Visitor pattern

Introduction to Visitor

Applying Visitor to the text's example of Composite design pattern

Typical sequence diagrams

Example code

General application of Visitor … many visitors

Introduction to Visitor

The visitor pattern allows you to define new operations for an object structure without changing the object structure.

- Although … some method infrastructure is needed
- We'll consider the application of ***Visitor*** to ***Composite***

How:

- two hierarchies: one for the object structure and one for the visitors
- In the visitor hierarchy create a subclass for each new operation

- A node to be visited receives an *accept(visitor)* message
- The node sends a *visit(this)* message to a visitor

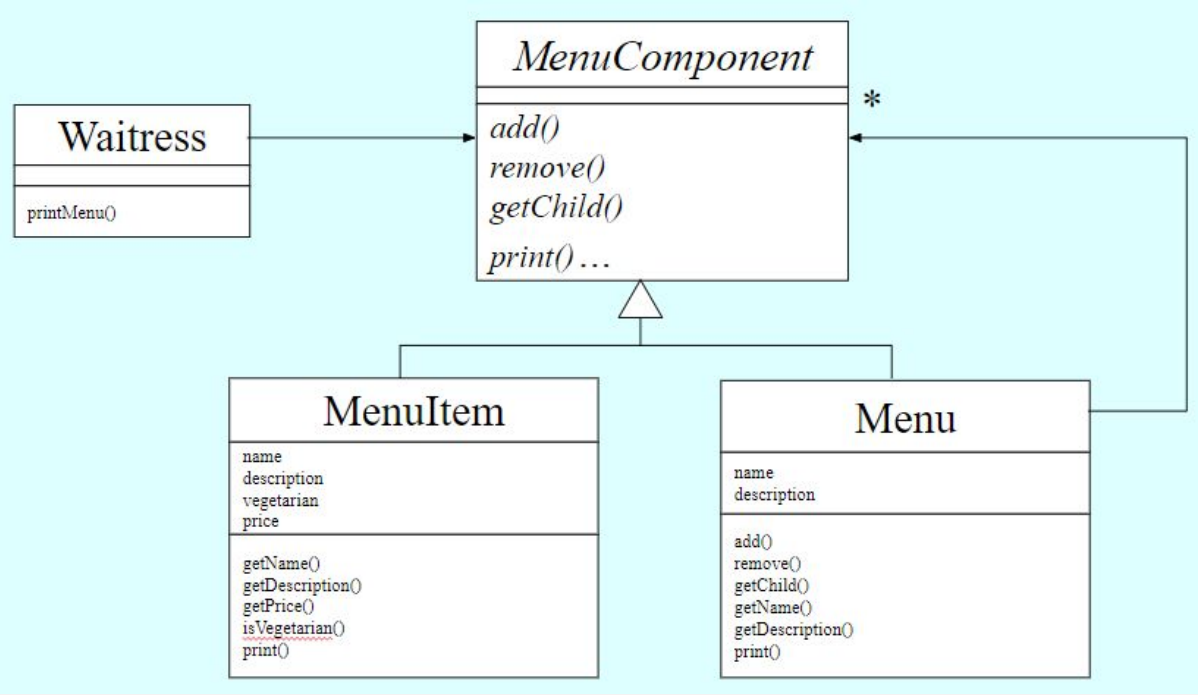
Composite Pattern – Text example

Consider our text's example of the Composite design pattern. The UML class diagram is shown below.

The UML classes representing the menu are arranged in two levels:

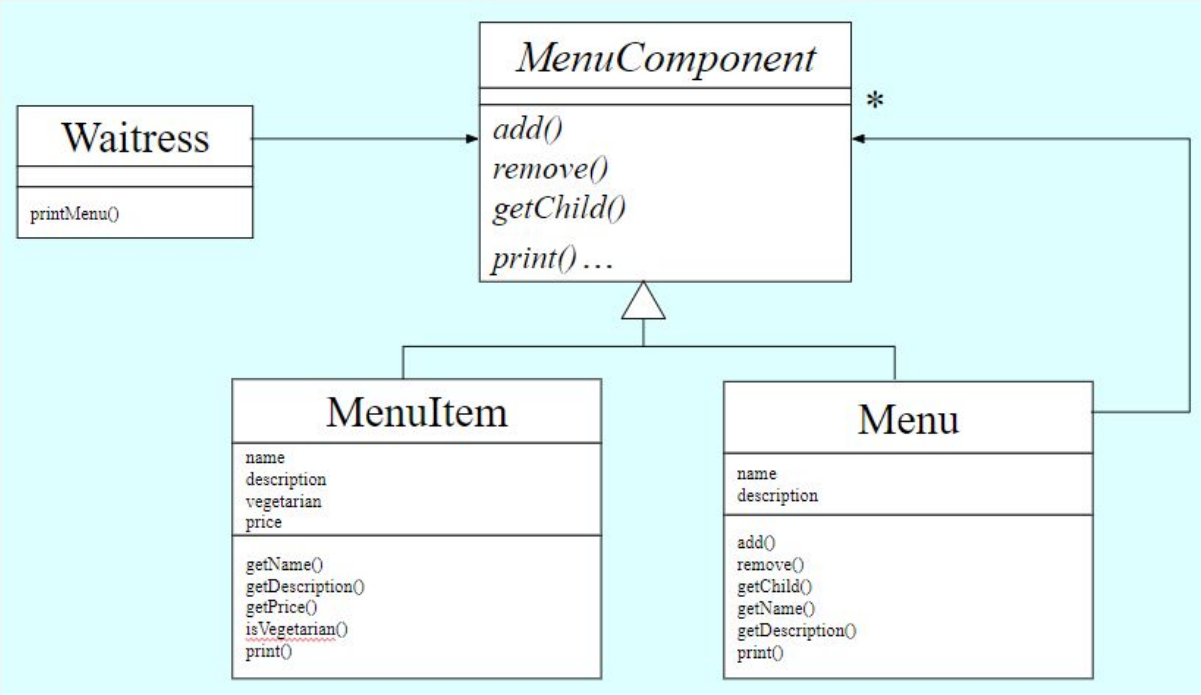
MenuComponent is the superclass

two subclasses MenuItem and Menu.



Composite Pattern – Text example

Recall the print methods in the subclasses - these are used to display the menu on the standard output device.



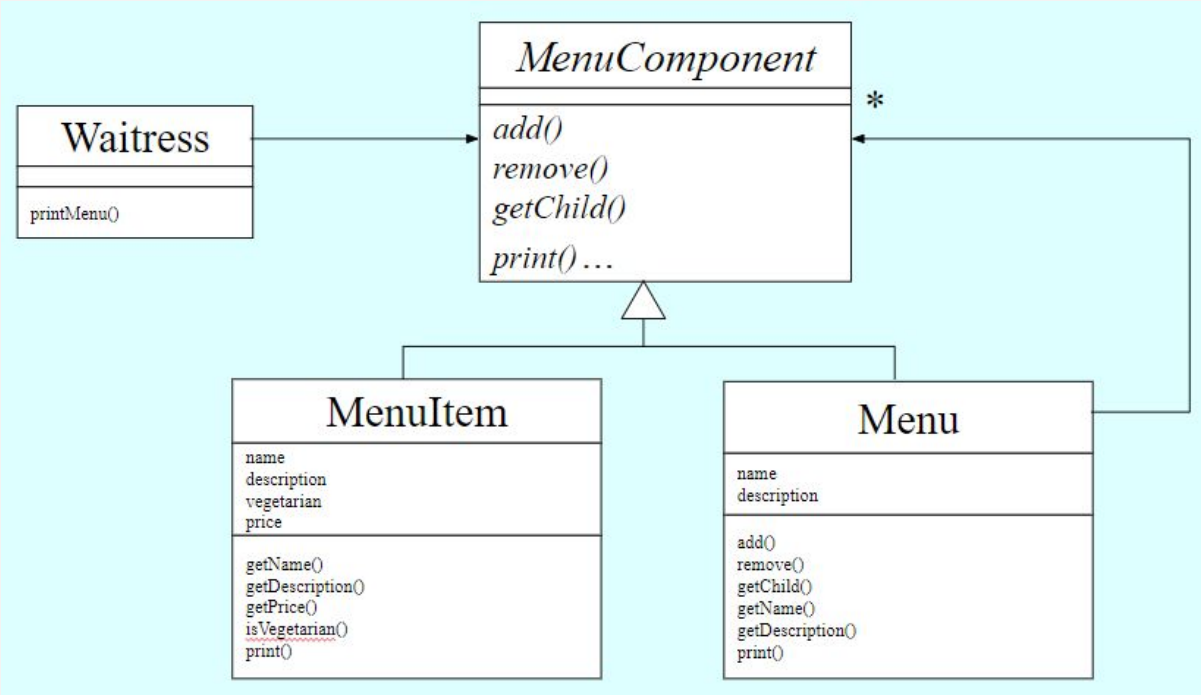
Now, suppose a **new requirement** comes along … should we modify this code that's working, or ...

Composite Pattern – Text example

Now, suppose there is a new requirement, to generate HTML so the menu can be displayed in a web page.

We could create more methods in this hierarchy to accomplish this new output. This means we would modify existing classes - classes that are working fine.

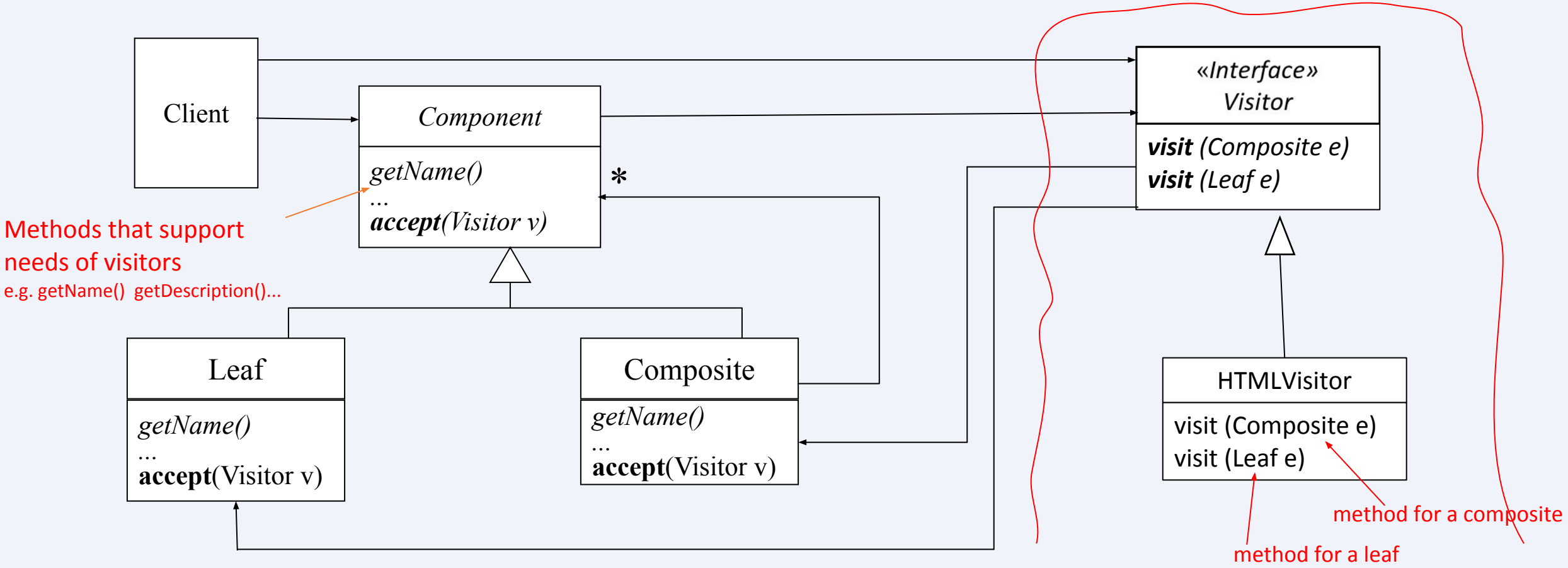
Sometimes modifying code is risky - as systems get more and more complex we run the risk of breaking existing code, and if we change one thing we should do exhaustive testing to ensure what was working before is still working.



An **alternative** would be to incorporate the Visitor design pattern where typically we add a visit method to MenuItem and to Menu … see next slide

Applying Visitor to Text Example

A visitor will visit the menu and generate HTML



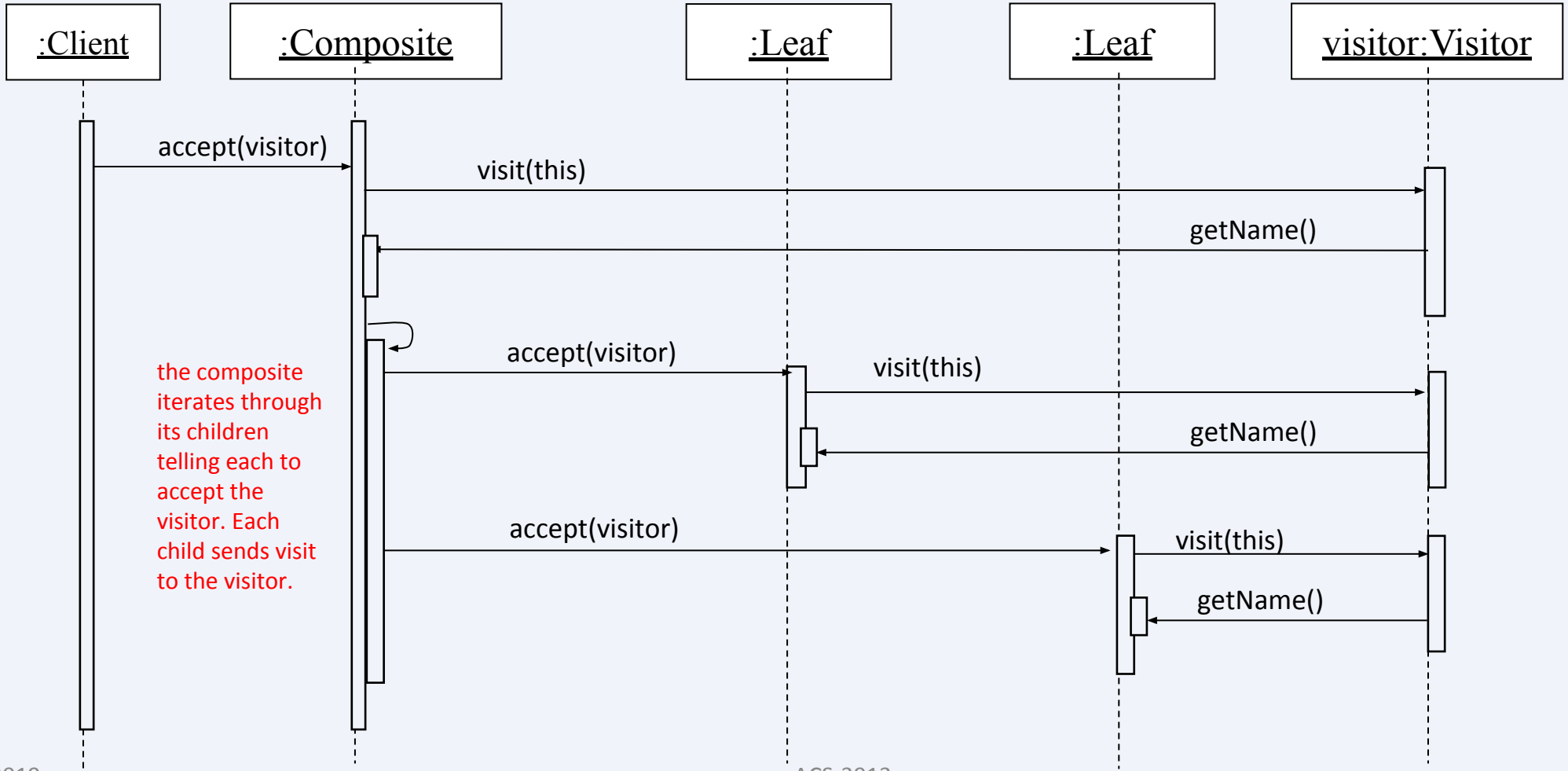
Component
Each component defines an *accept* method that takes a visitor as an argument

Visitor hierarchy
Each visitor subclass defines a *visit* method that takes a concrete component as an argument

Sample Sequence Diagram for collaboration at runtime

First scenario: Suppose we have a component hierarchy that has one composite with two leafs.

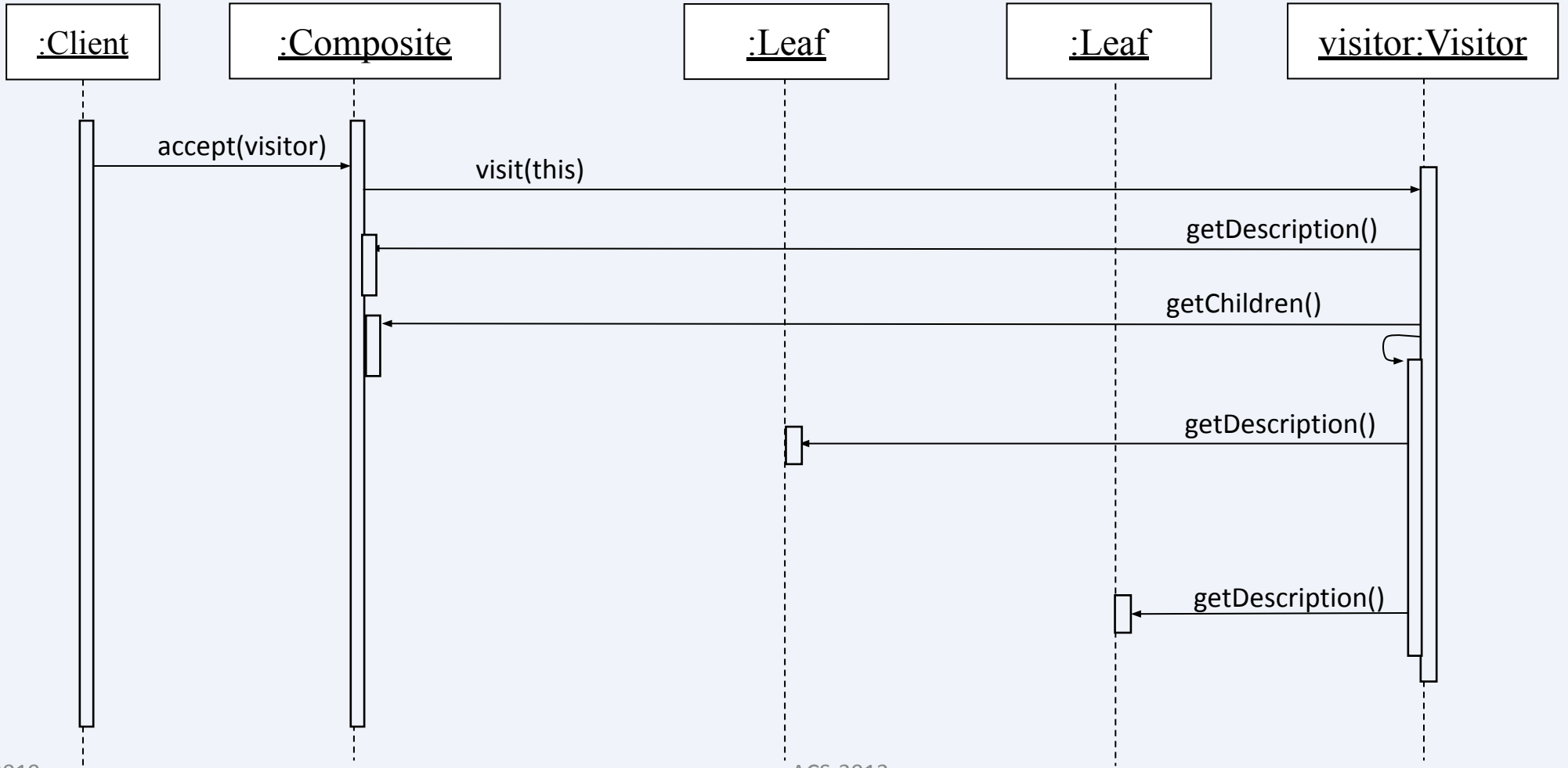
1. the client sends accept(visitor) to the composite object.
2. the composite object sends visit(this) to the visitor, and the visitor responds by sending some operation (for example, getName()) to the composite object
3. the composite iterates through its children ...



Sample Sequence Diagram for collaboration at runtime

Second scenario: Suppose we have a component hierarchy that has one composite with two leafs.

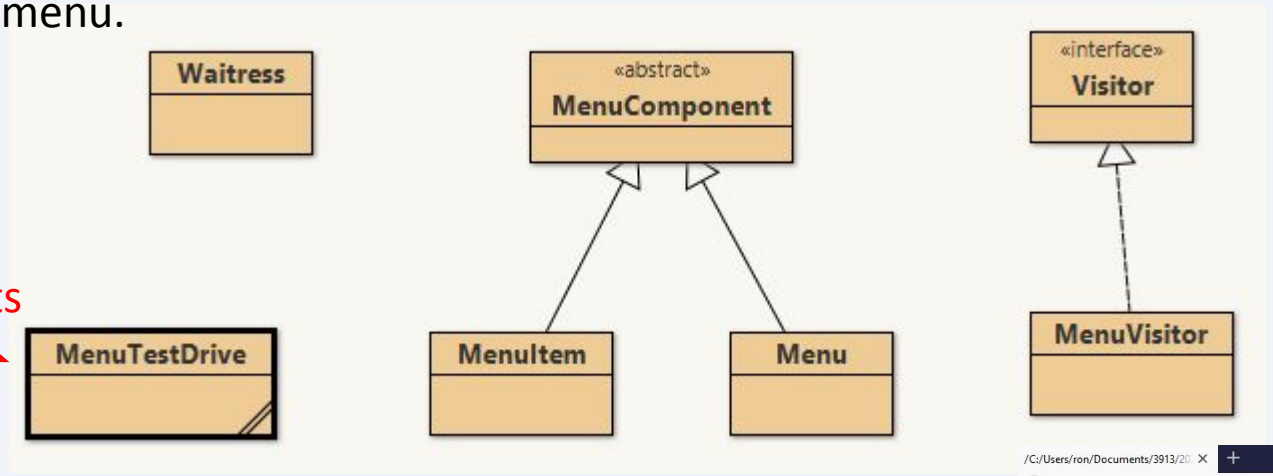
1. the client sends accept(visitor) to the composite object.
2. the composite object sends visit(this) to the visitor, and the visitor responds by sending , say, getDescription() to the composite object, and then getChildren() to the composite
3. the visitor sends getDescription to each child object



Example Code for Applying Visitor Applied to Text Example

Example code: see the course web page. Examine the sample code that involves a visitor for producing HTML output for the menu.

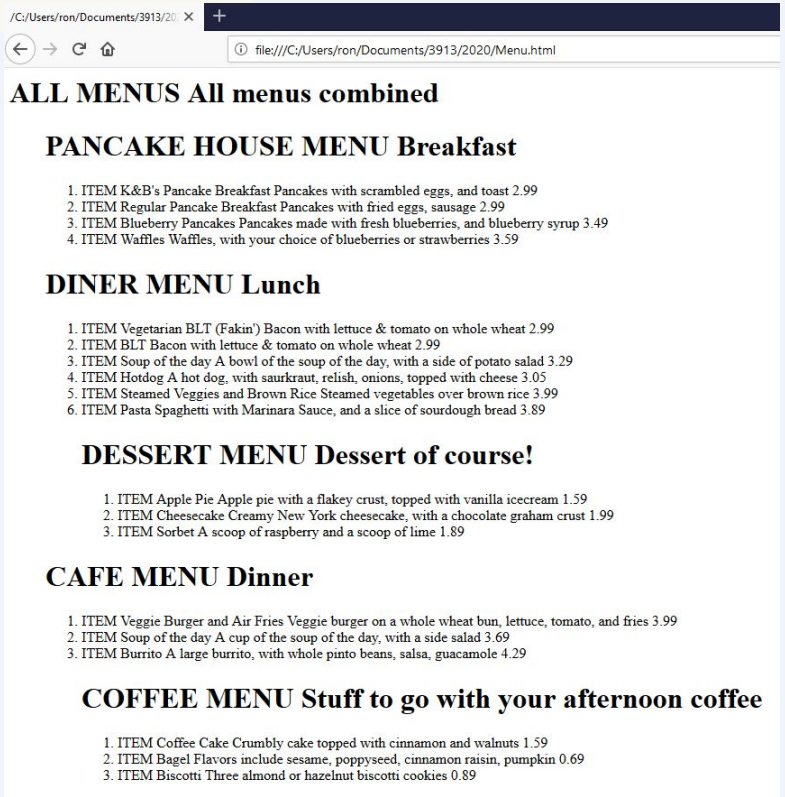
see the last statements in main()



Sample output :

```
<h1>ALL MENUS All menus combined</h1>
<ol>
<h1>PANCAKE HOUSE MENU Breakfast</h1>
<ol>
<li>ITEM K&B's Pancake Breakfast Pancakes with
scrambled eggs, and toast 2.99</li>
<li>ITEM Regular Pancake Breakfast Pancakes with
fried eggs, sausage 2.99</li>
<li>ITEM Blueberry Pancakes Pancakes made with
fresh blueberries, and blueberry syrup 3.49</li>
<li>ITEM Waffles Waffles, with your choice of
blueberries or strawberries 3.59</li>
</ol>
<h1>DINER MENU Lunch</h1>
<ol>
<li>ITEM Vegetarian BLT (Fakin') Bacon with lettuce &
tomato on whole wheat 2.99</li>
……..
```

viewed in browser



file:///C:/Users/ron/Documents/3913/2020/Menu.html

ALL MENUS All menus combined

PANCAKE HOUSE MENU Breakfast

1. ITEM K&B's Pancake Breakfast Pancakes with scrambled eggs, and toast 2.99
2. ITEM Regular Pancake Breakfast Pancakes with fried eggs, sausage 2.99
3. ITEM Blueberry Pancakes Pancakes made with fresh blueberries, and blueberry syrup 3.49
4. ITEM Waffles Waffles, with your choice of blueberries or strawberries 3.59

DINER MENU Lunch

1. ITEM Vegetarian BLT (Fakin') Bacon with lettuce & tomato on whole wheat 2.99
2. ITEM BLT Bacon with lettuce & tomato on whole wheat 2.99
3. ITEM Soup of the day A bowl of the soup of the day, with a side of potato salad 3.29
4. ITEM Hotdog A hot dog, with saurkraut, relish, onions, topped with cheese 3.05
5. ITEM Steamed Veggies and Brown Rice Steamed vegetables over brown rice 3.99
6. ITEM Pasta Spaghetti with Marinara Sauce, and a slice of sourdough bread 3.89

DESSERT MENU Dessert of course!

1. ITEM Apple Pie Apple pie with a flakey crust, topped with vanilla icecream 1.59
2. ITEM Cheesecake Creamy New York cheesecake, with a chocolate graham crust 1.99
3. ITEM Sorbet A scoop of raspberry and a scoop of lime 1.89

CAFE MENU Dinner

1. ITEM Veggie Burger and Air Fries Veggie burger on a whole wheat bun, lettuce, tomato, and fries 3.99
2. ITEM Soup of the day A cup of the soup of the day, with a side salad 3.69
3. ITEM Burrito A large burrito, with whole pinto beans, salsa, guacamole 4.29

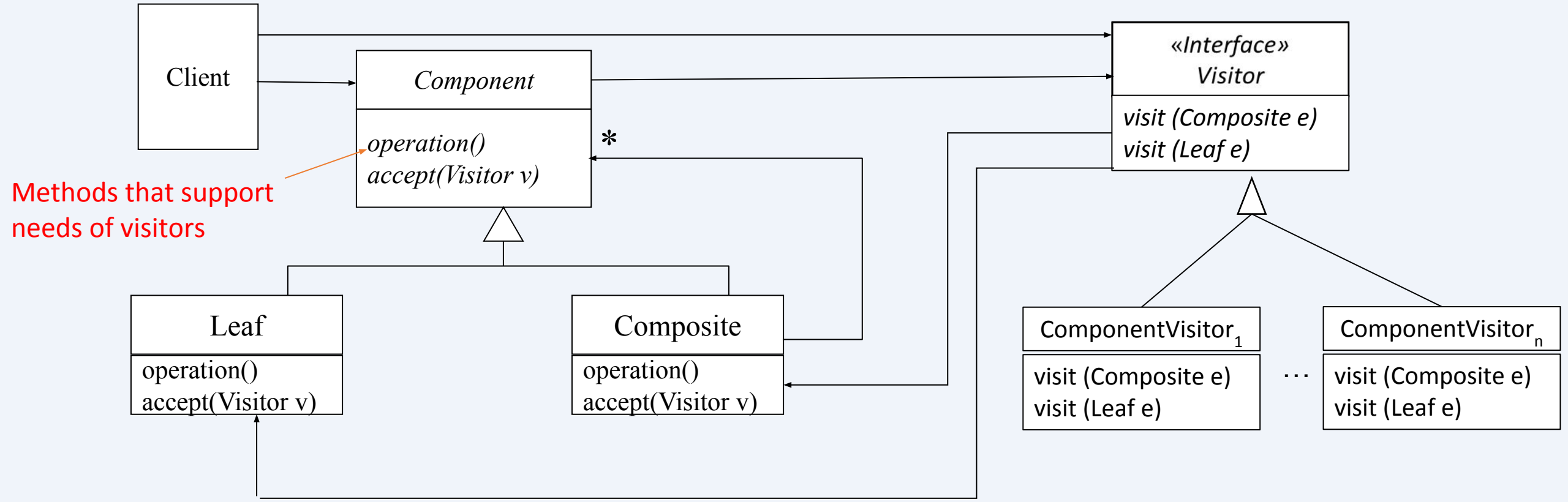
COFFEE MENU Stuff to go with your afternoon coffee

1. ITEM Coffee Cake Crumbly cake topped with cinnamon and walnuts 1.59
2. ITEM Bagel Flavors include sesame, poppyseed, cinnamon raisin, pumpkin 0.69
3. ITEM Biscotti Three almond or hazelnut biscotti cookies 0.89

In General …. Visitor applied to Composite

Many concrete visitors can be defined for the Composite, one for each purpose

…. see below where we have ComponentVisitor$_1$ … ComponentVisitor$_n$



Component

Each component defines an *accept* method that takes a visitor as an argument

Visitor

Each visitor defines a *visit* method that takes a concrete component as an argument (typically one method for each component class)