Composite Pattern

Shin-Jie Lee (李信杰)
Associate Professor
Computer and Network Center
Department of CSIE
National Cheng Kung University

Design Aspect of Composite

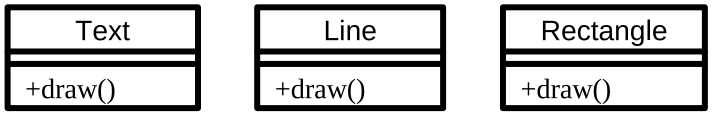
Structure and composition of an object

Outline

Schematic Capture Systems (Composite)

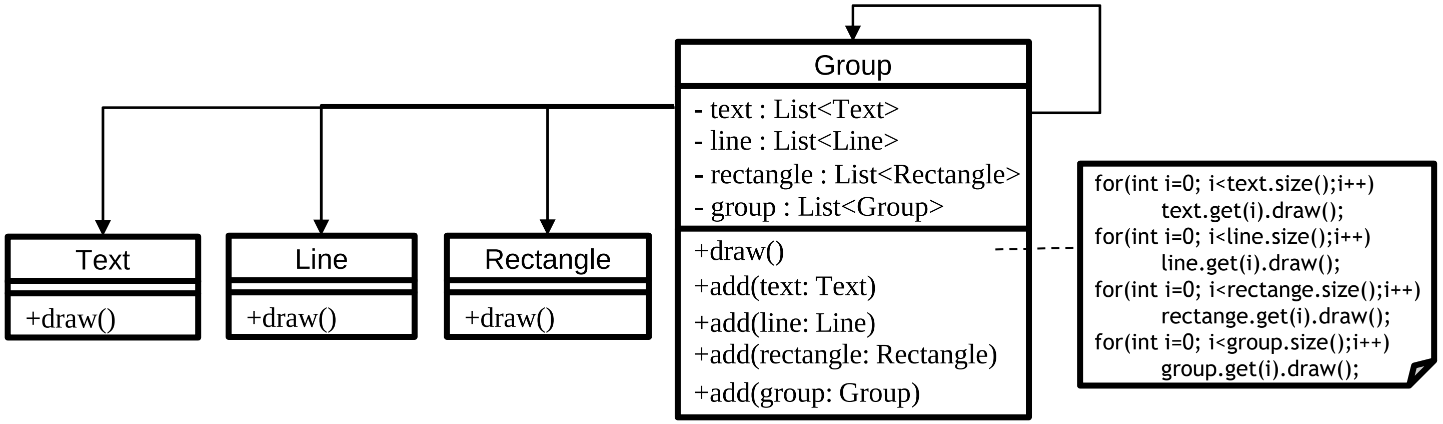
Requirements Statement$_1$

- In schematic capture application, there are some basic components can be drawn such as Text, Line and Rectangle.



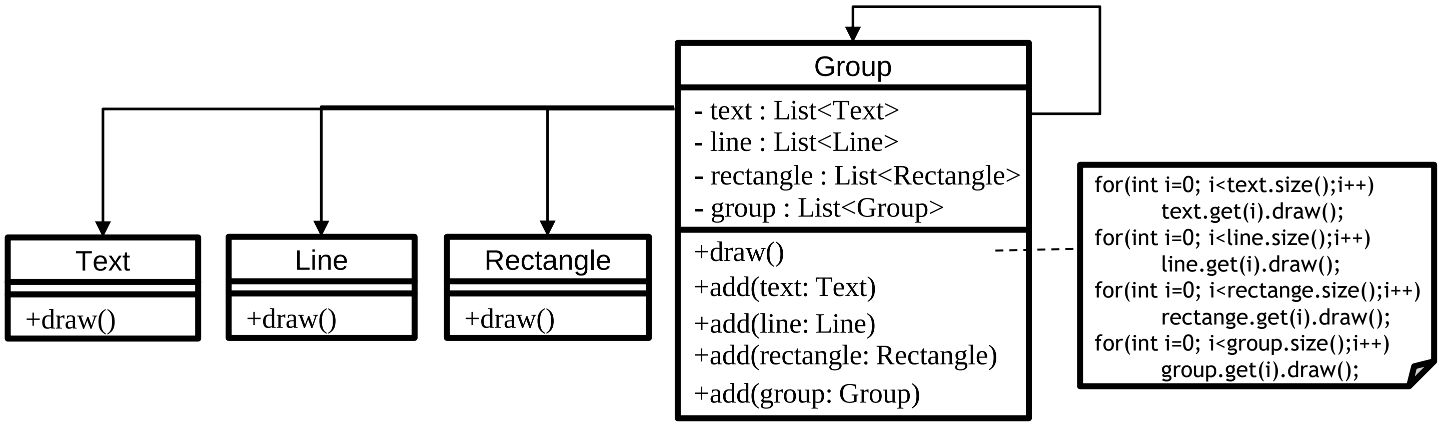
Requirements Statement$_2$

- The user can group basic components to form larger components, which in turn can be grouped to form still larger components.



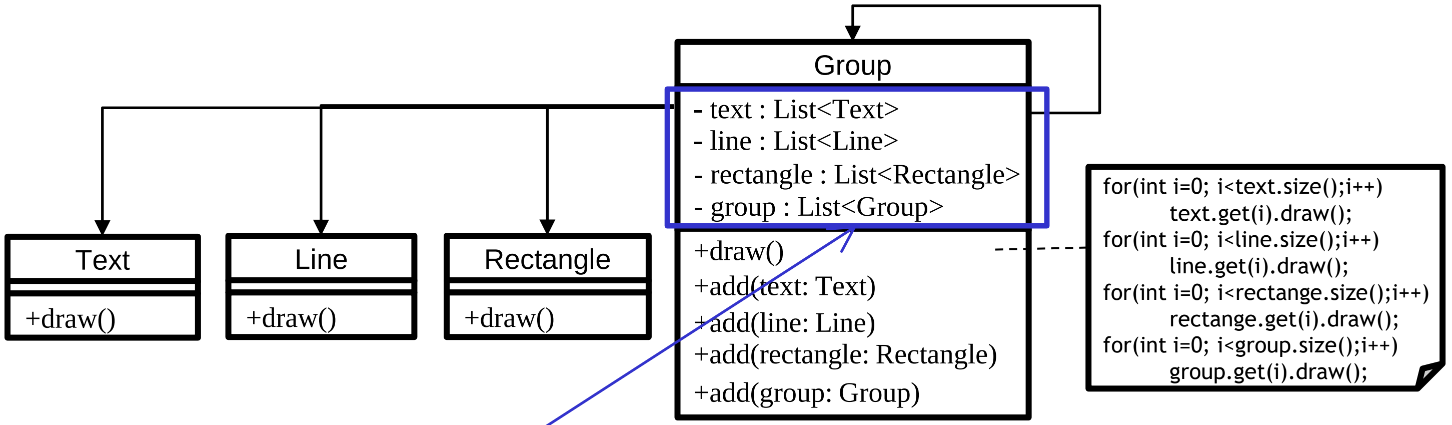
```
for(int i=0; i<text.size();i++)
        text.get(i).draw();
for(int i=0; i<line.size();i++)
        line.get(i).draw();
for(int i=0; i<rectange.size();i++)
        rectange.get(i).draw();
for(int i=0; i<group.size();i++)
        group.get(i).draw();
```

Initial Design



```
for(int i=0; i<text.size();i++)
        text.get(i).draw();
for(int i=0; i<line.size();i++)
        line.get(i).draw();
for(int i=0; i<rectange.size();i++)
        rectange.get(i).draw();
for(int i=0; i<group.size();i++)
        group.get(i).draw();
```

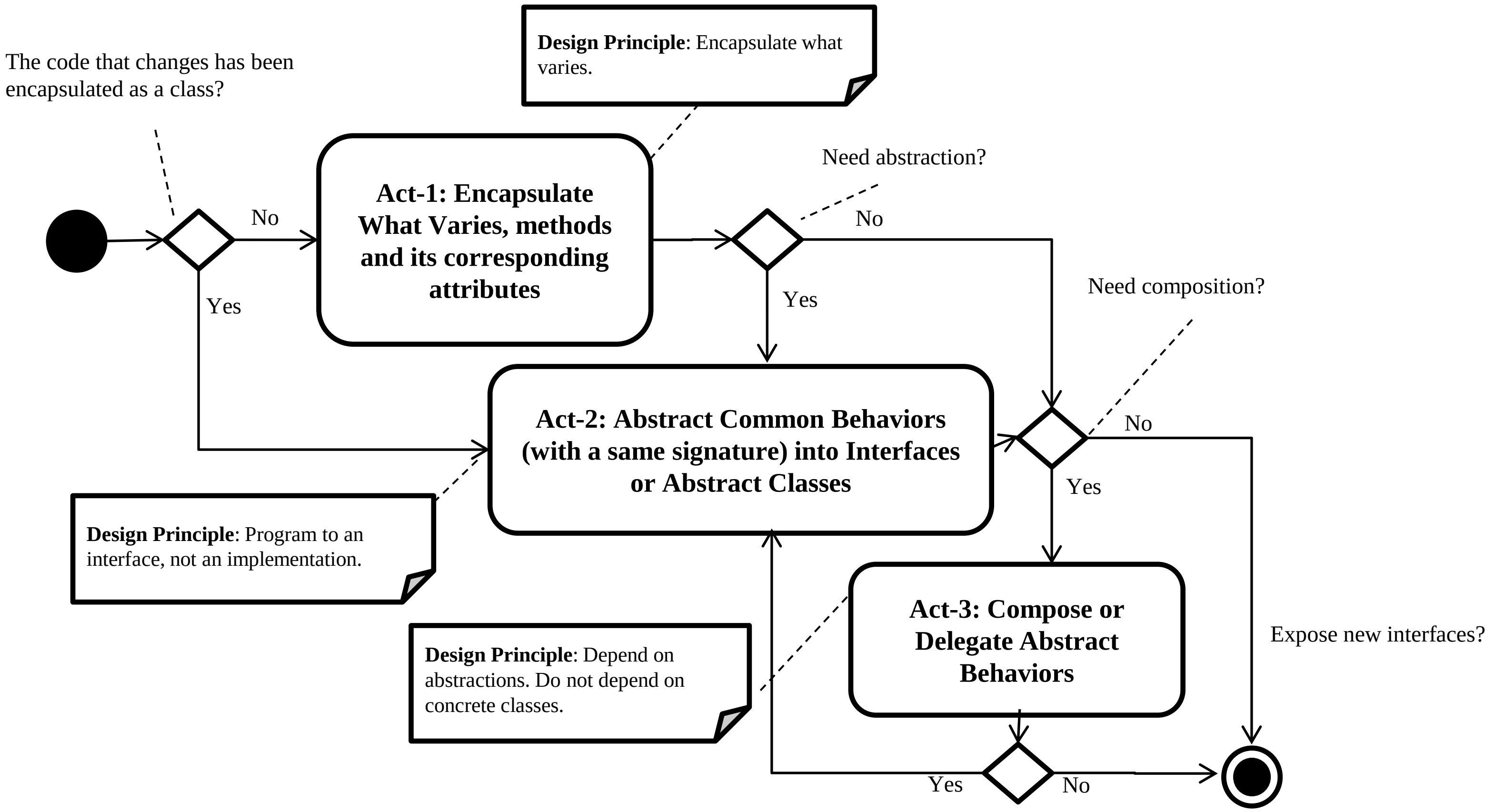
Problems with Initial Design



```
for(int i=0; i<text.size();i++)
        text.get(i).draw();
for(int i=0; i<line.size();i++)
        line.get(i).draw();
for(int i=0; i<rectange.size();i++)
        rectange.get(i).draw();
for(int i=0; i<group.size();i++)
        group.get(i).draw();
```

Problem : If a new requirement is to add a new basic component such as Triangle, then we need to modify Group to meet the new requirement.

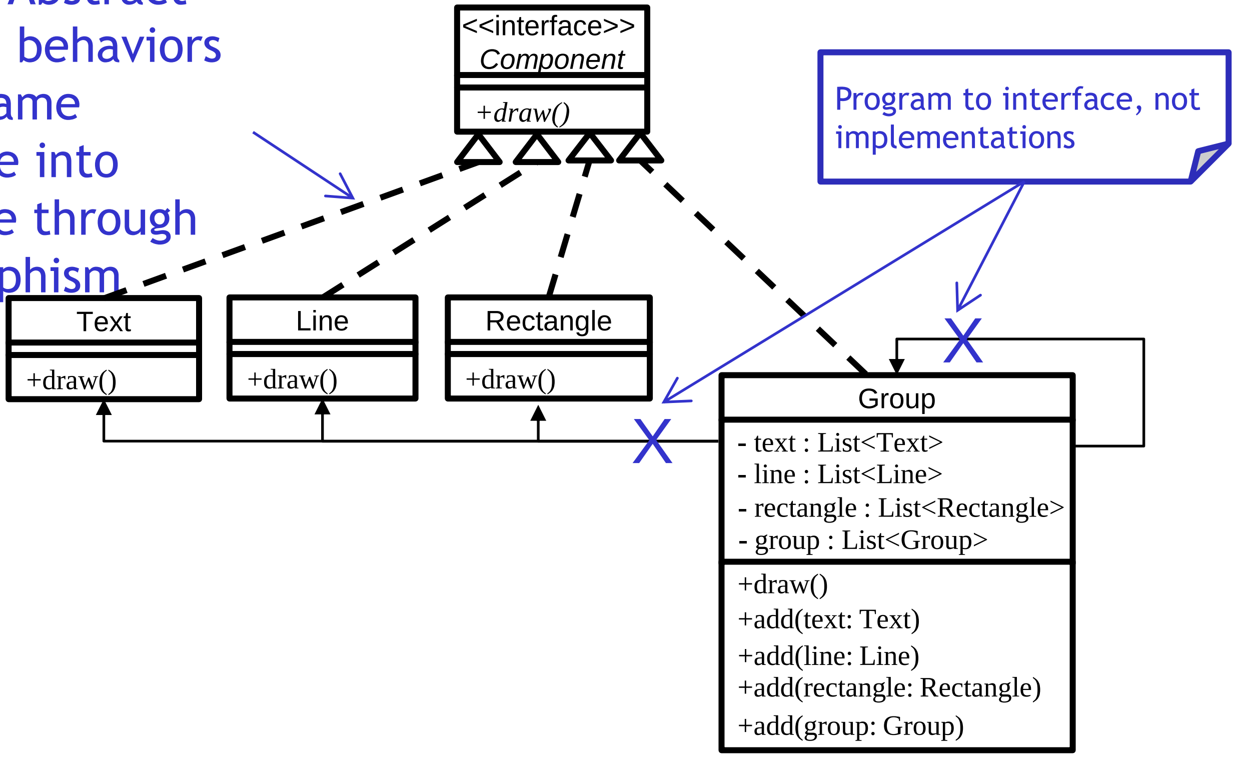
Design Process for Change



The code that changes has been encapsulated as a class?
Design Principle: Encapsulate what varies.
No
Yes
Act-1: Encapsulate What Varies, methods and its corresponding attributes
Need abstraction?
No
Yes
Need composition?
Act-2: Abstract Common Behaviors (with a same signature) into Interfaces or Abstract Classes
No
Yes
Design Principle: Program to an interface, not an implementation.
Act-3: Compose or Delegate Abstract Behaviors
Expose new interfaces?
Design Principle: Depend on abstractions. Do not depend on concrete classes.
Yes
No

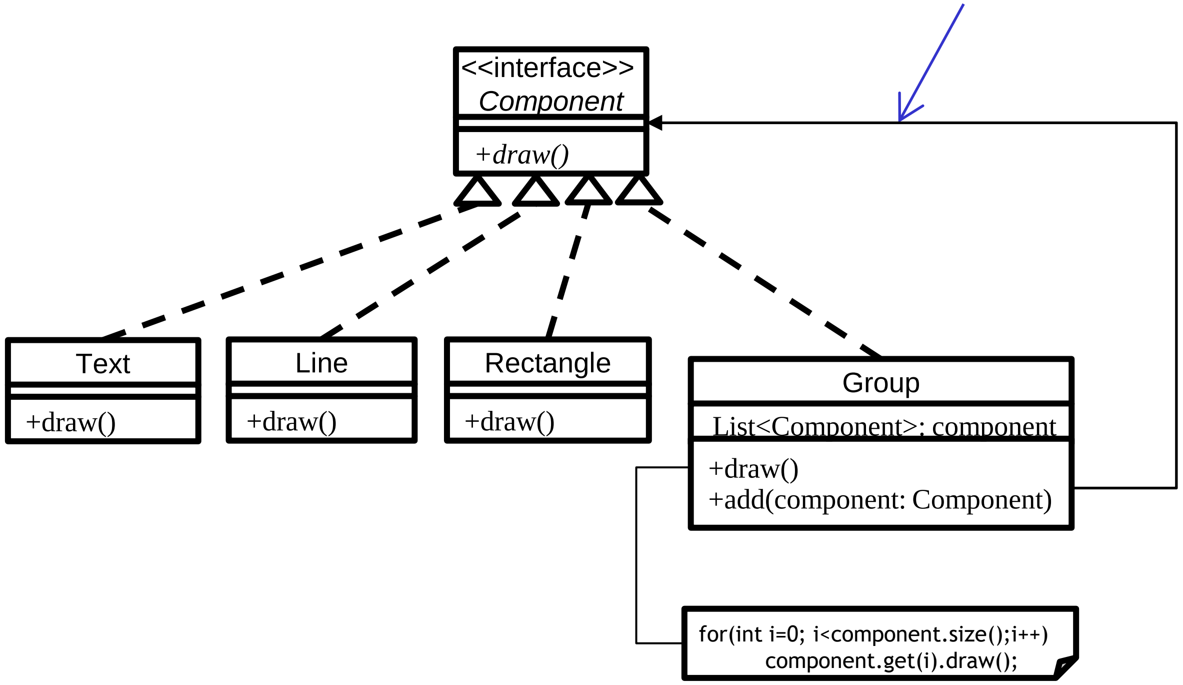
Act-2: Abstract Common Behaviors

Act-2.1: Abstract common behaviors with a same signature into interface through polymorphism



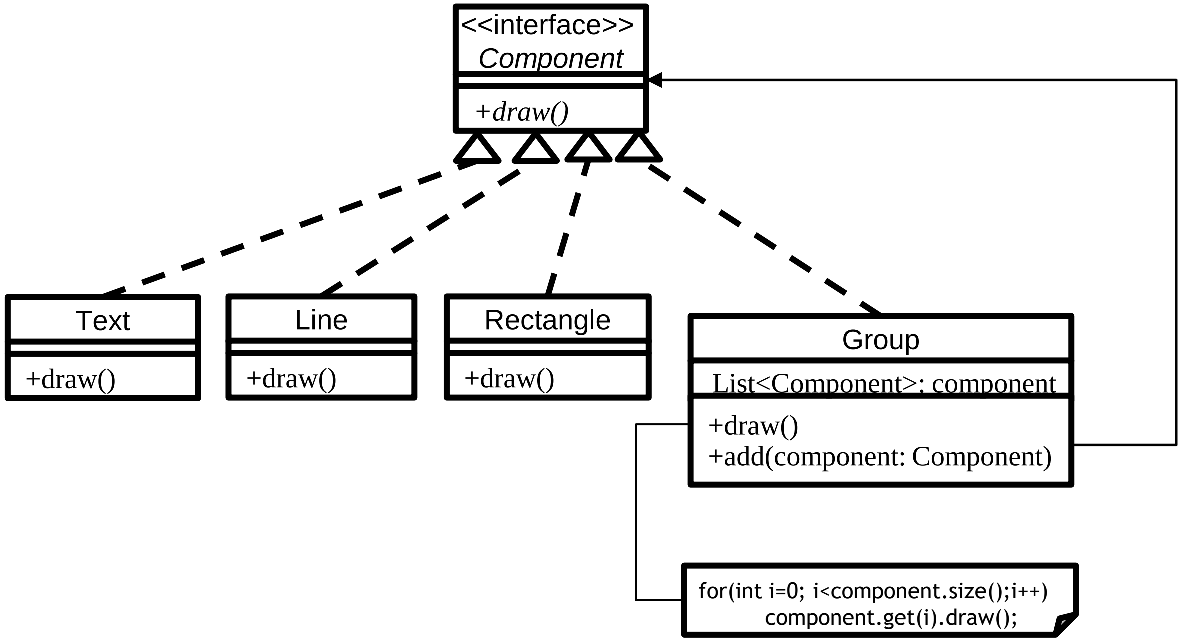
Act-3: Compose Abstract Behaviors

Act-3.1: Compose behaviors of an interface or an abstract class



```
for(int i=0; i<component.size();i++)
        component.get(i).draw();
```

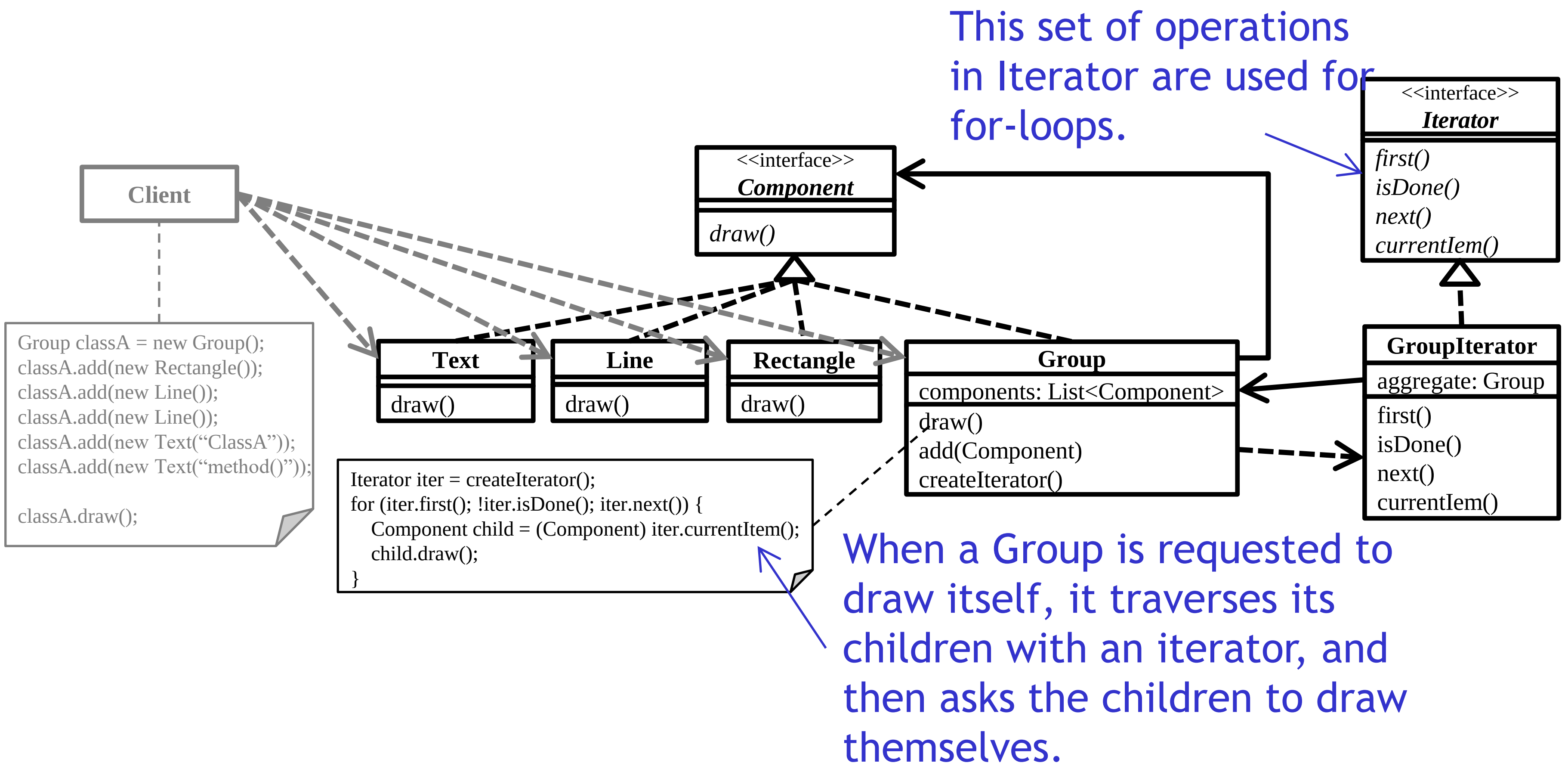
Refactored Design after Design Process



```
for(int i=0; i<component.size();i++)
        component.get(i).draw();
```

Draw Composite Objects with Iterator

- A class notation in a class diagram which is a Group that composes two Lines, one Rectangle, and several Texts is drawn on the screen.



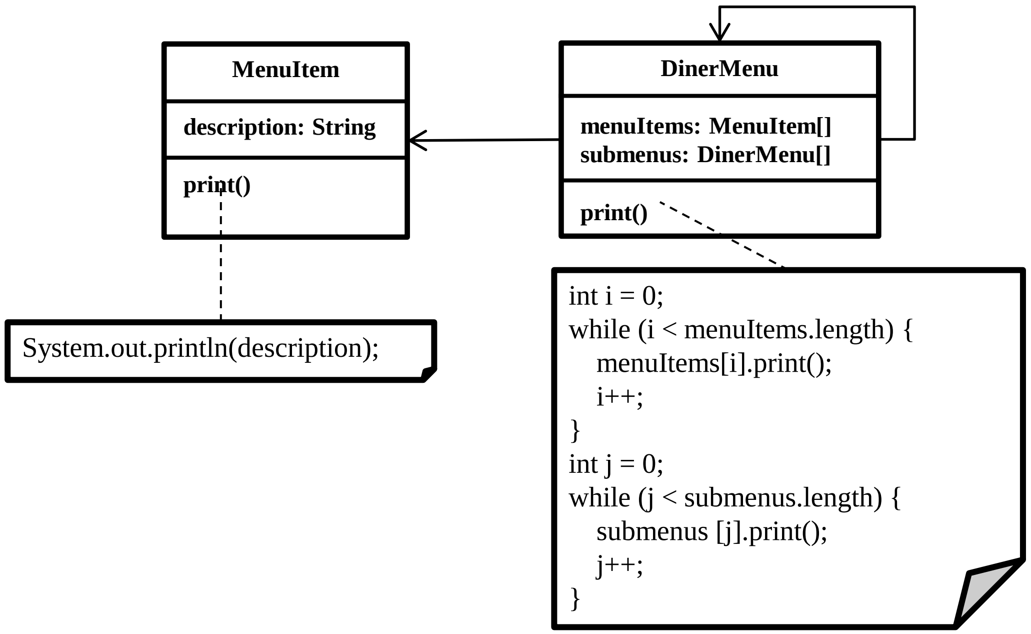
```
Group classA = new Group();
classA.add(new Rectangle());
classA.add(new Line());
classA.add(new Line());
classA.add(new Text(“ClassA”));
classA.add(new Text(“method()”));

classA.draw();
```

```
Iterator iter = createIterator();
for (iter.first(); !iter.isDone(); iter.next()) {
   Component child = (Component) iter.currentItem();
   child.draw();
}
```

Merge of Two Menus (Extended)

Requirements Statement

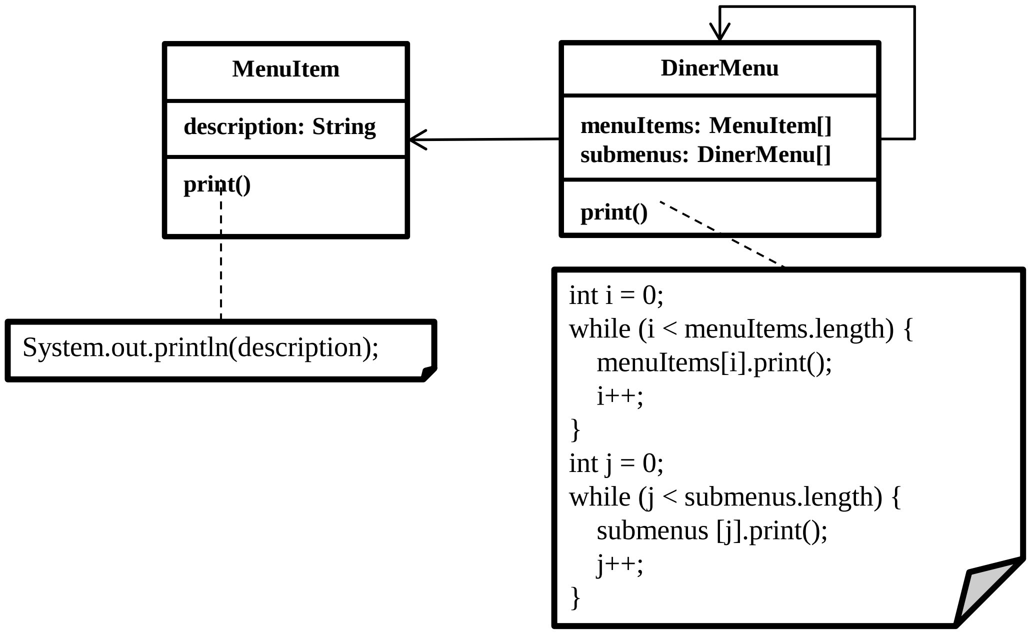
- Based on the Merge Two Menus example last week,
  - A dessert submenu is going to be added to the Diner menu.



```
System.out.println(description);
```

```
int i = 0;
while (i < menuItems.length) {
   menuItems[i].print();
   i++;
}
int j = 0;
while (j < submenus.length) {
   submenus [j].print();
   j++;
}
```

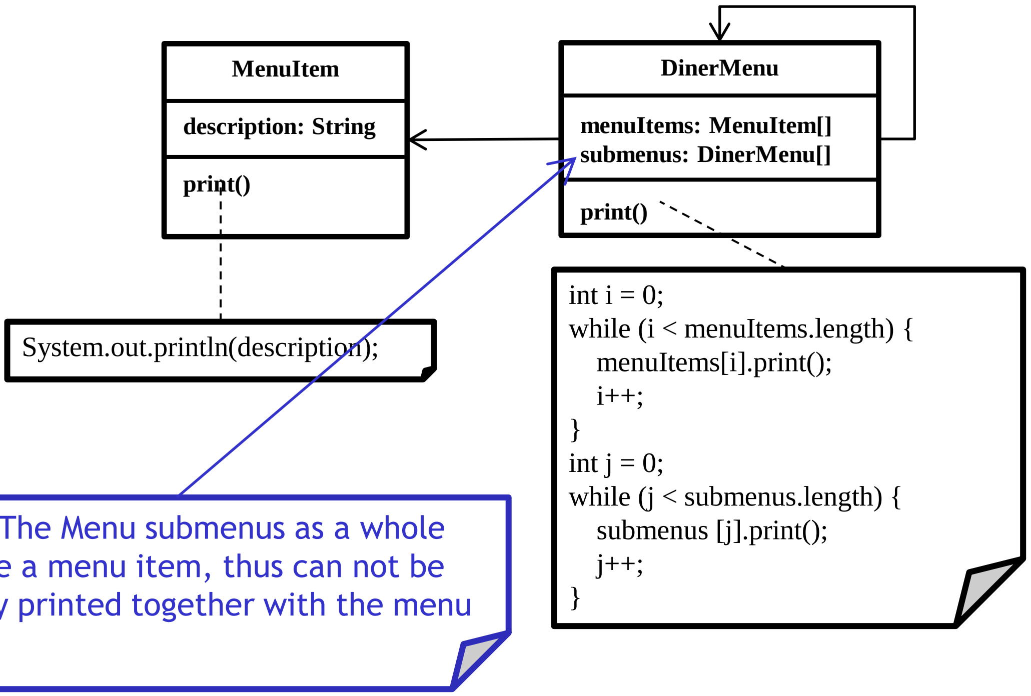
Initial Design - Class Diagram



```
System.out.println(description);
```

```
int i = 0;
while (i < menuItems.length) {
   menuItems[i].print();
   i++;
}
int j = 0;
while (j < submenus.length) {
   submenus [j].print();
   j++;
}
```

Problems with Initial Design

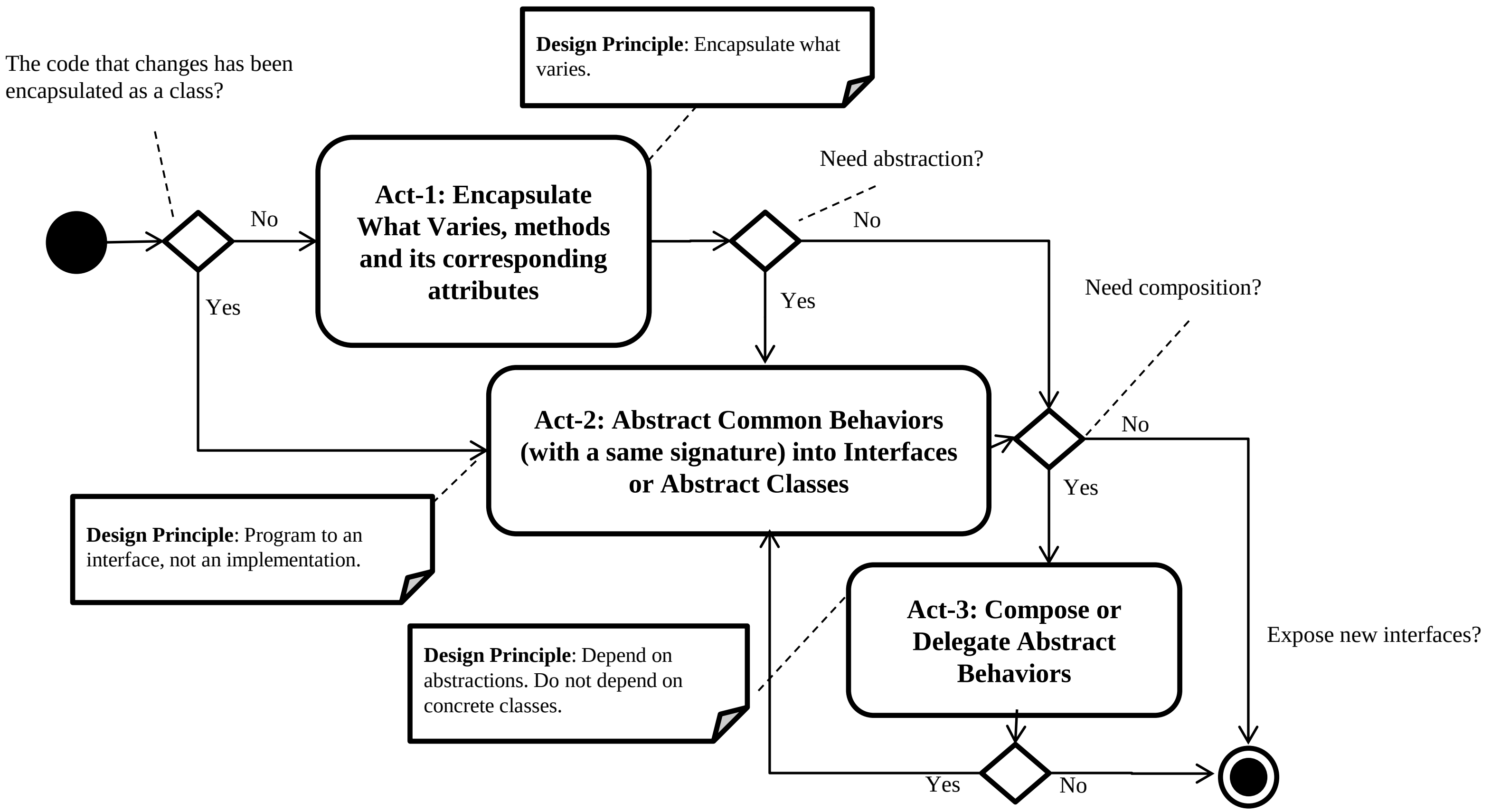


```
System.out.println(description);
```

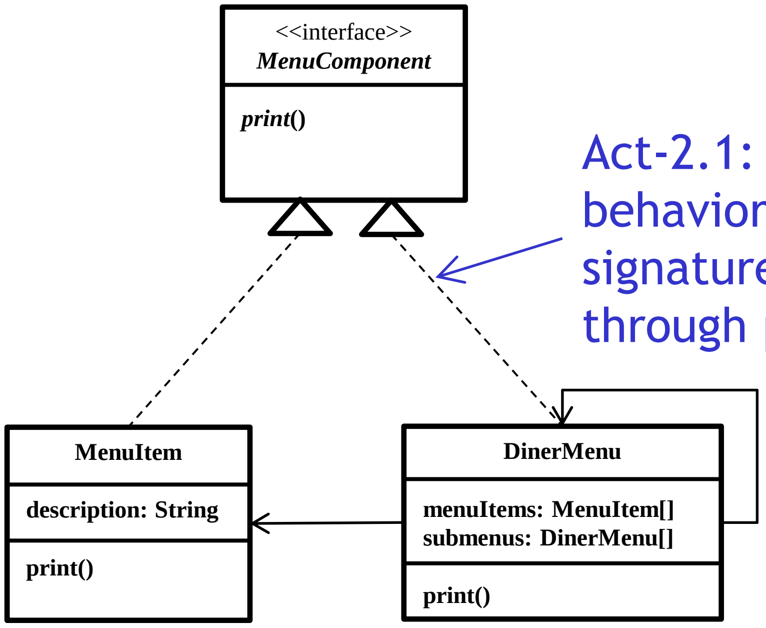
```
int i = 0;
while (i < menuItems.length) {
   menuItems[i].print();
   i++;
}
int j = 0;
while (j < submenus.length) {
   submenus [j].print();
   j++;
}
```

Problem: The Menu submenus as a whole can not be a menu item, thus can not be iteratively printed together with the menu items.

Design Process for Change

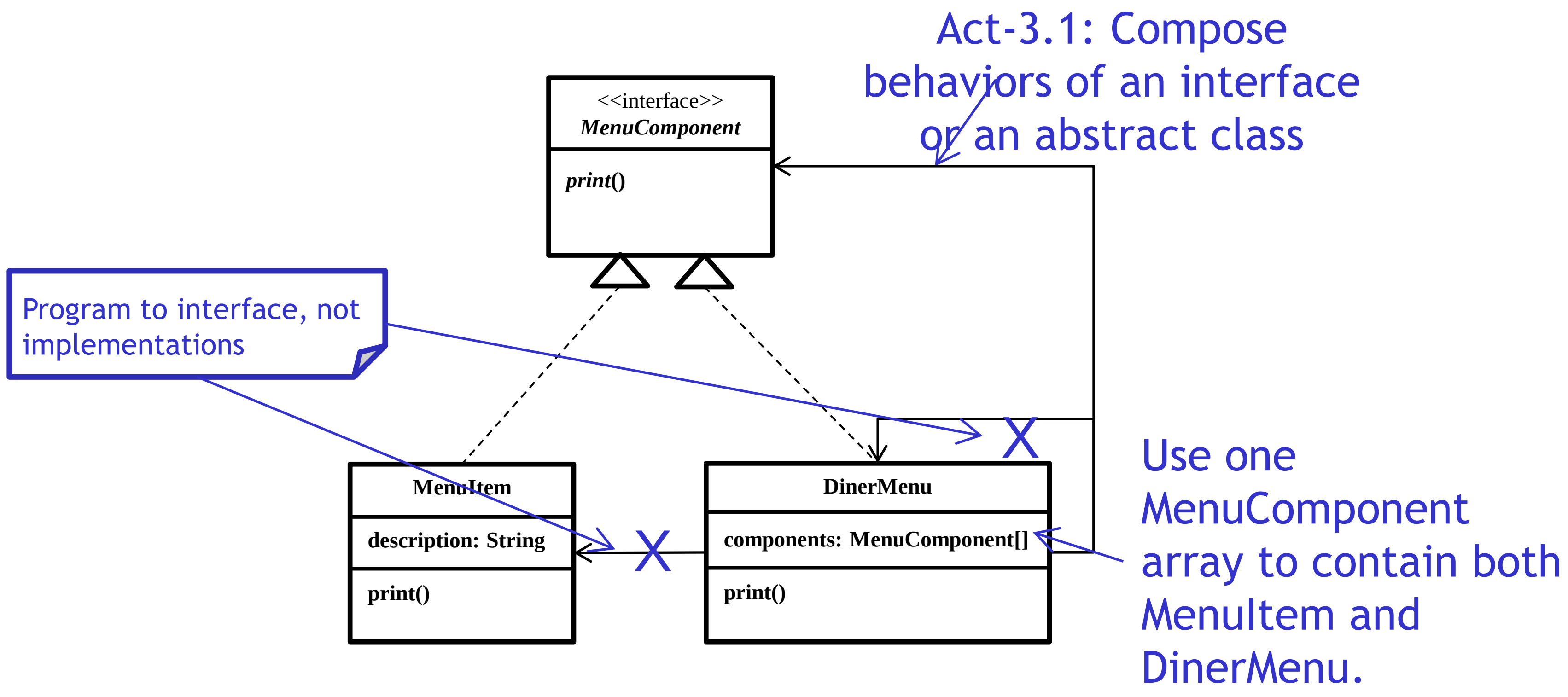

Design Principle: Encapsulate what varies.
The code that changes has been encapsulated as a class?
No
Yes
Act-1: Encapsulate What Varies, methods and its corresponding attributes
Need abstraction?
No
Yes
Need composition?
Act-2: Abstract Common Behaviors (with a same signature) into Interfaces or Abstract Classes
No
Yes
Design Principle: Program to an interface, not an implementation.
Act-3: Compose or Delegate Abstract Behaviors
Design Principle: Depend on abstractions. Do not depend on concrete classes.
Expose new interfaces?
Yes
No

Act-2: Abstract Common Behaviors

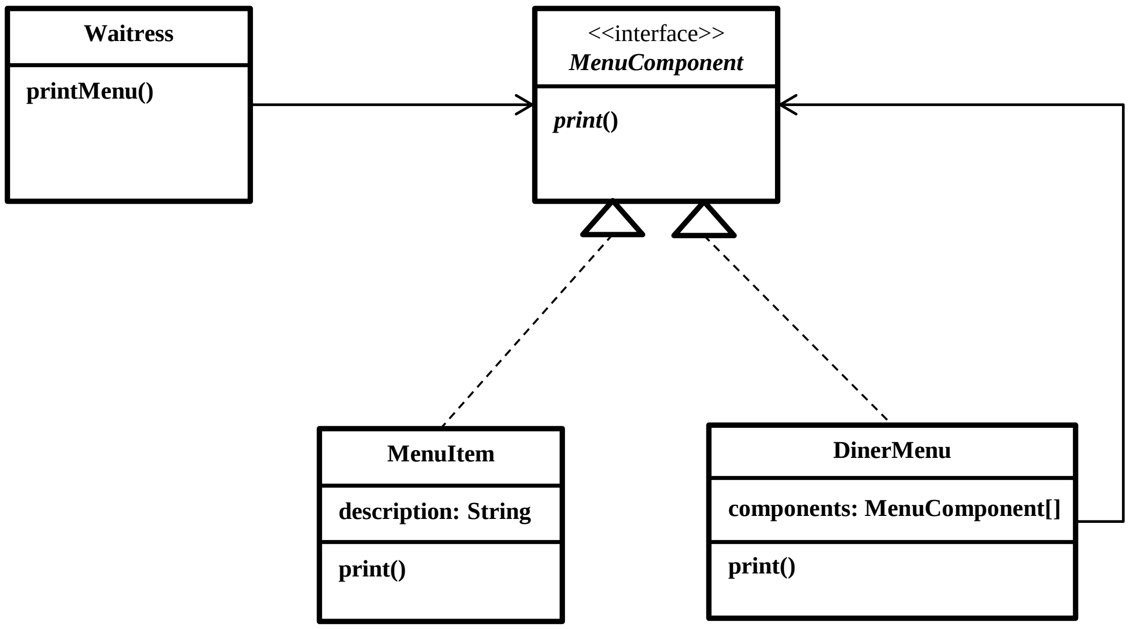


Act-2.1: Abstract common behaviors with a same signature into interface through polymorphism

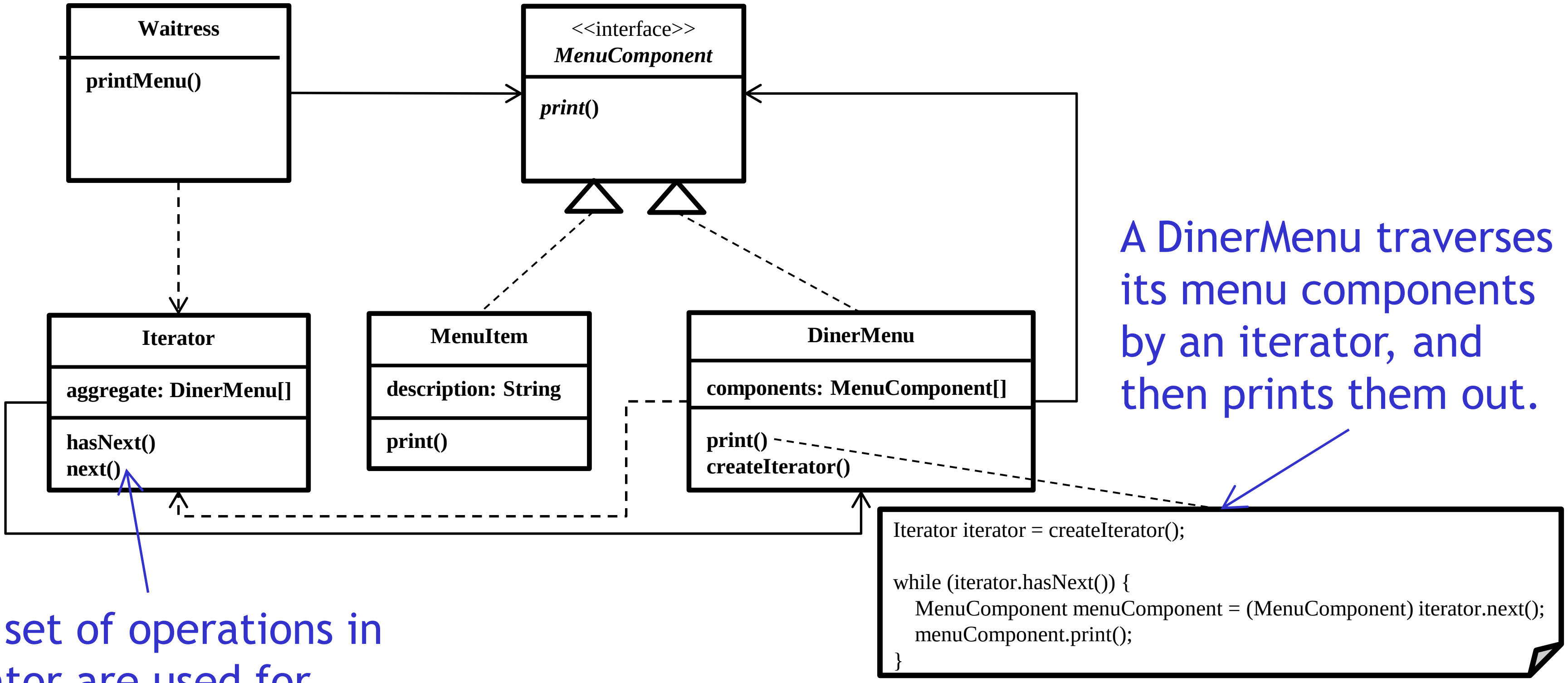
Act-3: Compose Abstract Behaviors


Act-3.1: Compose behaviors of an interface or an abstract class
<<interface>>
MenuComponent
print()
Program to interface, not implementations
X
X
MenuItem
description: String
print()
DinerMenu
components: MenuComponent[]
print()
Use one MenuComponent array to contain both MenuItem and DinerMenu.

Refactored Design after Design Process



Print Menus in Composite with Iterator



```
Iterator iterator = createIterator();

while (iterator.hasNext()) {
    MenuComponent menuComponent = (MenuComponent) iterator.next();
    menuComponent.print();
}
```

A DinerMenu traverses its menu components by an iterator, and then prints them out.

This set of operations in Iterator are used for while-loops.

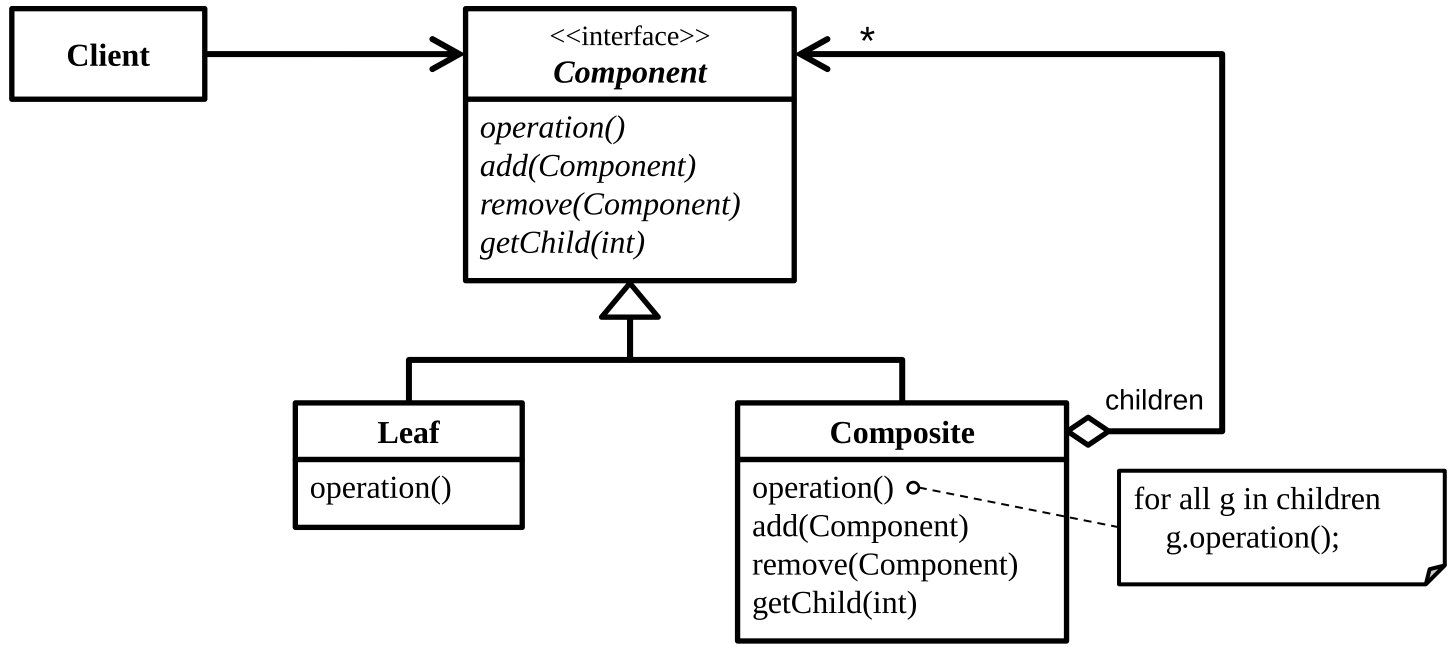
Recurrent Problem

- The user can group components to form larger components, which in turn can be grouped to form still larger components.
  - A simple implementation could define classes for primitives that act as containers for these primitives.
  - But there's a problem with this approach: Code that uses these classes must treat primitive and container objects differently, even if most of the time the user treats them identically.

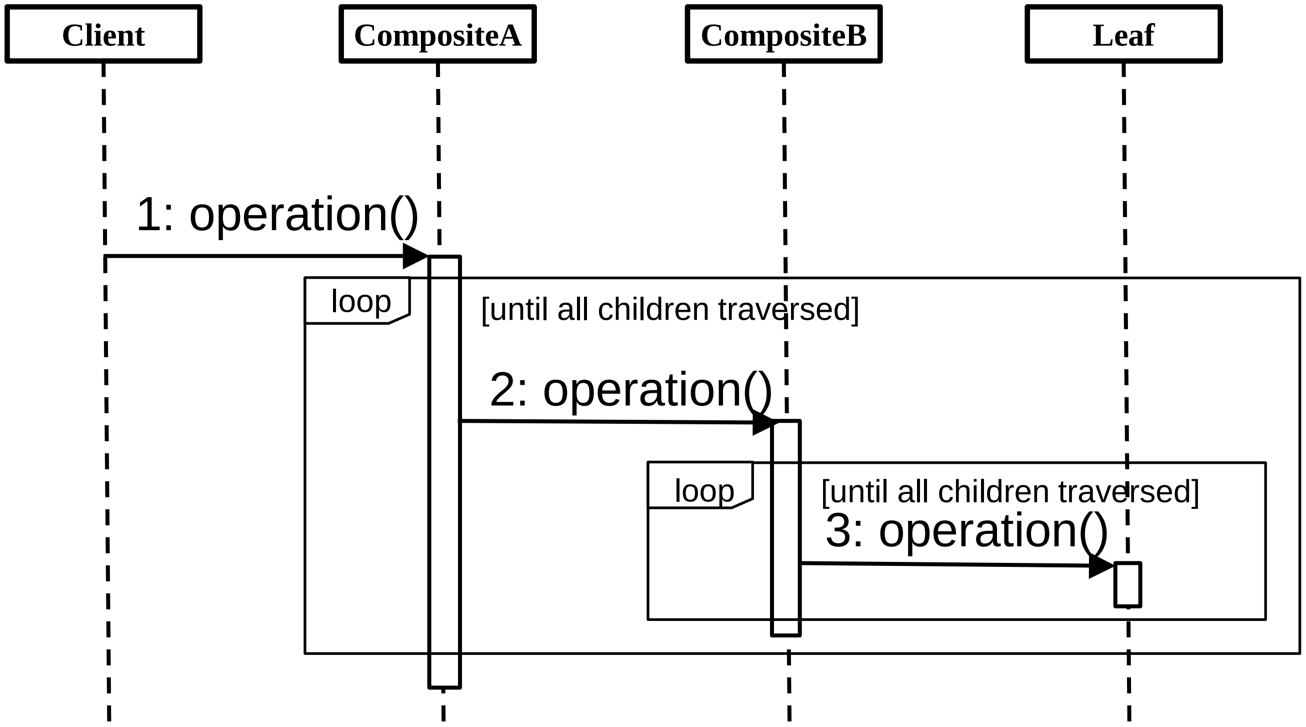
Intent

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

Composite Pattern Structure$_1$



Composite Pattern Structure$_2$



Composite Pattern Structure$_3$

	Instantiation	**Use**	**Termination**
Component	X	Client uses this interface to manipulate a Composite class or a Leaf class.	X
Composite	Don’t Care	Client adds, removes, and gets Composite or Leaf objects through Composite who acts as a container. When Client invokes Composite’s operation method, Composite invokes the same method of its child Component objects iteratively.	Don’t Care
Leaf	Don’t Care	Client adds, removes, and gets Leaf objects to/from Composite. Leaf executes its operation method when Composite or Client requests through polymorphism.	Don’t Care