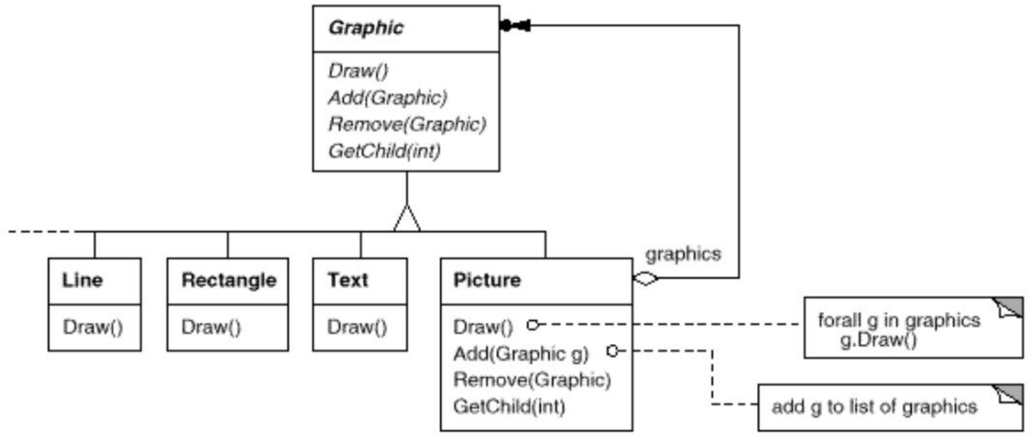
Composite、Flyweight、Chain of Responsibility

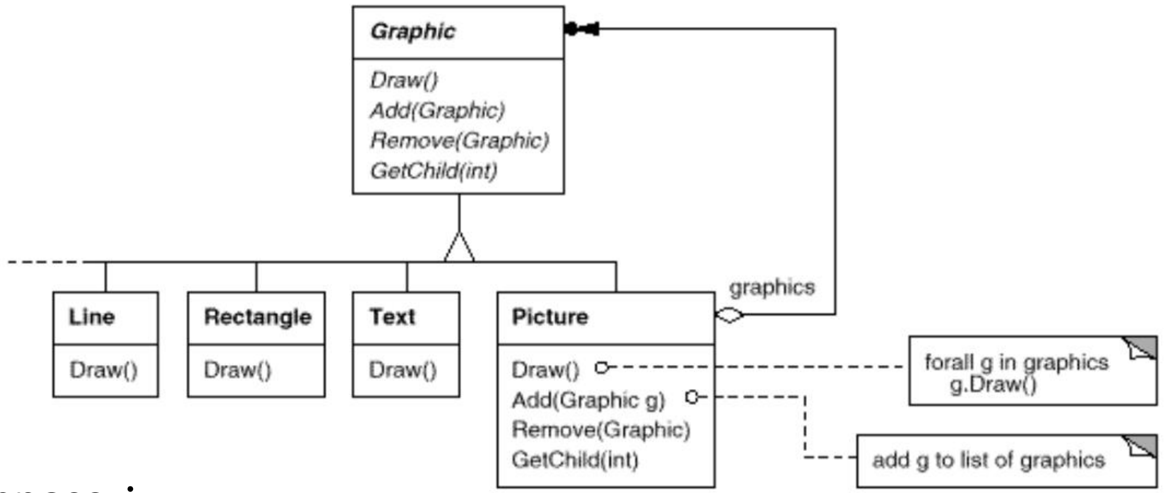
2019/10/30 OOSE課輔

Composite

- Intent
  - Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.



Composite



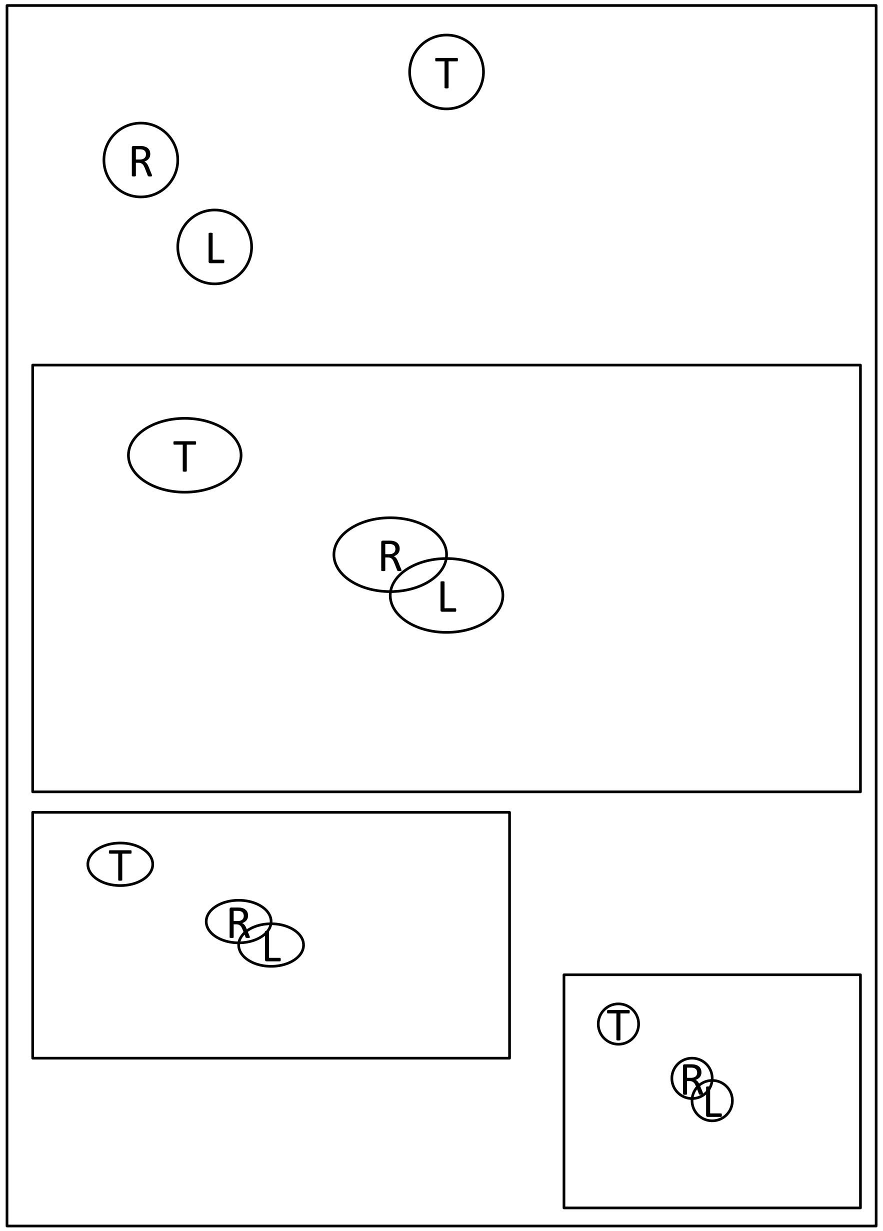
Suppose：

Composite □ <- Picture

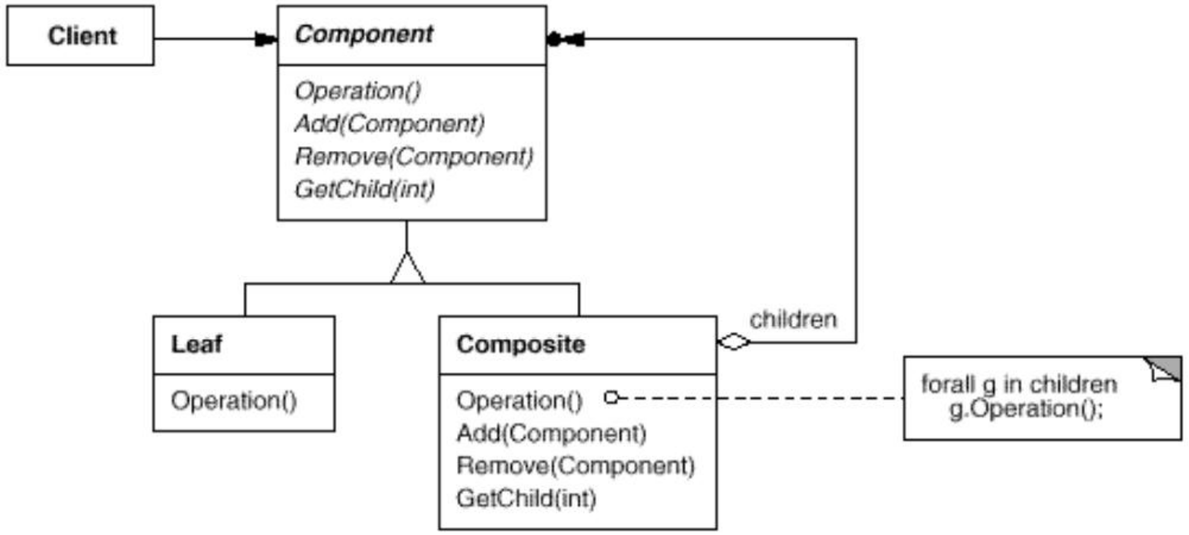
Leaf Ⓣ <- Text

Leaf Ⓡ <- Rectangle

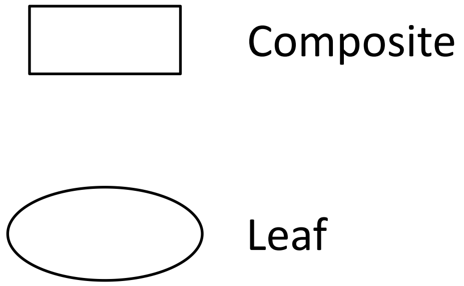
Leaf Ⓛ <- Line



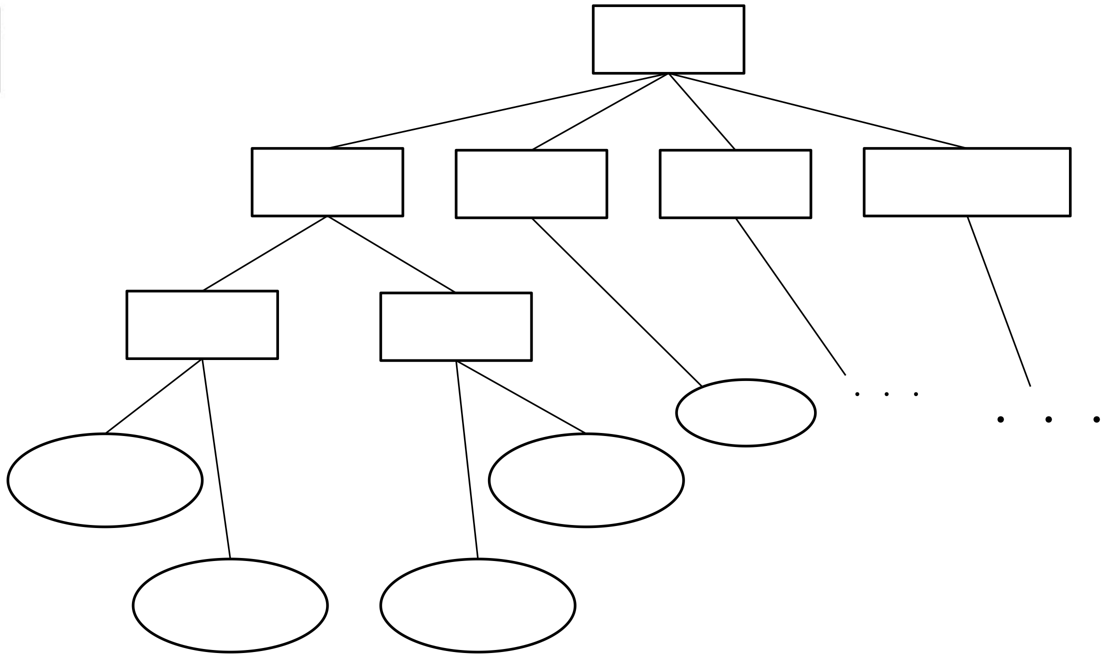
Structure



Client
Component
Operation()
Add(Component)
Remove(Component)
GetChild(int)
Leaf
Operation()
Composite
Operation()
Add(Component)
Remove(Component)
GetChild(int)
children
forall g in children
g.Operation();

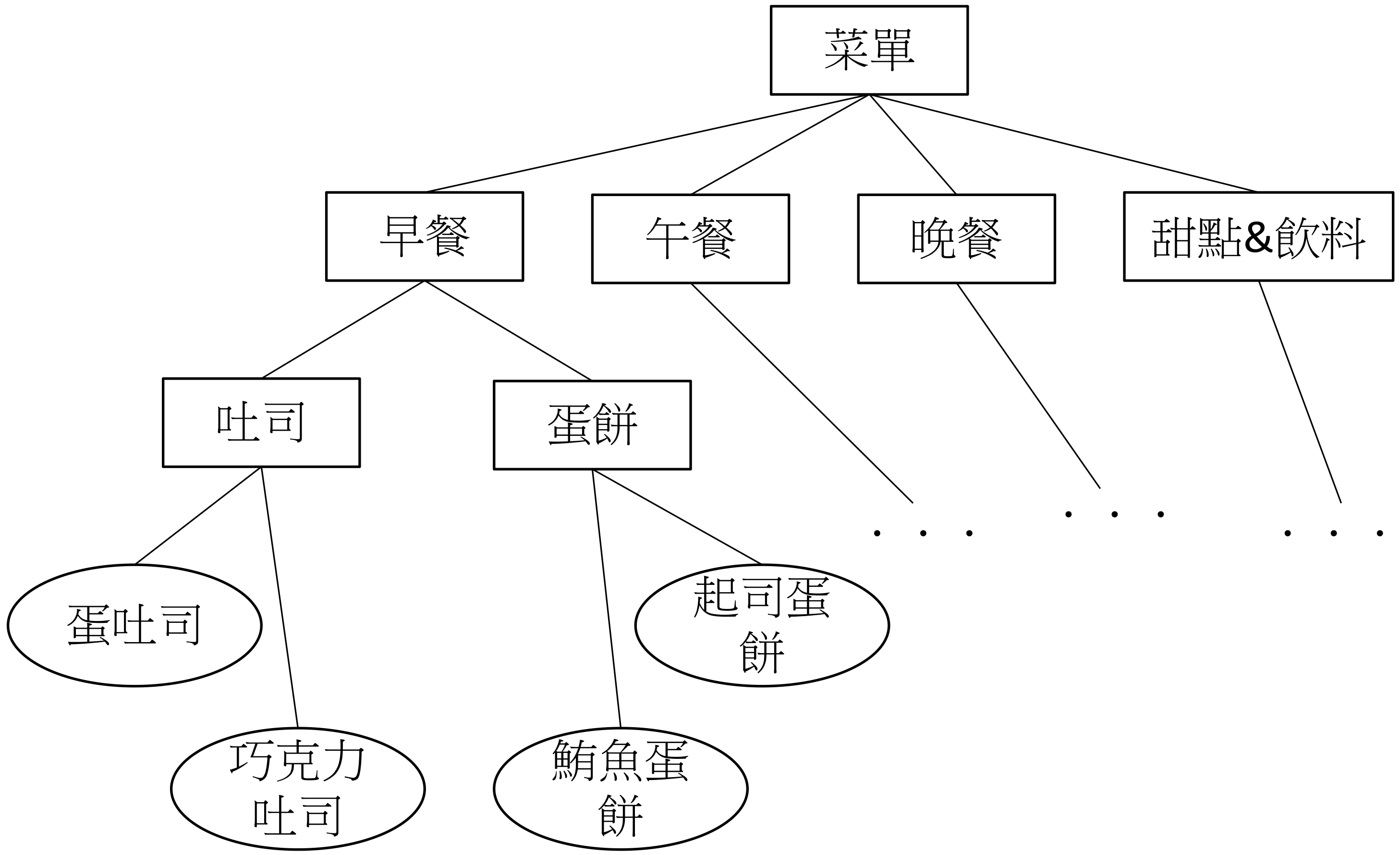


Composite
Leaf



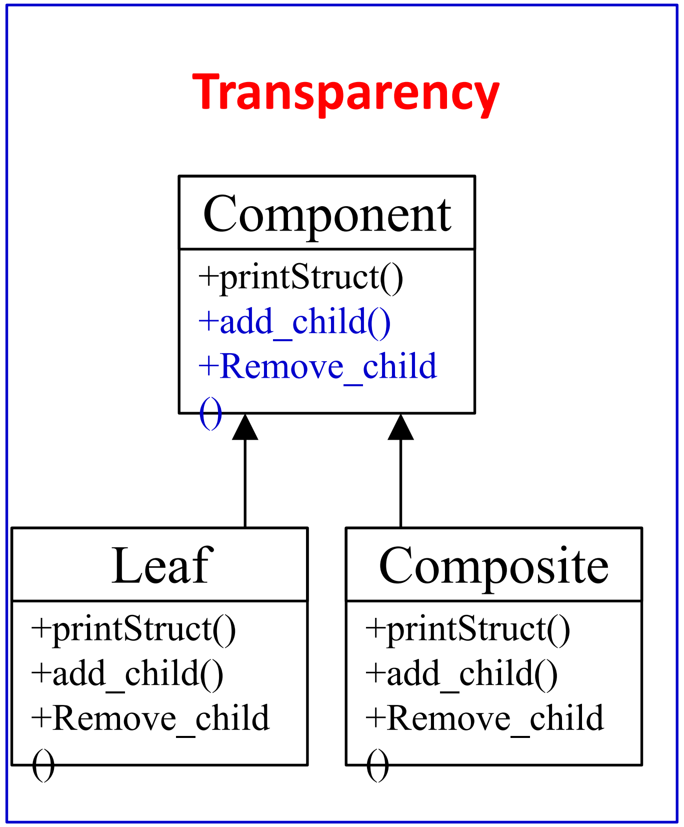
Example

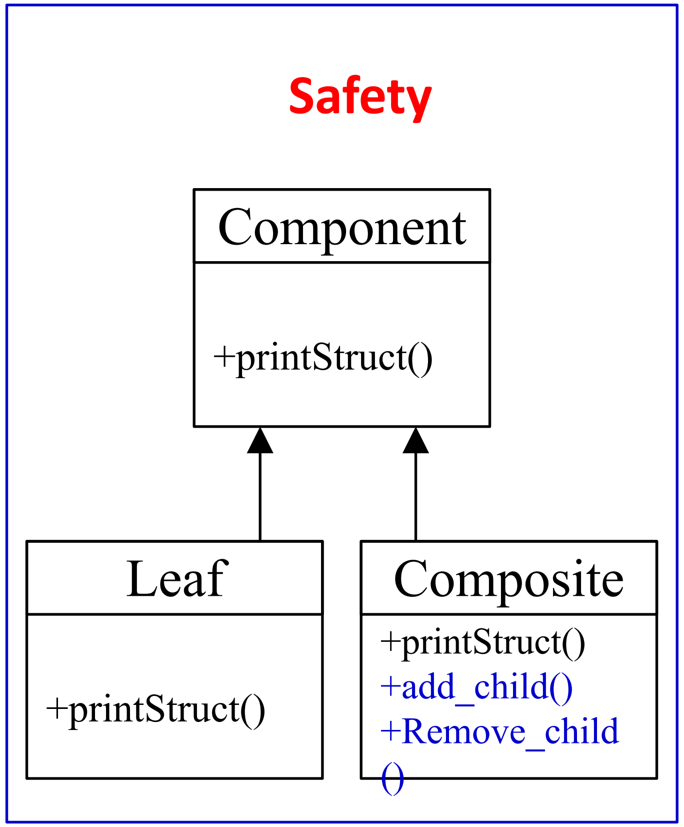
- 菜單
  - 早餐
    - 吐司
      - 蛋吐司
      - 巧克力吐司
    - 蛋餅
      - 起司蛋餅
      - 鮪魚蛋餅
  - 午餐
    - …
  - 晚餐
    - …
  - 甜點&飲料
    - …



Safety & Transparency

- 對Composite和Leaf一視同仁，無需特別區分，但可能會有設計出Leaf能使用add_child() & Remove_child()的情況。



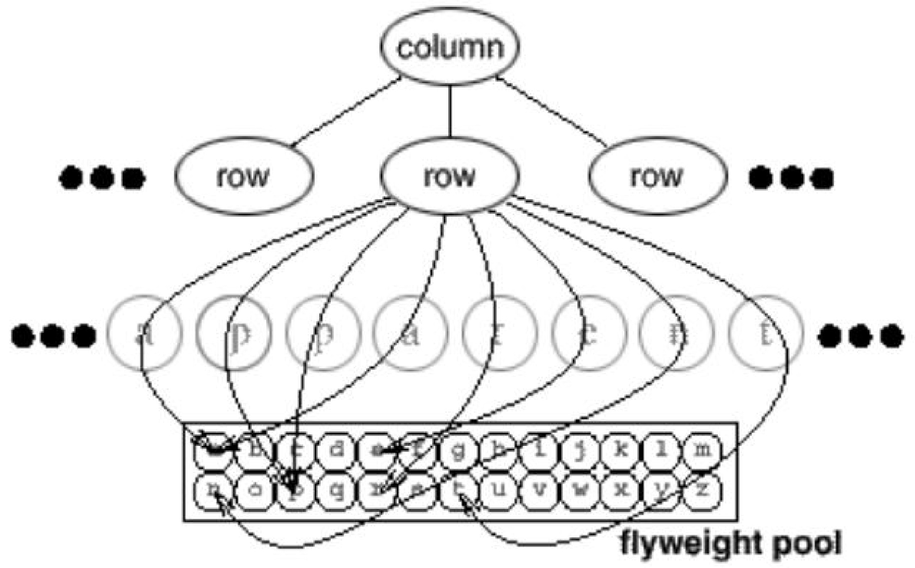


- 使用者需要去特地區分是在使用Composite還是Leaf。

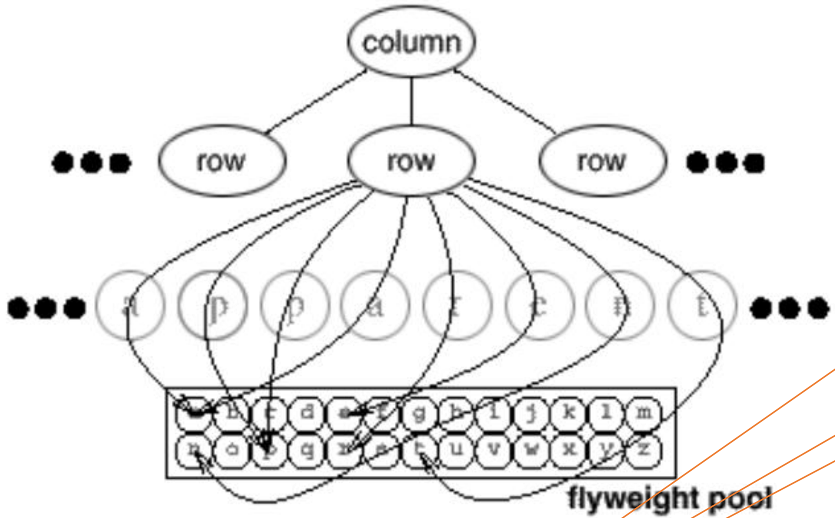
Code of Example

Flyweight

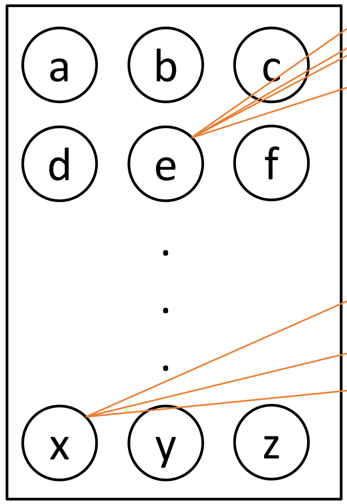
- Intent
  - Use sharing to support large numbers of fine-grained objects efficiently.



Flyweight

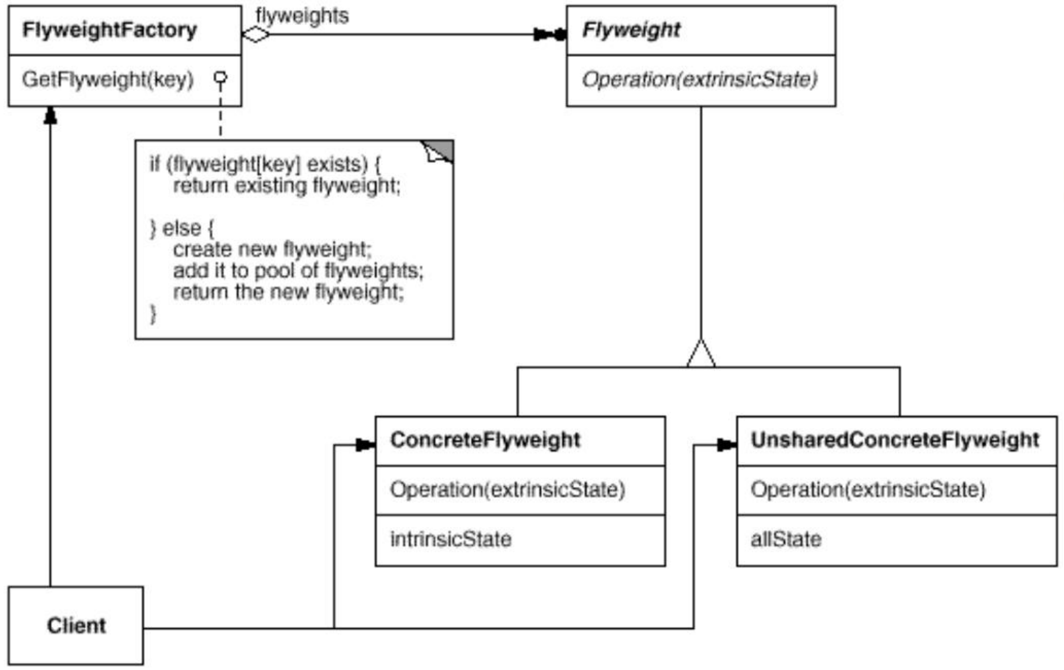


Suppose：



The class structure for these objects is shown next. Glyph is the abstract class for graphical objects, some of which may be flyweights. Operations that may depend on extrinsic state have it passed to them as a parameter. For example, Draw and Intersects must know which context the glyph is in before they can do their job.

Structure


FlyweightFactory
GetFlyweight(key)
flyweights
Flyweight
Operation(extrinsicState)
if (flyweight[key] exists) {
return existing flyweight;
} else {
create new flyweight;
add it to pool of flyweights;
return the new flyweight;
}
ConcreteFlyweight
Operation(extrinsicState)
intrinsicState
UnsharedConcreteFlyweight
Operation(extrinsicState)
allState
Client

Flyweight

- declares an interface through which flyweights can receive and act on extrinsic state.

ConcreteFlyweight (Character)

- implements the Flyweight interface and adds storage for intrinsic state, if any. A ConcreteFlyweight object must be sharable. Any state it stores must be intrinsic; that is, it must be independent of the ConcreteFlyweight object's context.

UnsharedConcreteFlyweight (Row, Column)

- not all Flyweight subclasses need to be shared. The Flyweight interface *enables* sharing; it doesn't enforce it. It's common for UnsharedConcreteFlyweight objects to have ConcreteFlyweight objects as children at some level in the flyweight object structure (as the Row and Column classes have).

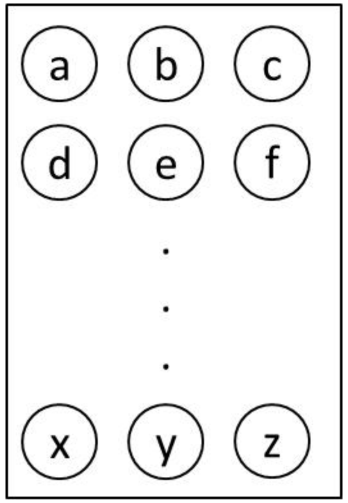
FlyweightFactory

- creates and manages flyweight objects.
- ensures that flyweights are shared properly. When a client requests a flyweight, the FlyweightFactory object supplies an existing instance or creates one, if none exists.

Client

- maintains a reference to flyweight(s).
- computes or stores the extrinsic state of flyweight(s).

Example



Intrinsic：letter	Extrinsic：position
e	see table

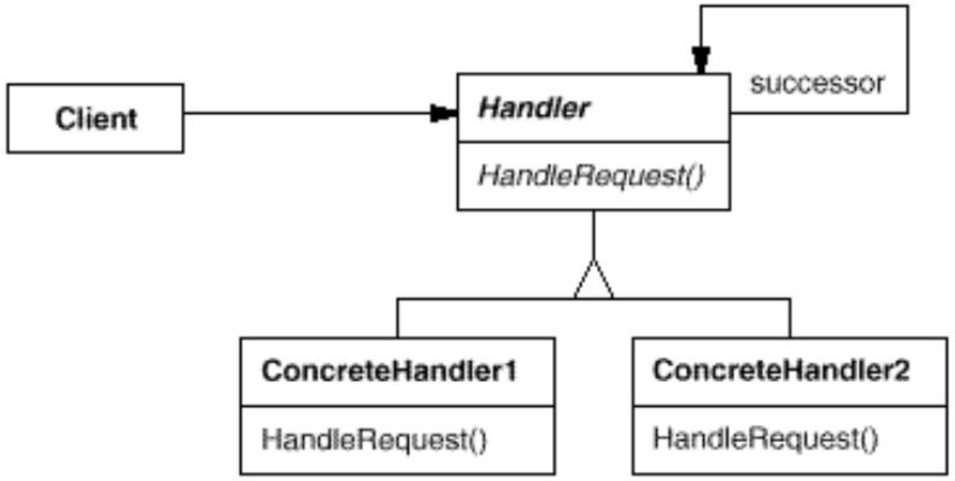
Column	Row
3	1
3	3
5	3
8	4
9	2

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	T	h	e		c	l	a	s	s																							
2	s	t	r	u	c	t	u	r	e		f	o	r																			
3	t	h	e	s	e		o	b	j	e	c	t	s		i	s																
4	s	h	o	w	n		n	e	x	t	.																					
5	G	l	y	p	h		i	s		t	h	e		a	b	s	t	r	a	c	t											
6	c	l	a	s	s		f	o	r		g	r	a	p	h	i	c	a	l													
7	o	b	j	e	c	t	s	,		s	o	m	e		o	f																
8	w	h	i	c	h		m	a	y		b	e		f	l	y	w	e	i	g	h	t	s	.								
9	O	p	e	r	a	t	i	o	n	s		t	h	a	t		m	a	y		d	e	p	e	n	d						
10	o	n		e	x	t	r	i	n	s	i	c		s	t	a	t	e		h	a	v	e		i	t						
11	p	a	s	s	e	d		t	o		t	h	e	m		a	s		a		p	a	r	a	m	e	t	e	r	.		
12	F	o	r		e	x	a	m	p	l	e	,		D	r	a	w		a	n	d		I	n	t	e	r	s	e	c	t	s
13	m	u	s	t		k	n	o	w		w	h	i	c	h		c	o	n	t	e	x	t		t	h	e					
14	g	l	y	p	h		i	s		i	n		b	e	f	o	r	e		t	h	e	y									
15	c	a	n		d	o		t	h	e	i	r		j	o	b	.															

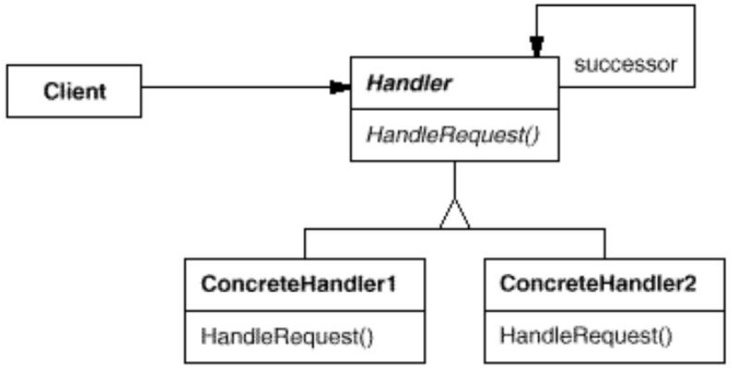
Code of Example

Chain of Responsibility

- Intent
  - Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.



Structure



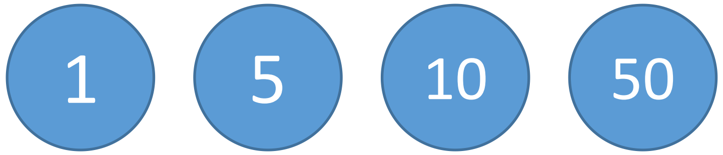
- **Client**
  - initiates the request to a ConcreteHandler object on the chain.
- **Handler** (HelpHandler)
  - defines an interface for handling requests.
  - (optional) implements the successor link.
- **ConcreteHandler** (PrintButton, PrintDialog)
  - handles requests it is responsible for.
  - can access its successor.
  - if the ConcreteHandler can handle the request, it does so; otherwise it forwards the request to its successor.

Collaborations

- When a client issues a request, the request propagates along the chainuntil a ConcreteHandler object takes responsibility for handling it.

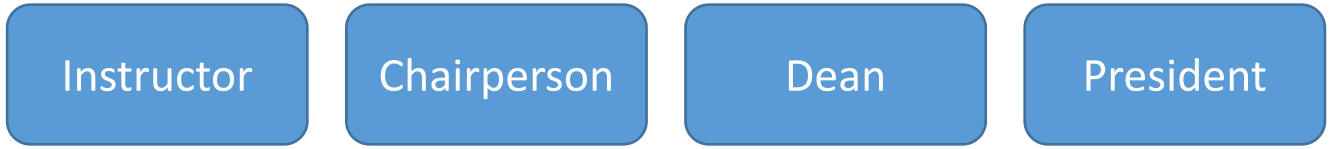
Example

- Vending Machine



Who has the greatest power to deal with the problem?

- Personal Leave



Who has the greatest power to deal with the problem?
Is it appropriate that the president makes this approval?
Does this pattern need default if it cannot handle the question?
Is this pattern based on the priority? What is the rule of this pattern?

Code of Example