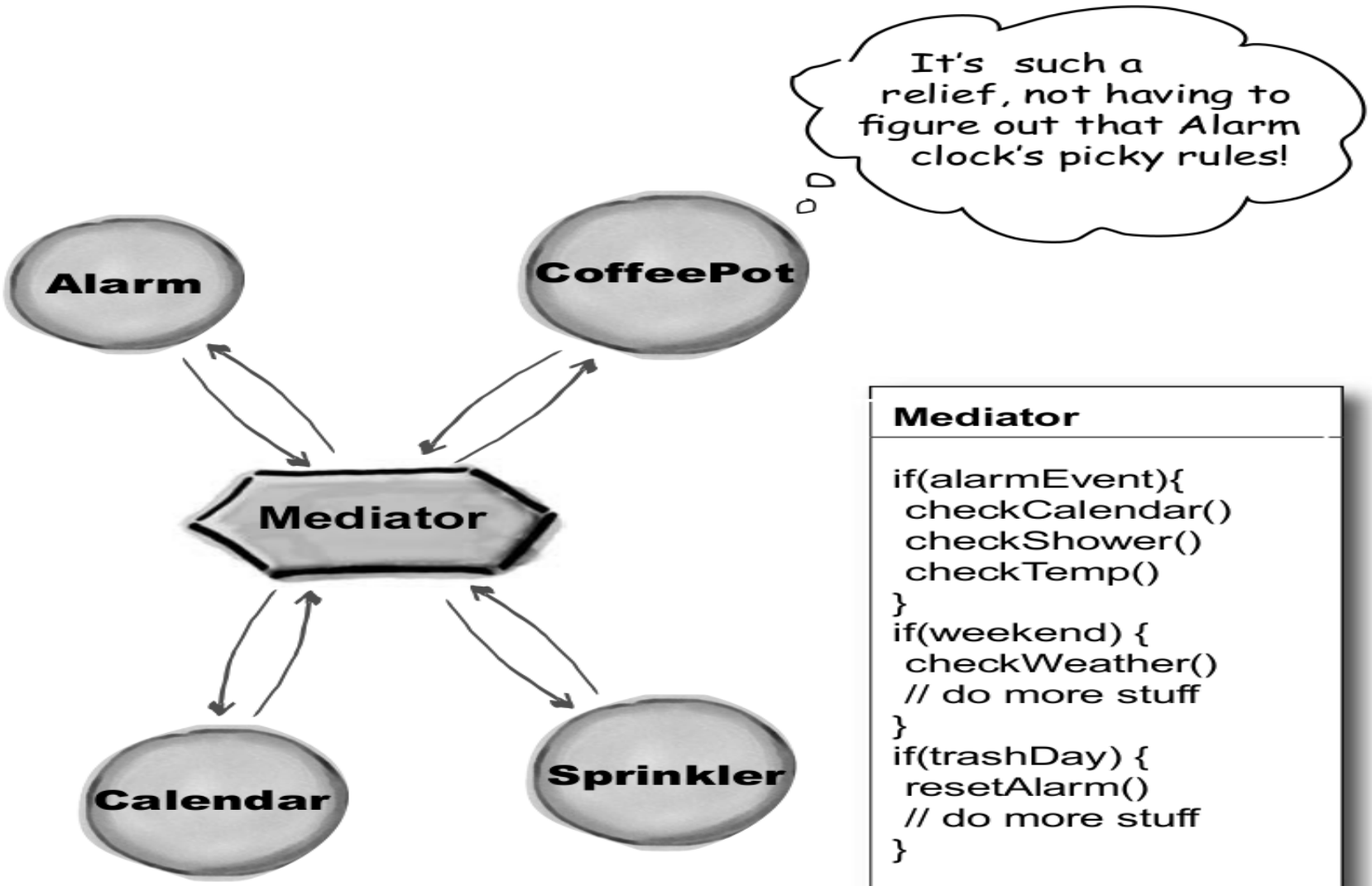
Sprinkler

```
onEvent() {
  checkCalendar()
  checkShower()
  checkTemp()
  checkWeather()
  // do more stuff
}
```

Mediator III

- Um objeto Mediador intermedeia a comunicação entre os objetos, simplificando-os.
- Eles informam o Mediador quando seu estado interno muda.
- Eles respondem a solicitações do Mediador.

Mediator IV


It's such a relief, not having to figure out that Alarm clock's picky rules!
Alarm
CoffeePot
Mediator
Calendar
Sprinkler
Mediator
if(alarmEvent){
checkCalendar()
checkShower()
checkTemp()
}
if(weekend) {
checkWeather()
// do more stuff
}
if(trashDay) {
resetAlarm()
// do more stuff
}

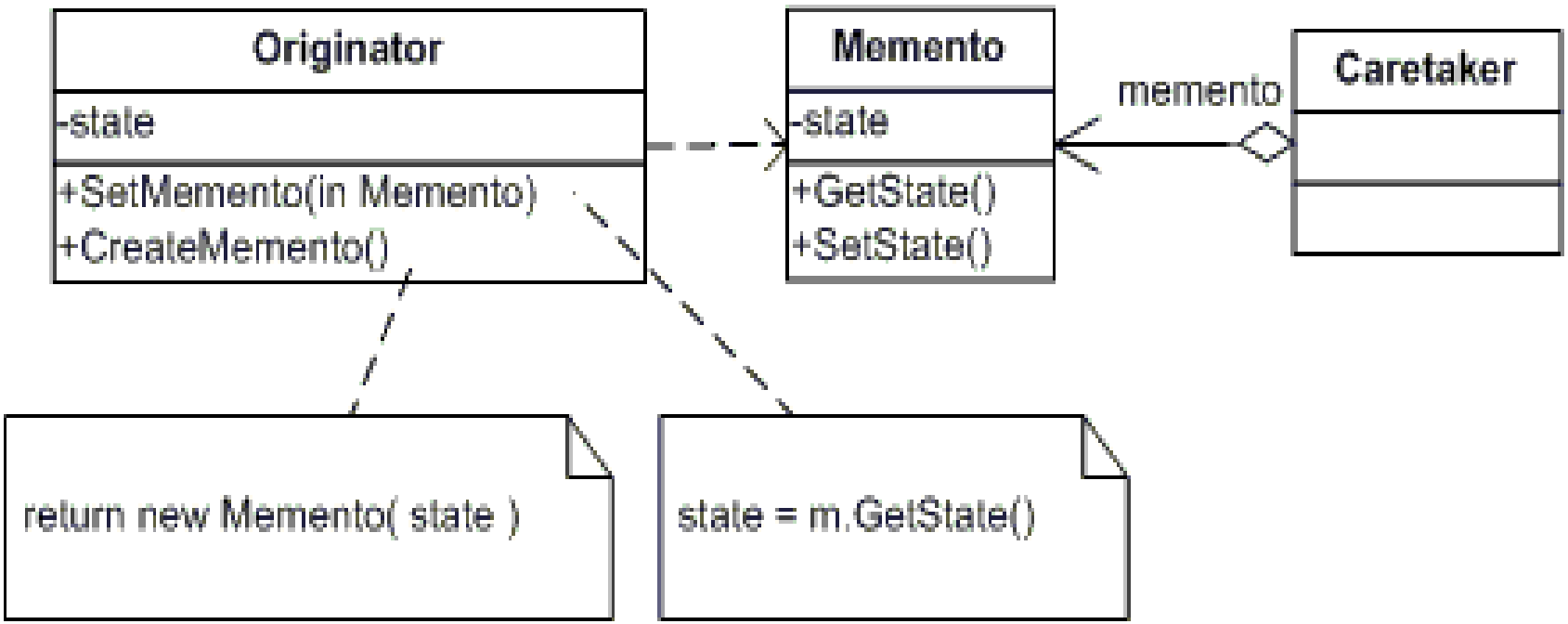
Memento I

Definição

Use o padrão quando você precisa retornar um objeto a um estado anterior; por exemplo, se o usuário solicita um "desfazer."

- Objetivos
  - Salvar o estado importante de um objeto chave do sistema.
  - Manter a encapsulação deste objeto chave.

Memento II


Originator
-state
+SetMemento(in Memento)
+CreateMemento()
Memento
-state
+GetState()
+SetState()
memento
Caretaker
return new Memento(state)
state = m.GetState()

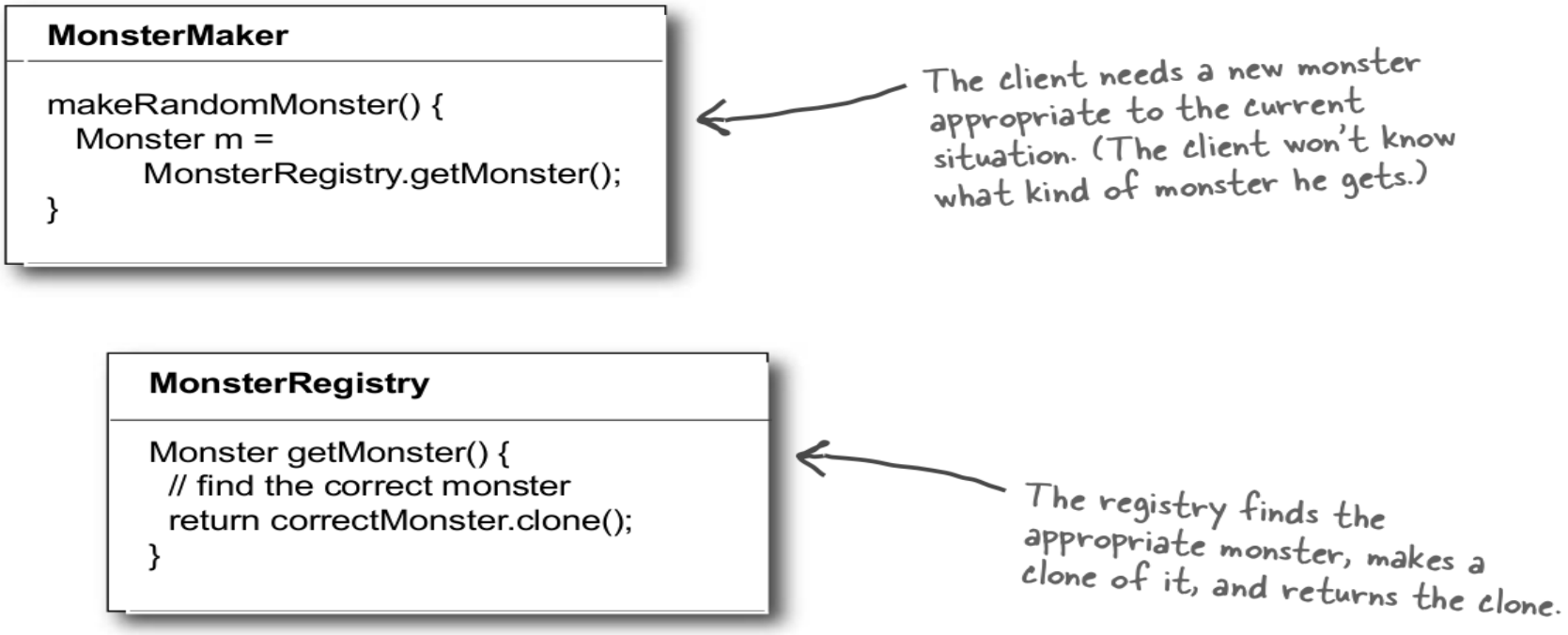
Prototype I

Definição

Use o padrão quando criar uma instância de uma classe é custosa ou complicada.

- Criar inimigos em um jogo, cujas características devem mudar de acordo com o ambiente.
- Permite criar novas instâncias copiando instâncias existentes.

Prototype II



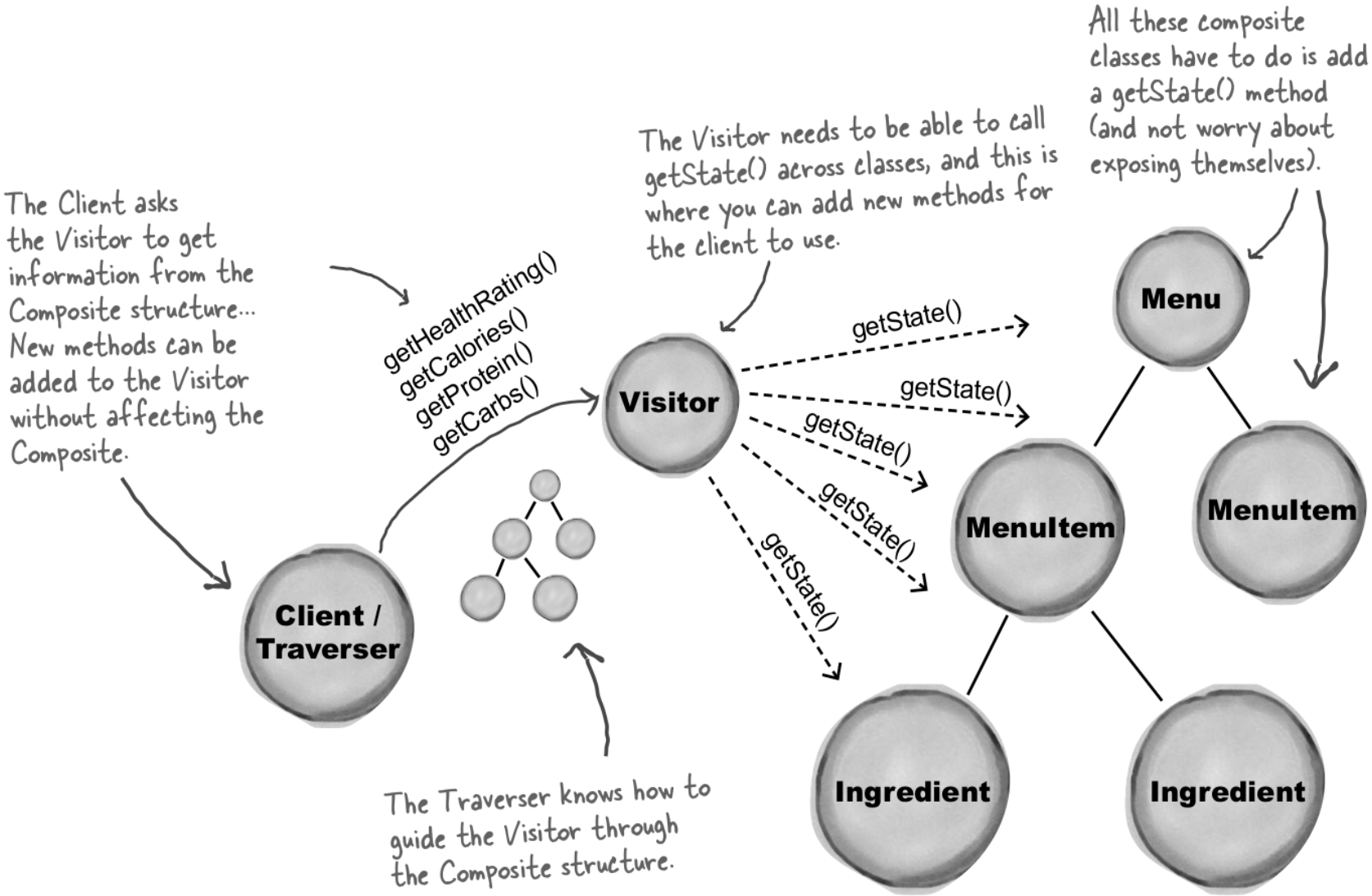
Visitor I

Definição

Use o padrão quando você quer adicionar capacidades a um objeto composto e o encapsulamento não é importante.

- Clientes de um restaurante querem informações nutricionais dos ingredientes dos pratos.
- Um objeto Traverser visita todos os elementos da composição. O objeto Visitor recebe o estado dos elementos e realiza operações no estado. Apenas o Visitor é mudado quando novas funcionalidades são requeridas.

Visitor II


All these composite classes have to do is add a getState() method (and not worry about exposing themselves).
The Visitor needs to be able to call getState() across classes, and this is where you can add new methods for the client to use.
The Client asks the Visitor to get information from the Composite structure... New methods can be added to the Visitor without affecting the Composite.
getHealthRating()
getCalories()
getProtein()
getCarbs()
Visitor
getState()
getState()
getState()
getState()
getState()
Menu
MenuItem
MenuItem
Ingredient
Ingredient
Client / Traverser
The Traverser knows how to guide the Visitor through the Composite structure.