Implementing the Menu Component

Okay, we're going to start with the MenuComponent abstract class; remember, the role of the menu component is to provide an interface for the leaf nodes and the composite nodes. Now you might be asking, "Isn't the MenuComponent playing two roles?" It might well be and we'll come back to that point. However, for now we're going to provide a default implementation of the methods so that if the MenuItem (the leaf) or the Menu (the composite) doesn't want to implement some of the methods (like getChild() for a leaf node) they can fall back on some basic behavior:

All components must implement the MenuComponent interface; however, because leaves and nodes have different roles we can't always define a default implementation for each method that makes sense. Sometimes the best you can do is throw a runtime exception.

MenuComponent provides default implementations for every method.

```
public abstract class MenuComponent {

    public void add(MenuComponent menuComponent) {
        throw new UnsupportedOperationException();
    }
    public void remove(MenuComponent menuComponent) {
        throw new UnsupportedOperationException();
    }
    public MenuComponent getChild(int i) {
        throw new UnsupportedOperationException();
    }

    public String getName() {
        throw new UnsupportedOperationException();
    }
    public String getDescription() {
        throw new UnsupportedOperationException();
    }
    public double getPrice() {
        throw new UnsupportedOperationException();
    }
    public boolean isVegetarian() {
        throw new UnsupportedOperationException();
    }

    public void print() {
        throw new UnsupportedOperationException();
    }
}
```

Because some of these methods only make sense for MenuItems, and some only make sense for Menus, the default implementation is UnsupportedOperationException. That way, if MenuItem or Menu doesn't support an operation, they don't have to do anything, they can just inherit the default implementation.

We've grouped together the "composite" methods – that is, methods to add, remove and get MenuComponents.

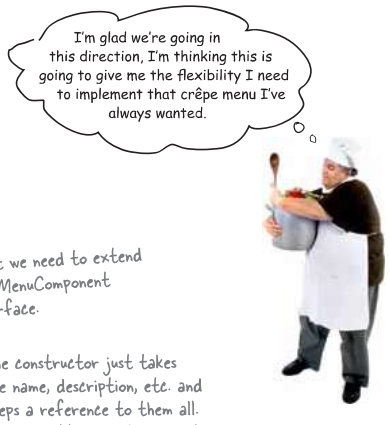
Here are the "operation" methods; these are used by the MenuItems. It turns out we can also use a couple of them in Menu too, as you'll see in a couple of pages when we show the Menu code.

print() is an "operation" method that both our Menus and MenuItems will implement, but we provide a default operation here.

Chapter 9. Well-Managed Collections
Head First Design Patterns By Eric Freeman, Elisabeth Freeman, Bert Bates, Kathy Sierra
ISBN: 0596007124 Publisher: O'Reilly
Print Publication Date: 2004/10/25

Implementing the Menu Item

Okay, let's give the MenuItem class a shot. Remember, this is the leaf class in the Composite diagram and it implements the behavior of the elements of the composite.




```
public class MenuItem extends MenuComponent {
    String name;
    String description;
    boolean vegetarian;
    double price;

    public MenuItem(String name,
                    String description,
                    boolean vegetarian,
                    double price)
    {
        this.name = name;
        this.description = description;
        this.vegetarian = vegetarian;
        this.price = price;
    }

    public String getName() {
        return name;
    }

    public String getDescription() {
        return description;
    }

    public double getPrice() {
        return price;
    }

    public boolean isVegetarian() {
        return vegetarian;
    }

    public void print() {
        System.out.print("  " + getName());
        if (isVegetarian()) {
            System.out.print("(v)");
        }
        System.out.println(", " + getPrice());
        System.out.println("     -- " + getDescription());
    }
}
```

First we need to extend the MenuComponent interface.

The constructor just takes the name, description, etc. and keeps a reference to them all. This is pretty much like our old menu item implementation.

Here's our getter methods – just like our previous implementation.

This is different from the previous implementation. Here we're overriding the print() method in the MenuComponent class. For MenuItem this method prints the complete menu entry: name, description, price and whether or not it's veggie.

Chapter 9. Well-Managed Collections
Head First Design Patterns By Eric Freeman, Elisabeth Freeman, Bert Bates, Kathy Sierra
ISBN: 0596007124 Publisher: O'Reilly
Print Publication Date: 2004/10/25

composite structure

Implementing the Composite Menu

Now that we have the MenuItem, we just need the composite class, which we're calling Menu. Remember, the composite class can hold MenuItems *or* other Menus. There's a couple of methods from MenuComponent this class doesn't implement: getPrice() and isVegetarian(), because those don't make a lot of sense for a Menu.

```
public class Menu extends MenuComponent {
    ArrayList menuComponents = new ArrayList();
    String name;
    String description;

    public Menu(String name, String description) {
        this.name = name;
        this.description = description;
    }

    public void add(MenuComponent menuComponent) {
        menuComponents.add(menuComponent);
    }

    public void remove(MenuComponent menuComponent) {
        menuComponents.remove(menuComponent);
    }

    public MenuComponent getChild(int i) {
        return (MenuComponent)menuComponents.get(i);
    }

    public String getName() {
        return name;
    }

    public String getDescription() {
        return description;
    }

    public void print() {
        System.out.print("\n" + getName());
        System.out.println(", " + getDescription());
        System.out.println("---------------------");
    }
}
```

Menu is also a MenuComponent, just like MenuItem.

Menu can have any number of children of type MenuComponent, we'll use an internal ArrayList to hold these.

This is different than our old implementation: we're going to give each Menu a name and a description. Before, we just relied on having different classes for each menu.

Here's how you add MenuItems or other Menus to a Menu. Because both MenuItems and Menus are MenuComponents, we just need one method to do both.

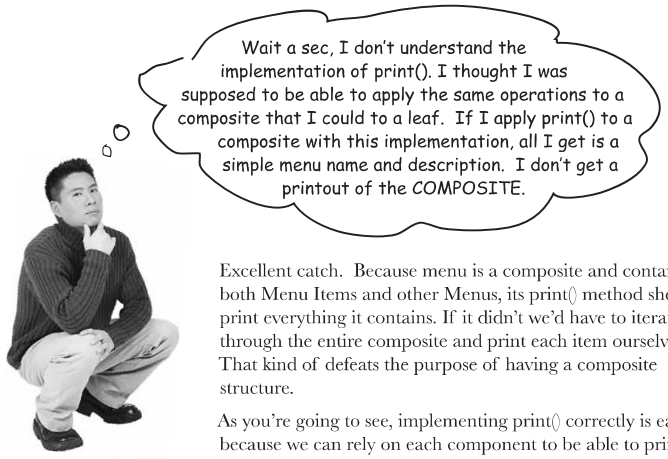
You can also remove a MenuComponent or get a MenuComponent.

Here are the getter methods for getting the name and description.

Notice, we aren't overriding getPrice() or isVegetarian() because those methods don't make sense for a Menu (although you could argue that isVegetarian() might make sense). If someone tries to call those methods on a Menu, they'll get an UnsupportedOperationException.

To print the Menu, we print the Menu's name and description.

Chapter 9. Well-Managed Collections
Head First Design Patterns By Eric Freeman, Elisabeth Freeman, Bert Bates, Kathy Sierra
ISBN: 0596007124 Publisher: O'Reilly
Print Publication Date: 2004/10/25



Excellent catch. Because menu is a composite and contains both Menu Items and other Menus, its print() method should print everything it contains. If it didn't we'd have to iterate through the entire composite and print each item ourselves. That kind of defeats the purpose of having a composite structure.

As you're going to see, implementing print() correctly is easy because we can rely on each component to be able to print itself. It's all wonderfully recursive and groovy. Check it out:

Fixing the print() method

```
public class Menu extends MenuComponent {
    ArrayList menuComponents = new ArrayList();
    String name;
    String description;

    // constructor code here

    // other methods here

    public void print() {
        System.out.print("\n" + getName());
        System.out.println(", " + getDescription());
        System.out.println("---------------------");

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

All we need to do is change the print() method to make it print not only the information about this Menu, but all of this Menu's components: other Menus and MenuItems.

Look! We get to use an Iterator. We use it to iterate through all the Menu's components... those could be other Menus, or they could be MenuItems. Since both Menus and MenuItems implement print(), we just call print() and the rest is up to them.

NOTE: If, during this iteration, we encounter another Menu object, its print() method will start another iteration, and so on.

Chapter 9. Well-Managed Collections
Head First Design Patterns By Eric Freeman, Elisabeth Freeman, Bert Bates, Kathy Sierra
ISBN: 0596007124 Publisher: O'Reilly
Print Publication Date: 2004/10/25

test drive the menu composite

Getting ready for a test drive...

It's about time we took this code for a test drive, but we need to update the Waitress code before we do – after all she's the main client of this code:

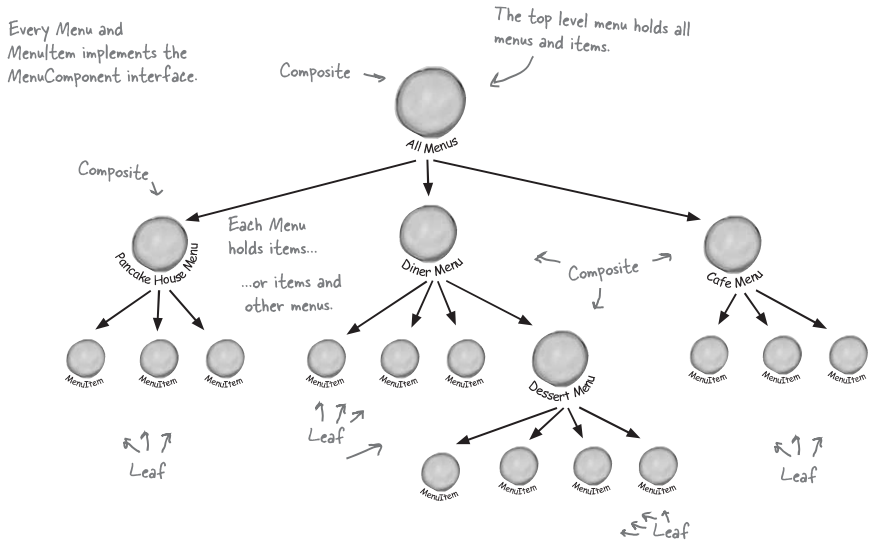
```
public class Waitress {
    MenuComponent allMenus;

    public Waitress(MenuComponent allMenus) {
        this.allMenus = allMenus;
    }

    public void printMenu() {
        allMenus.print();
    }
}
```

Yup! The Waitress code really is this simple. Now we just hand her the top level menu component, the one that contains all the other menus. We've called that allMenus.

All she has to do to print the entire menu hierarchy – all the menus, and all the menu items – is call print() on the top level menu.

We're gonna have one happy Waitress.

Okay, one last thing before we write our test drive. Let's get an idea of what the menu composite is going to look like at runtime:



Head First Design Patterns By Eric Freeman, Elisabeth Freeman, Bert Bates, Kathy Sierra
ISBN: 0596007124 Publisher: O'Reilly
Print Publication Date: 2004/10/25

Now for the test drive...

Okay, now we just need a test drive. Unlike our previous version, we're going to handle all the menu creation in the test drive. We could ask each chef to give us his new menu, but let's get it all tested first. Here's the code:

```
public class MenuTestDrive {
    public static void main(String args[]) {
        MenuComponent pancakeHouseMenu =
            new Menu("PANCAKE HOUSE MENU", "Breakfast");
        MenuComponent dinerMenu =
            new Menu("DINER MENU", "Lunch");
        MenuComponent cafeMenu =
            new Menu("CAFE MENU", "Dinner");
        MenuComponent dessertMenu =
            new Menu("DESSERT MENU", "Dessert of course!");

        MenuComponent allMenus = new Menu("ALL MENUS", "All menus combined");

        allMenus.add(pancakeHouseMenu);
        allMenus.add(dinerMenu);
        allMenus.add(cafeMenu);

        // add menu items here

        dinerMenu.add(new MenuItem(
            "Pasta",
            "Spaghetti with Marinara Sauce, and a slice of sourdough bread",
            true,
            3.89));

        dinerMenu.add(dessertMenu);

        dessertMenu.add(new MenuItem(
            "Apple Pie",
            "Apple pie with a flakey crust, topped with vanilla icecream",
            true,
            1.59));

        // add more menu items here

        Waitress waitress = new Waitress(allMenus);

        waitress.printMenu();
    }
}
```

Let's first create all the menu objects.

We also need two top level menu now that we'll name allMenus.

We're using the Composite add() method to add each menu to the top level menu, allMenus.

Now we need to add all the menu items, here's one example, for the rest, look at the complete source code.

And we're also adding a menu to a menu. All dinerMenu cares about is that everything it holds, whether it's a menu item or a menu, is a MenuComponent.

Add some apple pie to the dessert menu...

Once we've constructed our entire menu hierarchy, we hand the whole thing to the Waitress, and as you've seen, it's easy as apple pie for her to print it out.

Chapter 9. Well-Managed Collections
Head First Design Patterns By Eric Freeman, Elisabeth Freeman, Bert Bates, Kathy Sierra
ISBN: 0596007124 Publisher: O'Reilly
Print Publication Date: 2004/10/25