

CS525

Advanced Software Development

Lesson 9 – The Iterator & Composite Patterns (Part 2: Composite)

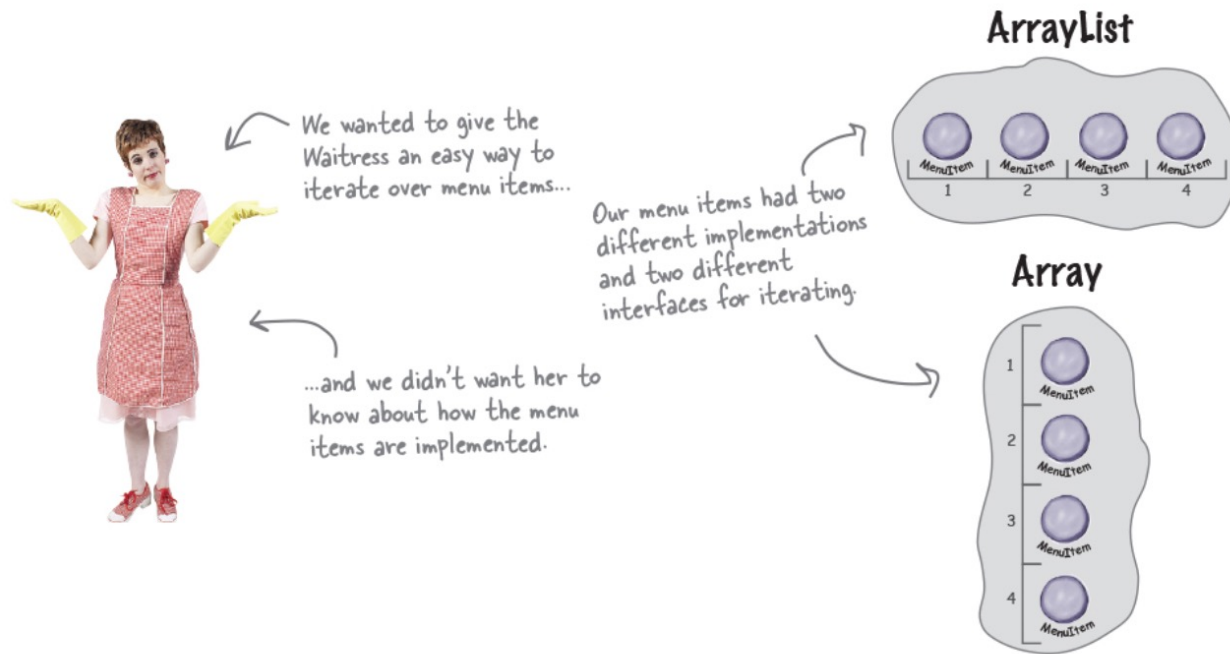
Design Patterns
Elements of Reusable Object-Oriented Software

Payman Salek, M.S.
March 2022



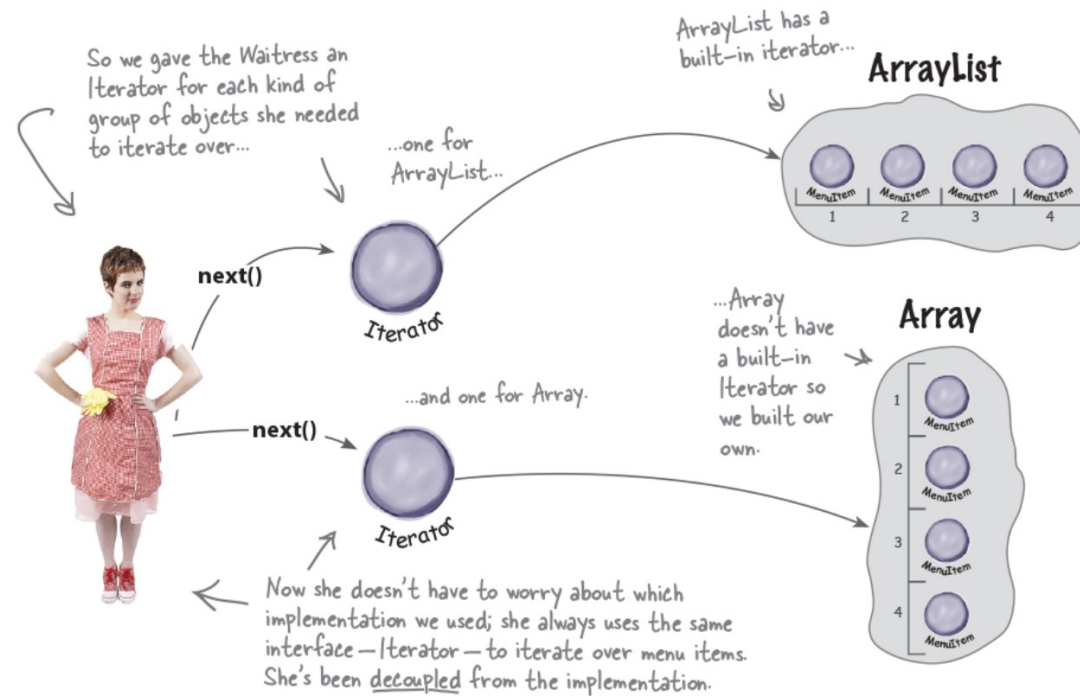
Setting the stage (So far...)

What did we do?



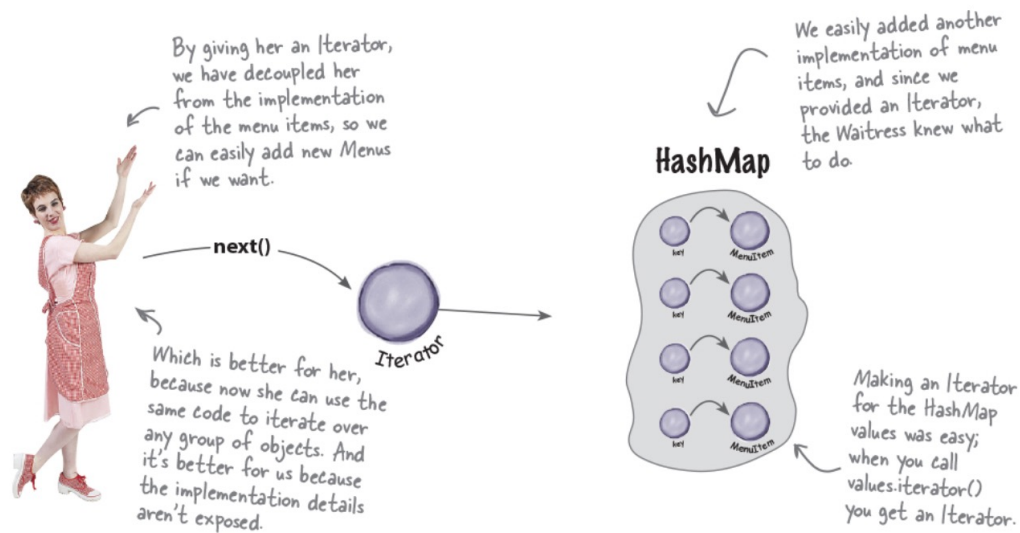
Setting the stage (So far...)

We decoupled the Waitress....



Setting the stage (So far...)

...and we made the Waitress more extensible



But there is still room for improvement

```
public void printMenu() {  
    Iterator<MenuItem> pancakeIterator = pancakeHouseMenu.createIterator();  
    Iterator<MenuItem> dinerIterator = dinerMenu.createIterator();  
    Iterator<MenuItem> cafeIterator = cafeMenu.createIterator();  
  
    System.out.println("MENU\n---\nBREAKFAST");  
    printMenu(pancakeIterator);  
  
    System.out.println("\nLUNCH");  
    printMenu(dinerIterator);  
  
    System.out.println("\nDINNER");  
    printMenu(cafeIterator);  
}
```


Three createIterator() calls.



Three calls to
printMenu.



Every time we add or remove a menu, we're going
to have to open this code up for changes.



Solution

```
public class Waitress {  
    List<Menu> menus;  
  
    public Waitress(List<Menu> menus) {  
        this.menus = menus;  
    }  
  
    public void printMenu() {  
        Iterator<Menu> menuIterator = menus.iterator();  
        while(menuIterator.hasNext()) {  
            Menu menu = menuIterator.next();  
            printMenu(menu.createIterator());  
        }  
    }  
  
    void printMenu(Iterator<MenuItem> iterator) {  
        while (iterator.hasNext()) {  
            MenuItem menuItem = iterator.next();  
            System.out.print(menuItem.getName() + ", ");  
            System.out.print(menuItem.getPrice() + " -- ");  
            System.out.println(menuItem.getDescription());  
        }  
    }  
}
```

Now we just take a list of menus, instead of each menu separately.

And we iterate through the menus, passing each menu's iterator to the overloaded printMenu() method.

No code changes here.

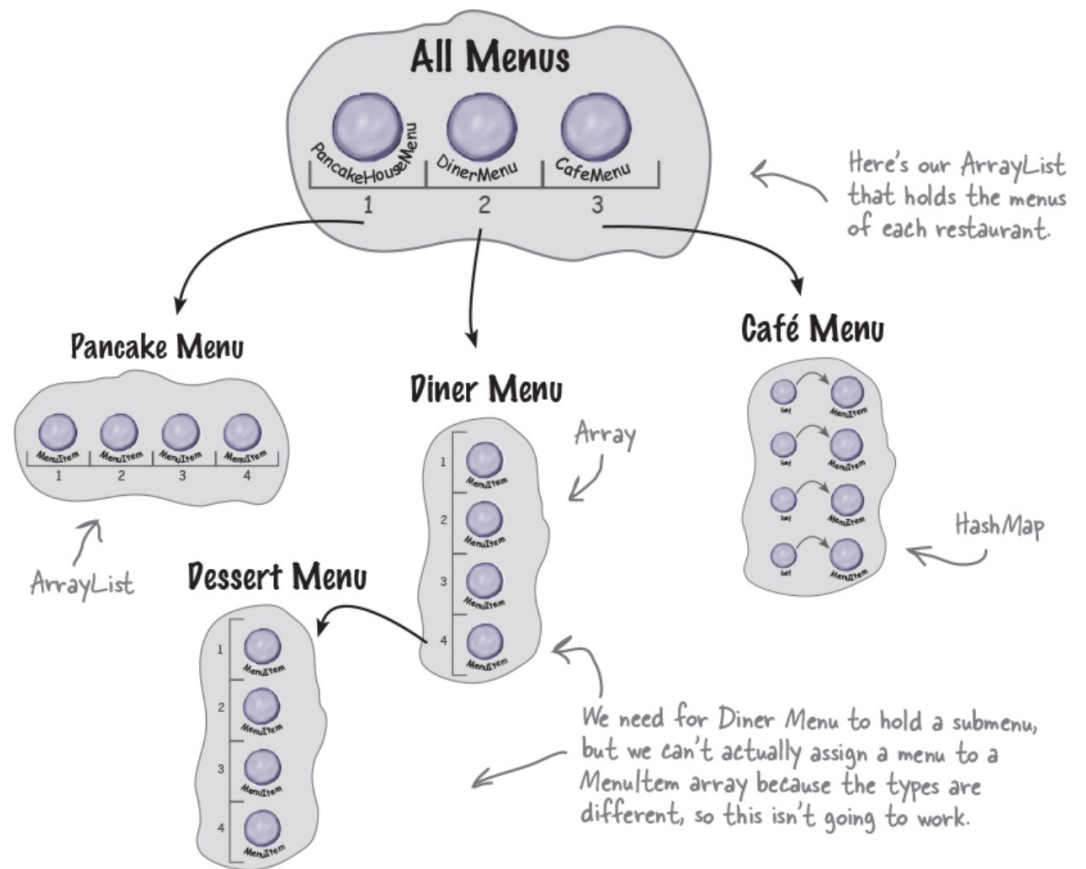
Just when we thought it was safe!

Now they want to add a dessert submenu.

Okay, now what? Now we have to support not only multiple menus, but menus within menus.

It would be nice if we could just make the dessert menu an element of the DinerMenu collection, but that won't work as it is now implemented.

The New Challenge



But this won't work!

We can't assign a dessert menu to a MenuItem array.

Time for a change!

What do we really need?

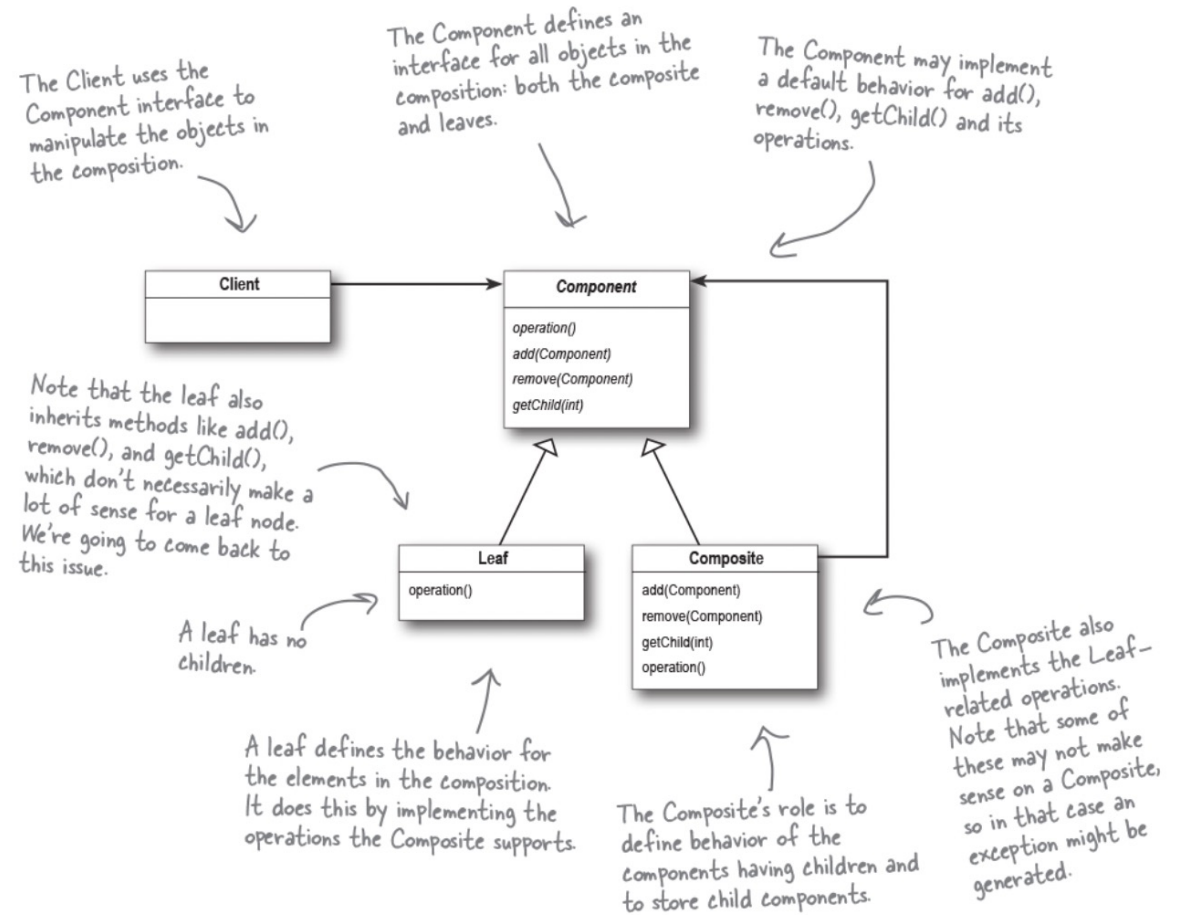
- We need some kind of a tree-shaped structure that will accommodate menus, submenus, and menu items.
- We need to make sure we maintain a way to traverse the items in each menu that is at least as convenient as what we're doing now with iterators.
- We may need to traverse the items in a more flexible manner. For instance, we might need to iterate over only the Diner's dessert menu, or we might need to iterate over the Diner's entire menu, including the dessert submenu.

The Composite Pattern

The Composite Pattern allows us to build structures of objects in the form of trees that contain both compositions of objects and individual objects as nodes.

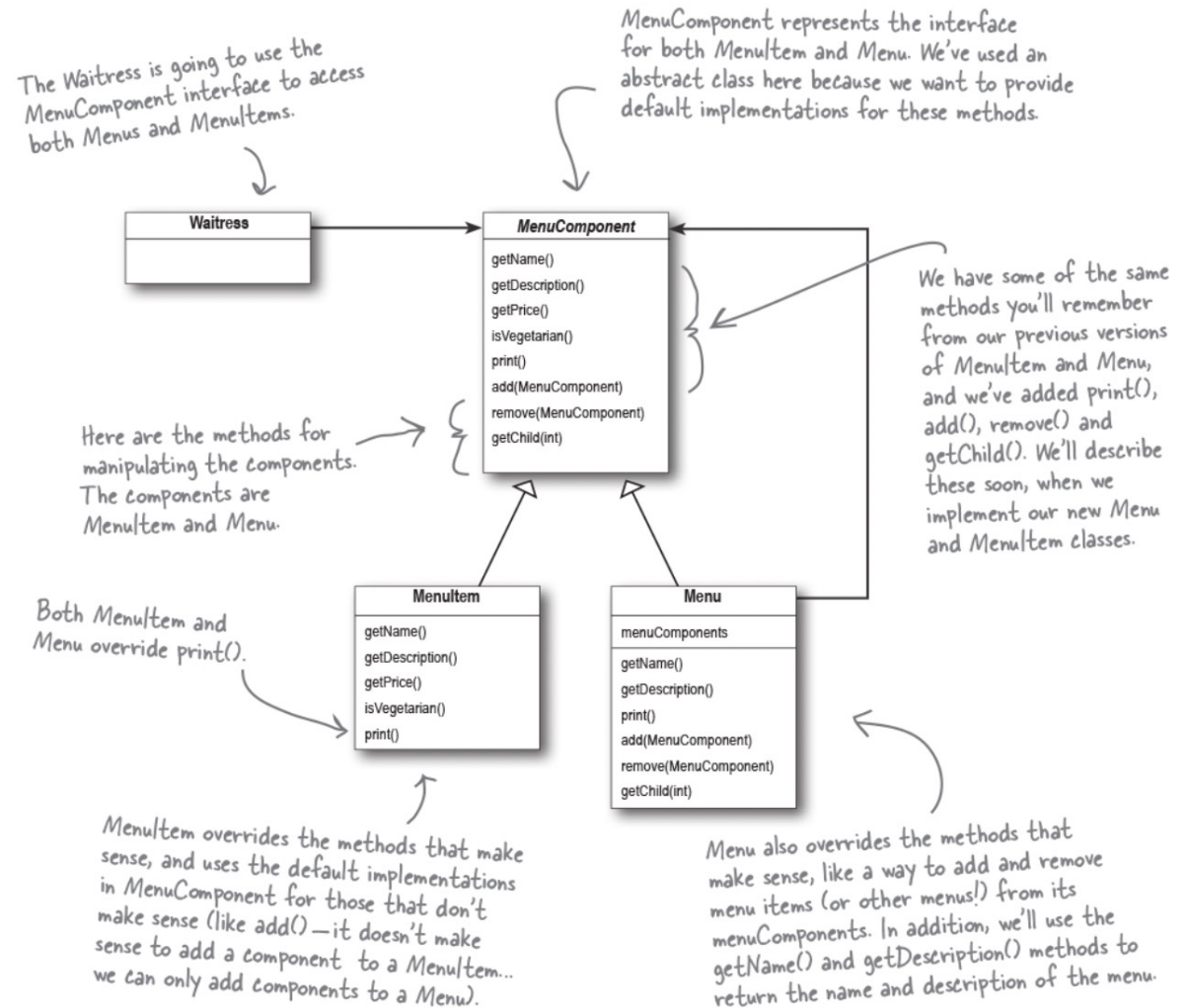
Using a composite structure, we can apply the same operations over both composites and individual objects. In other words, in most cases we can ignore the differences between compositions of objects and individual objects.

UML Class Diagram for Composite



UML Class Diagram for Our Solution

6/6/22



Implementing the Menu Component

6/6/22

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, it doesn't have to do anything; it 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.

Implementing the Menu Item

6/6/22

```
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 MenuItem 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.

Implementing the Menu (Composite Item)

Incomplete Version

6/6/22

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.

```
public class Menu extends MenuComponent {
    List<MenuComponent> menuComponents = new ArrayList<MenuComponent>();
    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 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("-----");
    }
}
```

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 its name and description.

Fixing the print() method

```
public class Menu extends MenuComponent {  
    List<MenuComponent> menuComponents = new ArrayList<MenuComponent>();  
    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("-----");  
  
        for (MenuComponent menuComponent : menuComponents) {  
            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 MenuItem's.


Look! We get to use an Iterator behind the scenes of the enhanced for loop. We use it to iterate through all the Menu's components...those could be other Menus, or they could be MenuItem's.

Since both Menus and MenuItem's 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.

Test Drive!

```
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.

Summary

6/6/22

