|  |  |
| --- | --- |
| logo-dai-hoc-gia-dinh | **BỘ GIÁO DỤC VÀ ĐÀO TẠO**  **TRƯỜNG ĐẠI HỌC GIA ĐỊNH**  **KHOA CÔNG NGHỆ THÔNG TIN** |

**BÁO CÁO**

**LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG NÂNG CAO**

**TÊN ĐỀ TÀI**

**THE ITERATOR AND COMPOSITE PATTERNS**

**Giảng viên hướng dẫn: LÊ HUỲNH PHƯỚC**

**Nhóm thực hiện đề tài: nhóm 4**

**Thành viên thực hiện:**

**NGUYỄN TRƯỜNG HÙNG - 2004110032 - K14DCPM01**

**TRẦN QUANG KHẢI - 2004110035 - K14DCPM01**

**BÙI TIẾN ĐẠT - 2004110039 - K14DCPM01**

**NGUYỄN LÊ QUANG THUẬN - 2004110045 - K14DCPM02**

**Khóa: 14**

*Học kỳ 3, ngày 23 tháng 08 năm 2022*

**MỤC LỤC**

THE ITERATOR AND COMPOSITE PATTERNS

[I. ITERATOR 1](#_Toc112338520)

[1. Giới thiệu sơ về Collection 1](#_Toc112338521)

[2. Đặt vấn đề 1](#_Toc112338522)

[3. Ví dụ 2](#_Toc112338523)

[4. Làm cách nào để giảm bớt vòng lặp 6](#_Toc112338524)

[5. Giới thiệu Iterator Pattern 7](#_Toc112338525)

[6. Áp dụng Iterator Pattern vào DinerMenu 8](#_Toc112338526)

[7. Làm lại DinerMenu với Iterator Pattern 9](#_Toc112338527)

[8. SỬA CODE WAITRESS 9](#_Toc112338528)

[9. TESTING CODE 10](#_Toc112338529)

[10. Cải tiến menu với Iterator Pattern của Java 11](#_Toc112338530)

[11. Iterator đã cho chúng ta những gì 13](#_Toc112338531)

[12. Định nghĩa Iterator Pattern 14](#_Toc112338532)

[13. Thêm CafeMenu (sử dụng Collection Hashtable) 15](#_Toc112338533)

[14. Thêm CafeMenu vào Waitress 15](#_Toc112338534)

[15. Cải tiến Waitress 17](#_Toc112338535)

[II. Composite 19](#_Toc112338536)

[1. Vấn đề mới 19](#_Toc112338537)

[2. Cách giải quyết vấn đề ? 19](#_Toc112338538)

[3. Định nghĩa Composite Pattern 20](#_Toc112338539)

[4. Thiết kế Menu với Composite pattern 23](#_Toc112338540)

[5. CÀI ĐẶT MENU COMPONENT 24](#_Toc112338541)

[6. CÀI ĐẶT MENU ITEM 25](#_Toc112338542)

[7. CÀI ĐẶT COMPOSITE MENU 26](#_Toc112338543)

[8. SỬA PHƯƠNG THỨC PRINT() 27](#_Toc112338544)

[9. Sửa lại Class Waitress và test drive 28](#_Toc112338545)

[10. Composite kết hợp với Iterator 30](#_Toc112338546)

[10.1 Sơ đồ lớp 30](#_Toc112338547)

[10.2 Sửa class Menu 30](#_Toc112338548)

[10.3 Sử dụng collection ArrayList 31](#_Toc112338549)

[10.4 Sử dụng collection HashMap 32](#_Toc112338550)

[10.5 Test drive 33](#_Toc112338551)

[10.6 CompositeIterator 35](#_Toc112338552)

[10.7 In toàn bộ món chay 37](#_Toc112338553)

[III. Tổng kết 38](#_Toc112338554)

[1. Tóm tắt Iterator 38](#_Toc112338555)

[2. Tóm tắt Composite 38](#_Toc112338556)

**LỜI NÓI ĐẦU**

Design pattern là một kỹ thuật dành cho lập trình hướng đối tượng. Nó cung cấp cho ta cách tư duy trong từng tình huống của việc lập trình hướng đối tượng, và phân tích thiết kế hệ thống phần mềm.Nó cần thiết cho cả các nhà lập trình và nhà phân tích thiết kế. Đối với những người chuyên về lập trình thì việc nắm vững công cụ lập trình thôi chưa đủ,họ cần phải có một tư duy, một kỹ năng giải quyết các tình huống nhỏ của công việc xây dựng phần mềm mà họ là người thi hành.Việc giải quyết này phải đảm bảo tính ổn định là họ có thể giải quyết được trong mọi tình huống, với thời gian đúng tiến độ, phương pháp giải quyết hợp lý và đặc biệt là phải theo một chuẩn nhất định.Những nhà phân tích thiết kế mức cao, việc nắm vững công cụ lập trình có thể là không cần thiết, nhưng họ cũng cần phải biết được ở những khâu nhỏ nhất chi tiết nhất của thiết kế của họ đưa ra có thể thực hiện được hay không và nếu thực hiện được thì có thể thực hiện như thế nào, và sẽ theo một chuẩn ra sao.

Design pattern được dùng khắp ở mọi nơi, trong các phần mềm hướng đối tượng các hệ thống lớn. Trong các chương trình trò chơi, ... Và cả trong các hệ thống tính toán song song,..

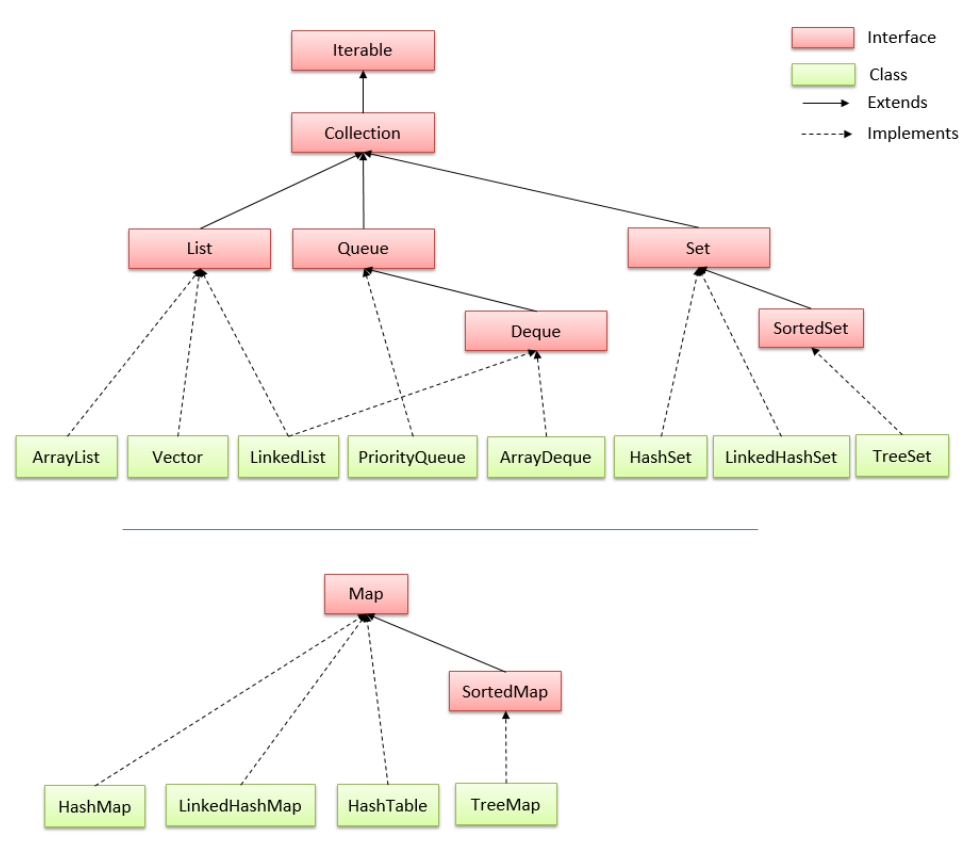
Design pattern thể hiện tính kinh nghiệm của công việc lập trình, xây dựng và thiết kế phần mềm.Có thể chúng ta đã gặp design pattern ở đâu đó, trong các ứng dụng, cũng có thể chúng ta đã từng sử dụng những mẫu tương tự như design pattern để giải quyết những tình huống của mình, nhưng chúng ta không có một khái niệm gì về nó cả.Trong nội dung đồ án môn học này nhóm em xin trình bày những hiểu biết của mình về design pattern theo hướng tiếp cận mang tính kinh nghiệm. Việc cài đặt các mẫu được trình bày trên một tài liệu đi kèm.

Chúng em xin cảm ơn sự hướng dẫn của thầy Lê Huỳnh Phước, đã giúp đỡ chúng em hoàn thành đồ án môn học này.

1. ITERATOR
2. Giới thiệu sơ về Collection

* Các đối tượng có kiểu dữ liệu giống nhau có thể được gom lại thành 1 nhóm gọi là collection (tập hợp). Vậy cụ thể Collection là gì?
* Collection là:
* Tập hợp dùng lưu trữ, thao tác trên một nhóm các đối tượng.
* Các đối tượng của tập hợp có thể thuộc nhiều loại dữ liệu khác nhau
* Số phần tử trong tập hợp có thể thêm hoặc bớt

Sau đây là một số Collection:

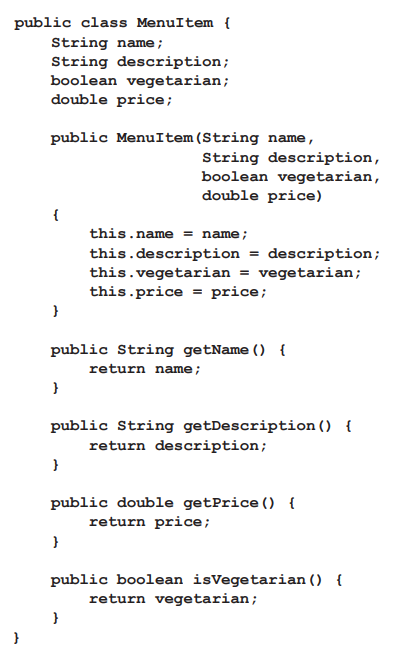


1. Đặt vấn đề

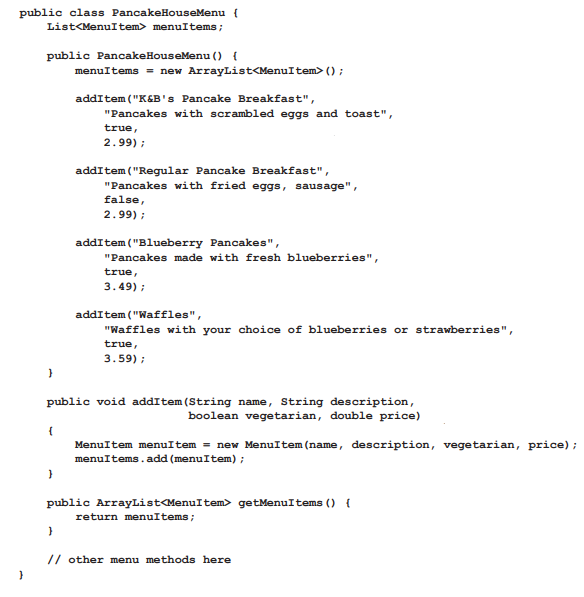
* Mỗi loại **collection** như **Array**, **Stack**, **List** hoặc **ArrayList**, **Hashtable, vv** có những lợi thế và sự đánh đổi riêng. Nhưng đến một lúc nào đó, client của bạn sẽ muốn duyệt qua những đối tượng đó, và khi đó, bạn sẽ tự tay cài đặt thuật toán duyệt của mình (duyệt 1 **mảng** sẽ khác duyệt 1 **ArrayList**). Điều đó không chuyên nghiệp. Khi xem qua phần Iterator pattern, bạn sẽ thấy cách bạn có thể cho phép client của mình duyệt qua các đối tượng mà không cần phải xem qua cách bạn lưu trữ các đối tượng.

1. Ví dụ

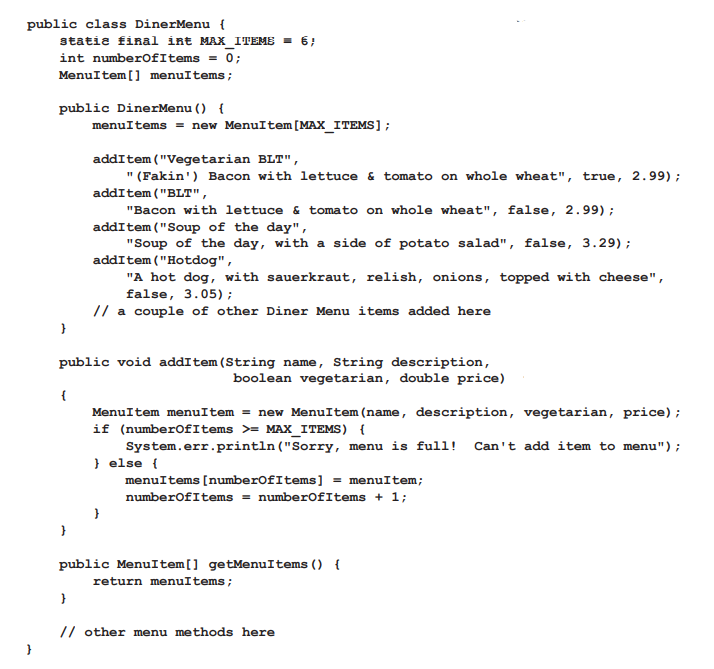
* Có 2 nhà hàng bán đồ ăn tối và bán đồ ăn sáng sáp nhập khi 2 nhà hàng sáp nhập với nhau Ông Mel có một DinerMenu (Menu bữa tối) được lưu vào array còn ông Lou có một PancakeHouseMenu (Menu bữa sáng) được lưu vào ArrayList mỗi menu bao gồm nhiều món ăn và mỗi món ăn (menuItem) đều có tên, mô tả, giá.



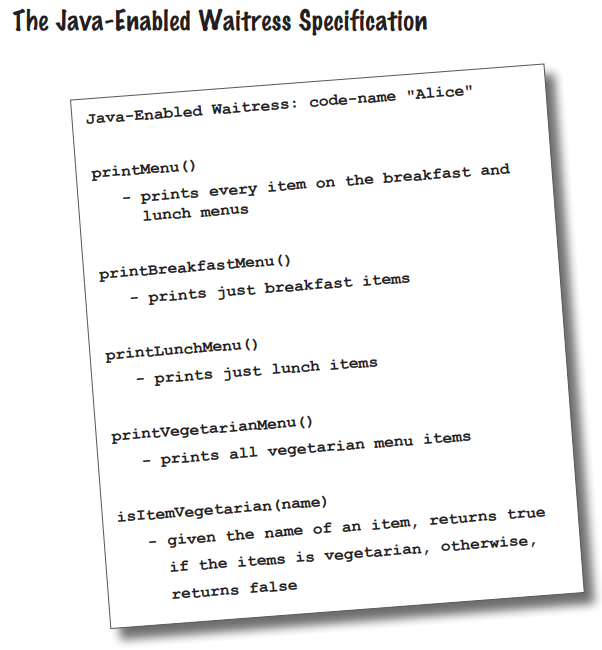
Đây là lớp lưu trữ dữ liệu của món ăn hoặc thức uống



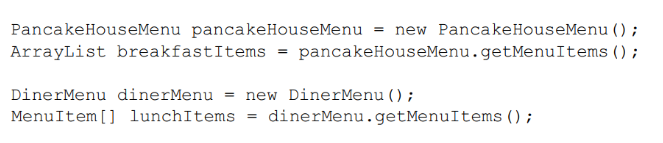
Lớp PancakeHouseMenu sử dụng collection là List. Còn lớp DinnerMenu thì sử dụng mảng để chứa MenuItem. Sau đây là lớp DinnerMenu:



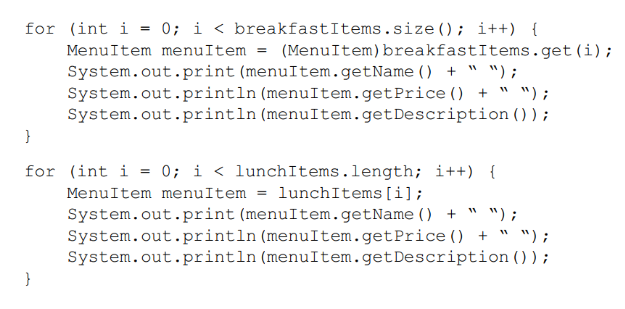
Ở đây chúng ta có thể thấy 2 menu được viết theo cấu trúc danh sách khác nhau. Điều đó làm phức tạp mọi thứ, hãy thử thực hiện một ứng dụng client sử dụng hai menu đó là tạo ra một phục vụ **Java-enabled Waitress**. Mô tả cho người phục vụ để cô ấy có thể in một menu có sự hợp nhất giữa Diner và Pancake House theo yêu cầu của khách hàng.



* Để in tất cả các item trên mỗi menu, bạn sẽ cần gọi phương thức getMenuItem() trên PancakeHouseMenu và DinerMenu để lấy các menu item tương ứng của chúng. Lưu ý rằng mỗi phương thức trả về một kiểu lưu trữ khác nhau (mảng và ArrayList):



* Bây giờ, để in ra các item từ PancakeHouseMenu, chúng ta sẽ duyệt qua các item trên ArrayList breakfastItems. Và để in ra các Diner item, chúng ta sẽ duyệt qua một mảng.

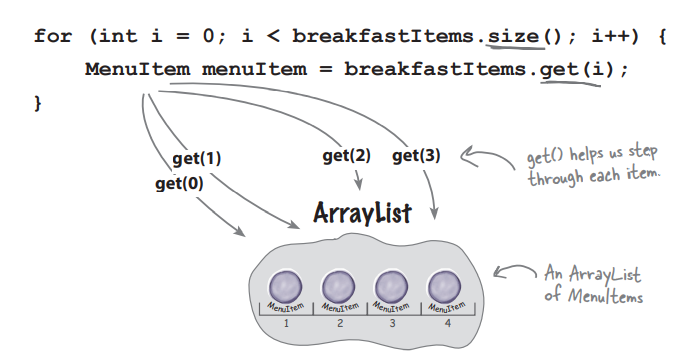


Đoạn code trên cho thấy có 2 vòng lặp vì 2 menu sử dụng collection khác nhau. Nếu có thêm menu sử dụng collection thứ 3 sẽ cần thêm 1 vòng lặp khác . Điều đó sẽ khiến code khó bảo trì vì có nhiều vòng lặp

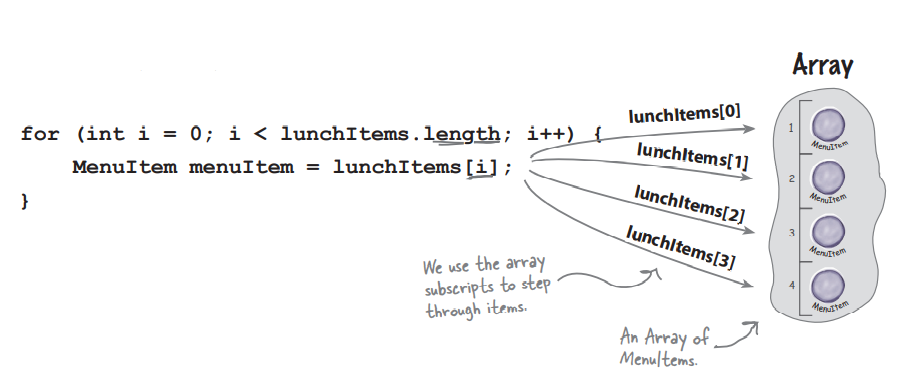
1. Làm cách nào để giảm bớt vòng lặp

Để giảm bớt vòng lặp hãy đóng gói những gì thay đổi. Những gì đang thay đổi ở đây đó là: “duyệt phần tử” được gây ra bởi các kiểu tập hợp khác nhau, được return từ các menu.

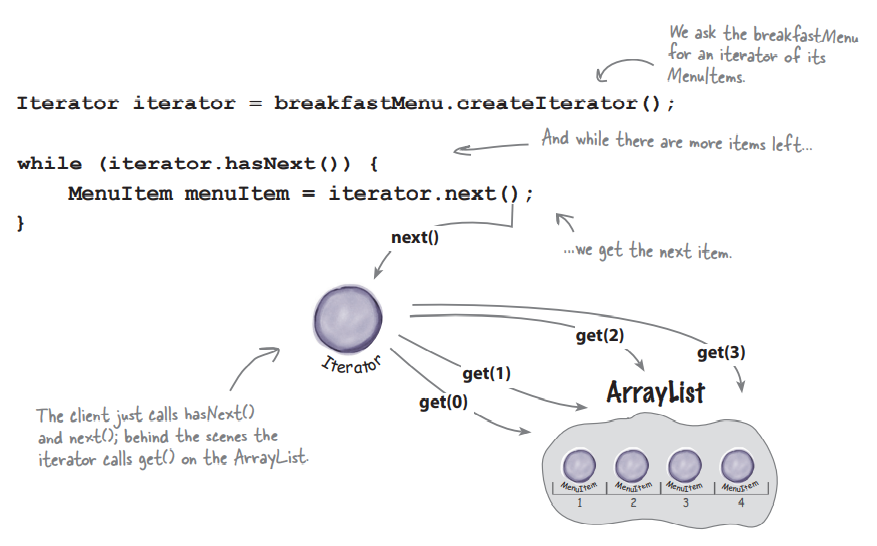
* Để lặp qua các breakfast item, chúng ta sử dụng các phương thức size() và get() trên ArrayList:



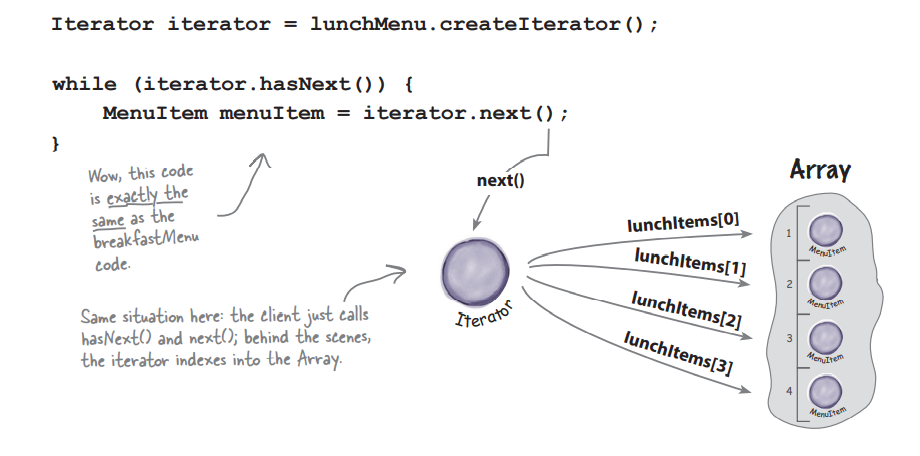
* Và để lặp lại qua các lunch item, chúng ta sử dụng độ dài của Array và index trên mảng MenuItem.



* Bây giờ nếu chúng ta tạo một đối tượng, hãy gọi nó là Iterator, nó gói gọn cách chúng ta lặp qua một Collection chứa các đối tượng ? Hãy để thử cái này trên ArrayList

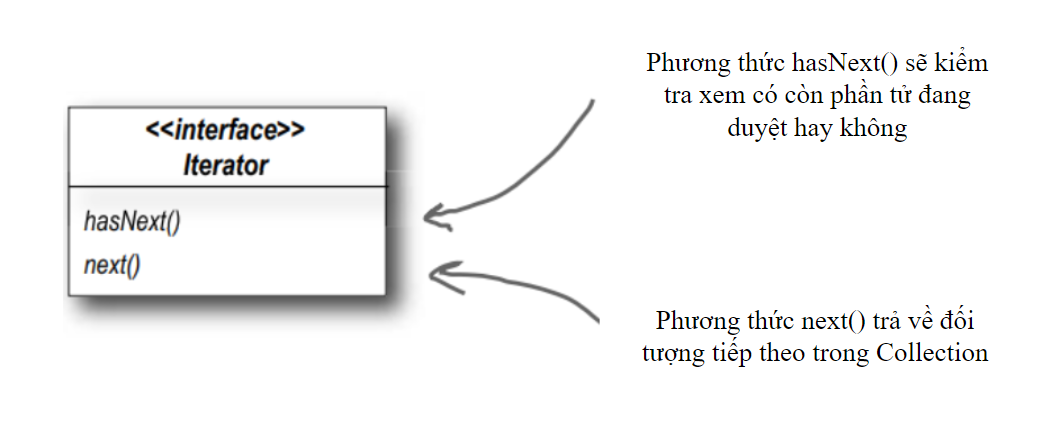


* Hãy cùng thử Iterator với Array (mảng)

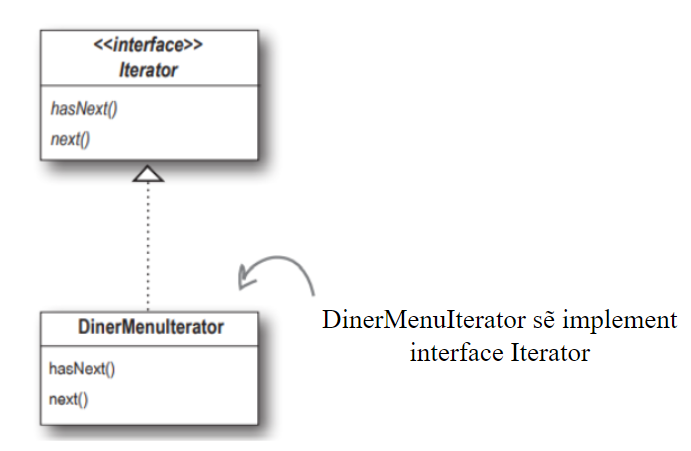


1. Giới thiệu Iterator Pattern

Điều đầu tiên bạn cần biết về Iterator Pattern là nó dựa trên interface có tên là Iterator. Ở đây, một Iterator interface có thể có:

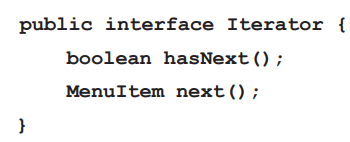


Bây giờ, khi chúng ta có interface này, chúng ta có thể triển khai Iterator pattern cho bất kỳ loại Collection nào: mảng, ArrayList hoặc List, hashtables, . Khi chúng ta muốn triển khai Iterator pattern cho Array được sử dụng trong DinerMenu. Nó sẽ trông như thế này:

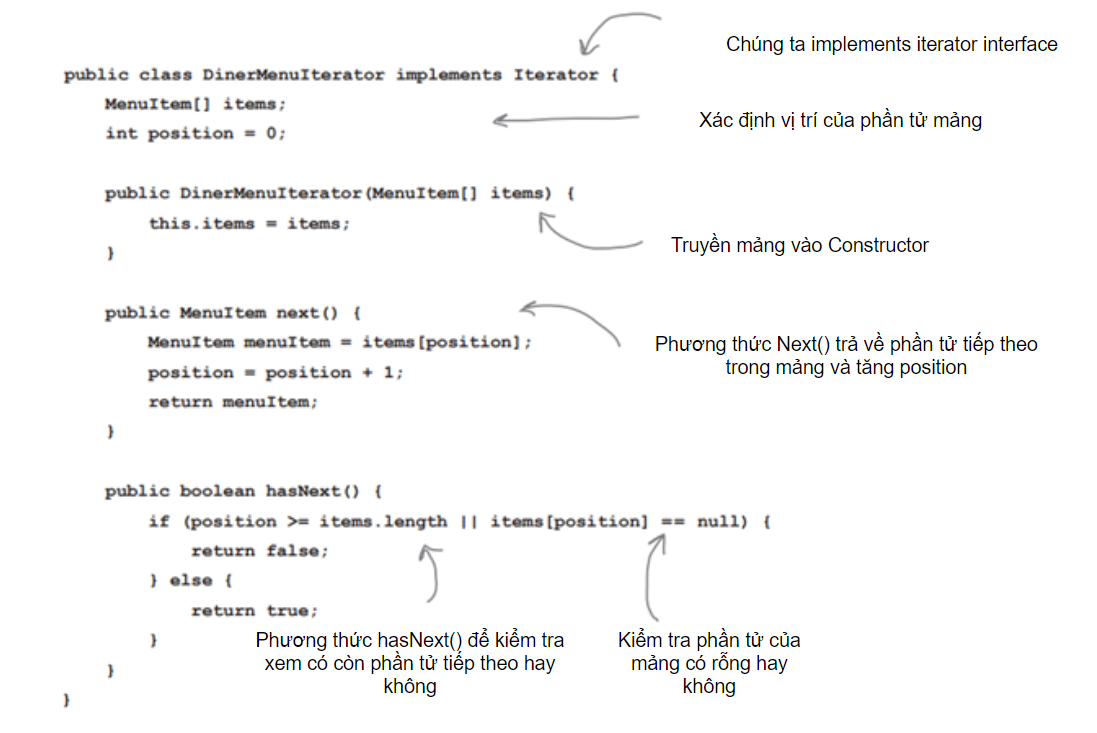


1. Áp dụng Iterator Pattern vào DinerMenu

Để thêm Iterator vào DinerMenu, trước tiên chúng ta cần định nghĩa Iterator Interface:

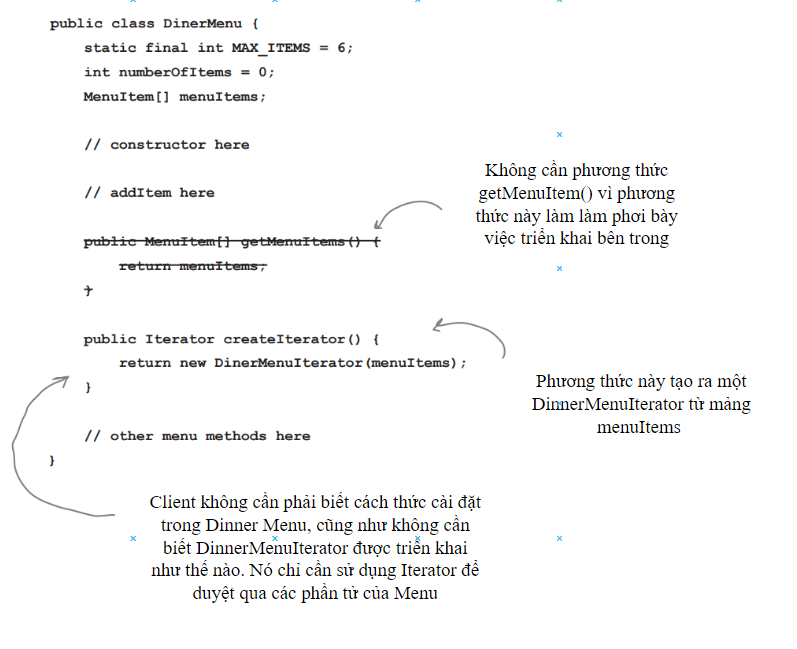


Và bây giờ chúng ta cần implement một Iterator cho Diner Menu:



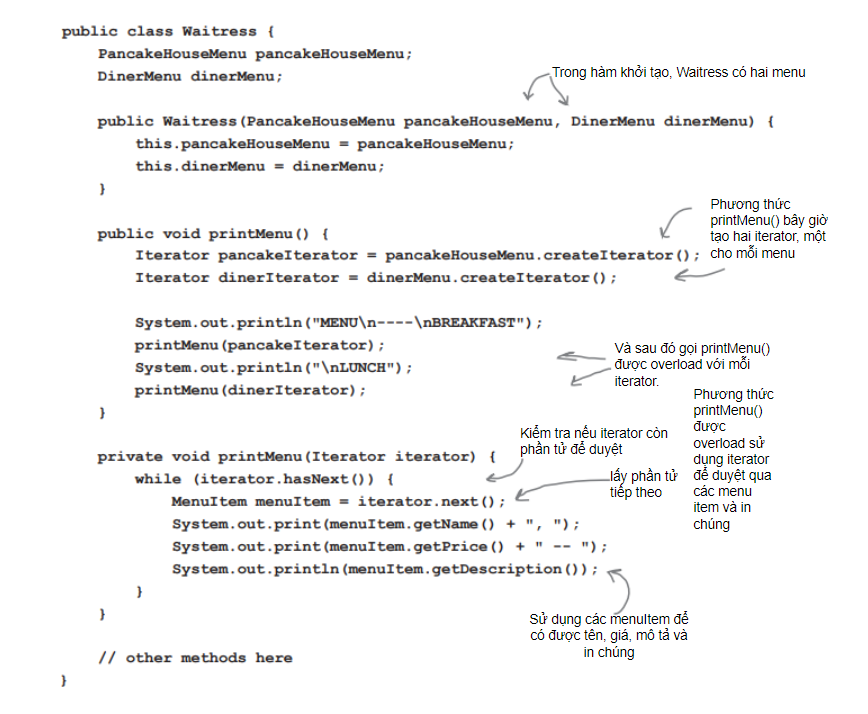
1. Làm lại DinerMenu với Iterator Pattern

Tất cả những gì chúng ta cần làm là thêm một phương thức để tạo **DinerMenuIterator** và return lại cho client:



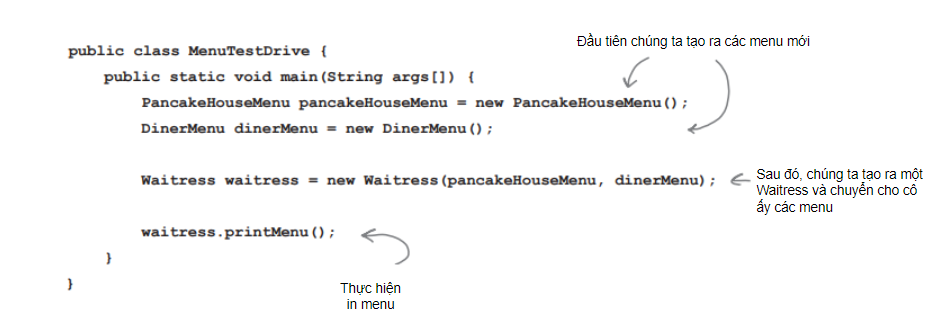
1. SỬA CODE WAITRESS

Bây giờ chúng ta cần tích hợp **iterator pattern** vào Waitress. Việc tích hợp khá đơn giản: đầu tiên tạo một phương thức **printMenu()**, sau đó sử dụng phương thức **createIterator()** trên mỗi menu để lấy **Iterator** và chuyển nó sang phương thức mới.

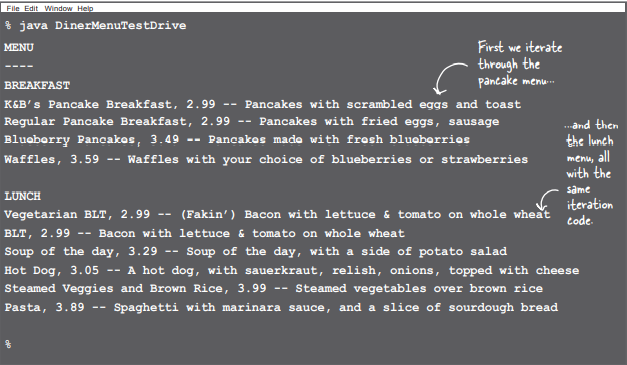


1. TESTING CODE

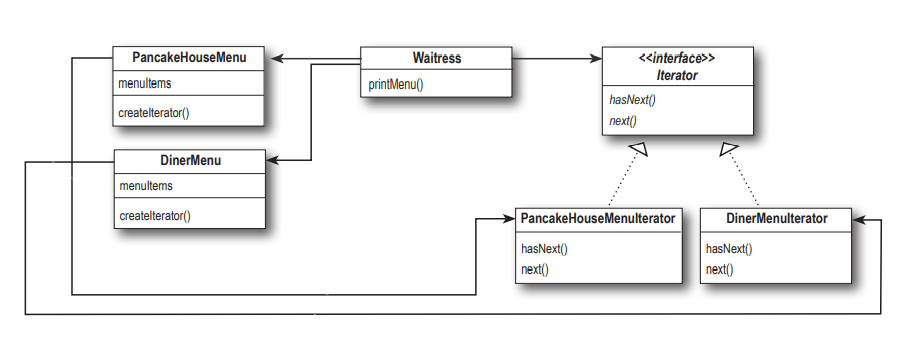
Viết một số test drive và xem cách Waitress hoạt động…



Kết quả

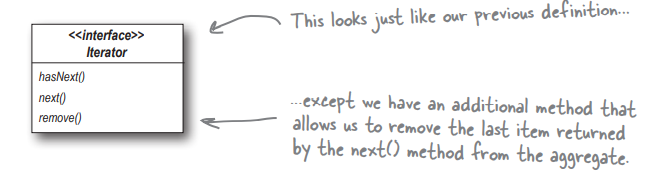


Hiện tại chúng ta có sơ đồ lớp như sau:



1. Cải tiến menu với Iterator Pattern của Java

Java cung cấp sẵn cho chúng ta interface Iterator. Hãy xem qua interface java.util.Iterator:



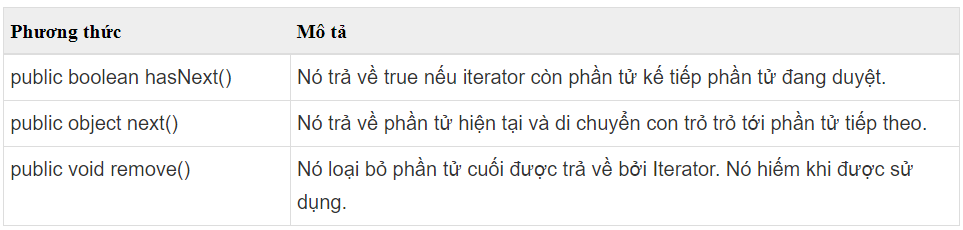
Bạn có thể sử dụng interator của java để:

Duyệt các phần tử từ đầu đến cuối của một collection.

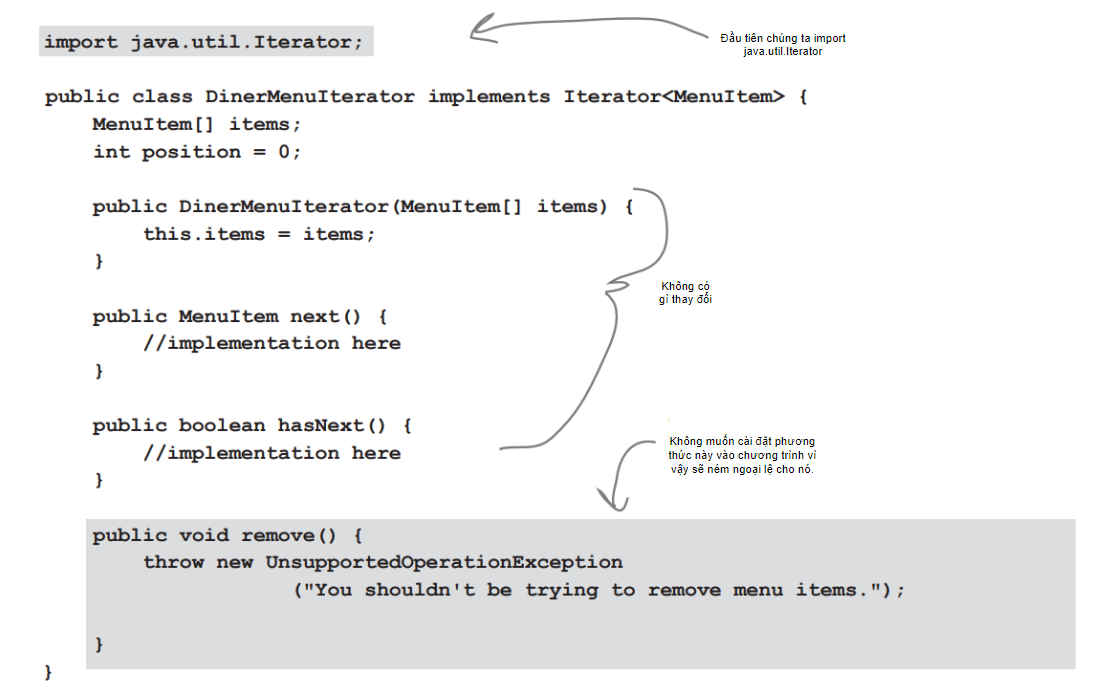
Iterator cho phép xóa phần tử khi lặp một collection.

Một số phương thức của Iterator trong Java

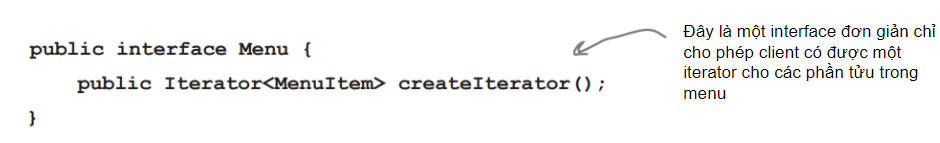
Có ba phương thức trong Iterator như sau:



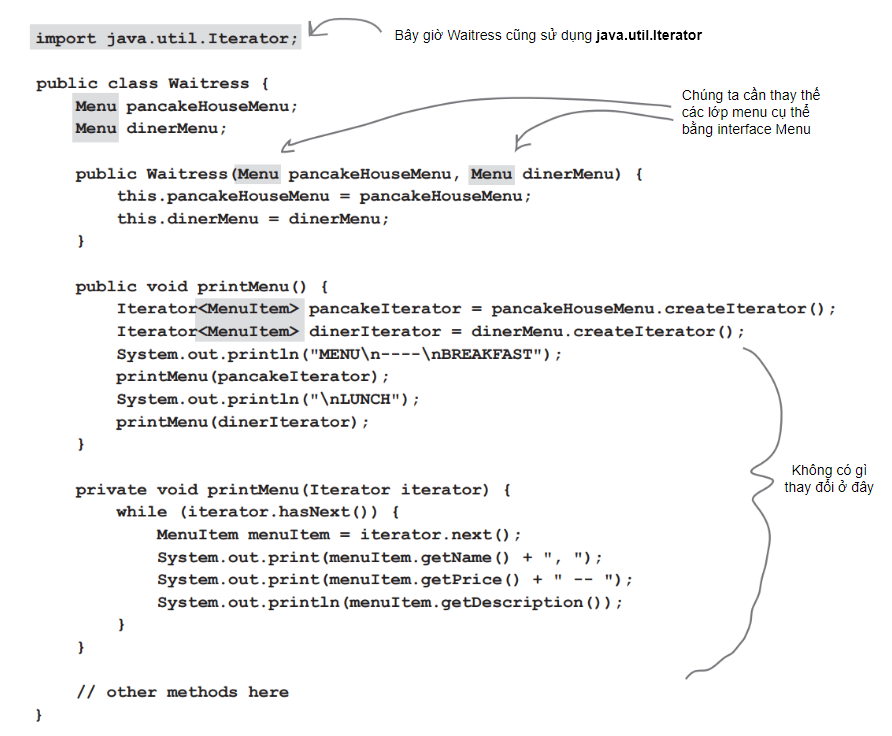
Bây giờ chúng ta sẽ cài đặt java.util.Iterator cho DinerMenu



Chúng ta cần cung cấp cho các Menu một interface chung và code lại Waitress một chút. Bây giờ chúng ta sẽ cho phép các đầu bếp kiểm soát các menu của họ bằng cách đưa phương thức đó ra khỏi public interface:



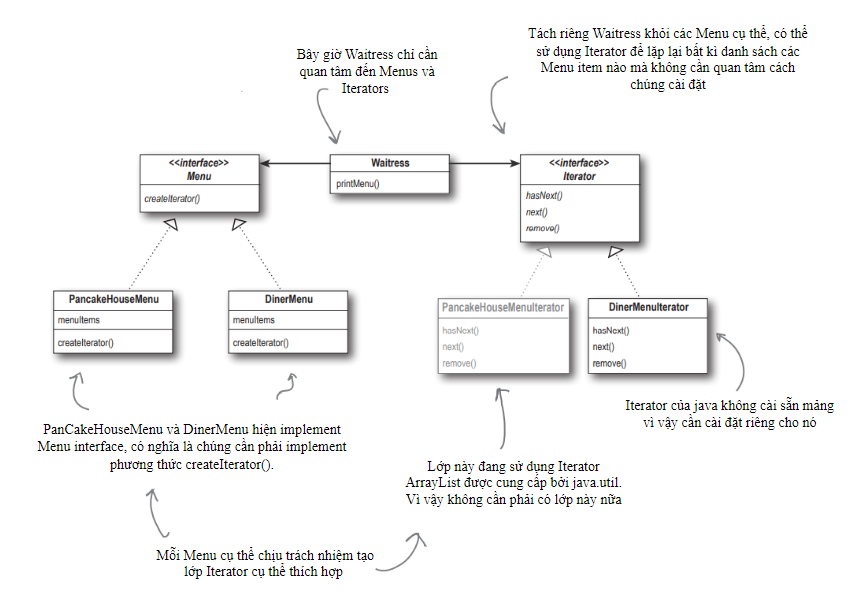
Tiếp theo cần implements Menu vào 2 lớp cụ thể là PancakeHouseMenu và DinerMenu, sau đó cập nhật Waitress:



1. Iterator đã cho chúng ta những gì

Các lớp PancakeHouseMenu và DinerMenu implement cùng một interface là Menu. Waitress có thể tham chiếu từng đối tượng menu bằng interface chứ không phải lớp cụ thể. Do đó sẽ giảm bớt sự phụ thuộc giữa Waitress và các lớp cụ thể bằng cách code với interface, chứ không code với implement => Điều này giải quyết vấn đề của Waitress phụ thuộc vào menu cụ thể.

Interface Menu mới có một phương thức createIterator(), được implement bởi PancakeHouseMenu và DinerMenu. Mỗi lớp menu đảm nhận trách nhiệm tạo ra một Iterator cụ thể phù hợp cho việc thực hiện bên trong các menu item => Điều này giải quyết vấn đề của Waitress phụ thuộc vào việc triển khai MenuItem.



1. Định nghĩa Iterator Pattern

Iterator Pattern cung cấp một cách để truy cập các phần tử của một tập hợp đối tượng một cách tuần tự mà không làm lộ đại diện bên dưới của nó.

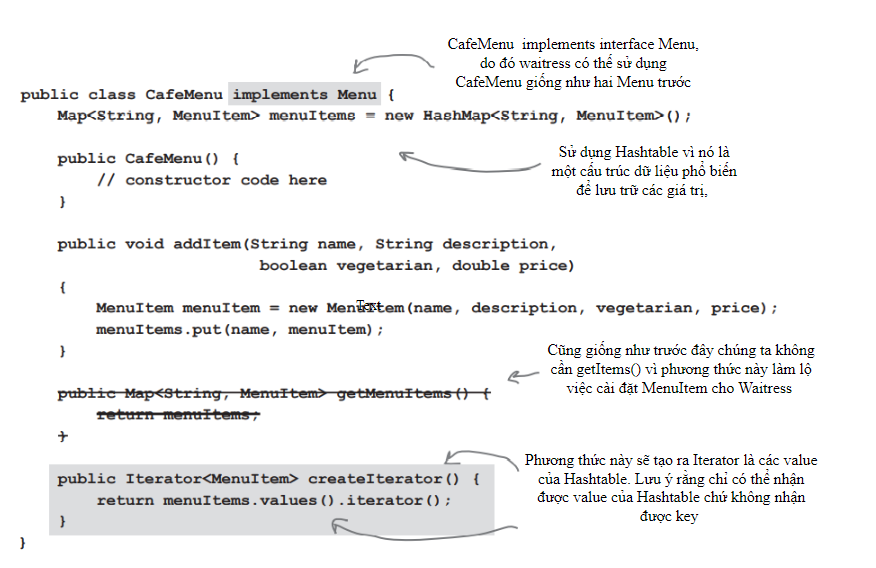
Định nghĩa này rất có ý nghĩa: mẫu cung cấp cho bạn một cách để duyệt qua các phần tử của tập hợp mà không cần phải biết cách mọi thứ được thể hiện bên trong. Bạn đã thấy điều đó với hai Menu được cài đặt. Nhưng hiệu quả của việc sử dụng các Iterator Pattern trong thiết kế của bạn là: bạn có thể viết code đa hình hoạt động với bất kỳ tập hợp nào – giống như phương thức printMenu(), nó không quan tâm các menu item được lưu trong một Array hay ArrayList (hoặc bất cứ thứ gì khác có thể tạo Iterator), miễn là nó có thể tạo được Iterator.

Iterator Pattern: Sơ đồ lớp

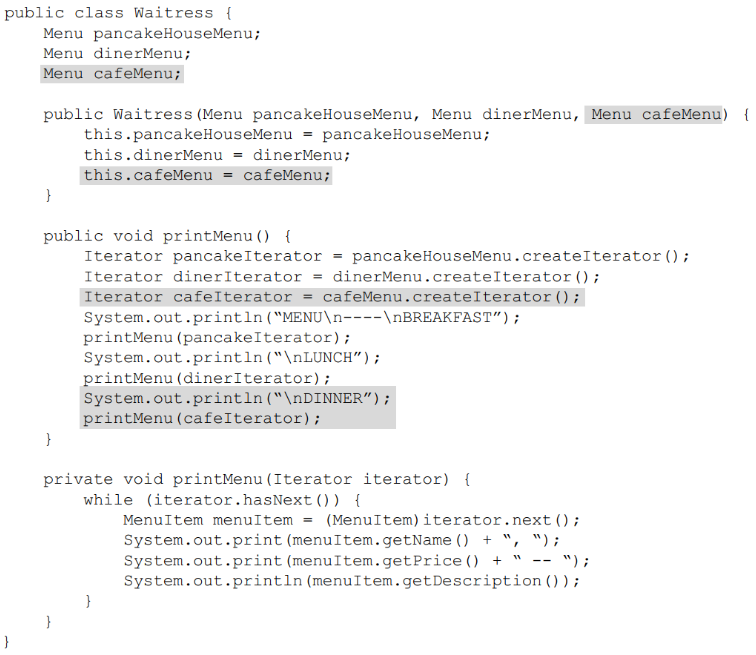


1. Thêm CafeMenu (sử dụng Collection Hashtable)

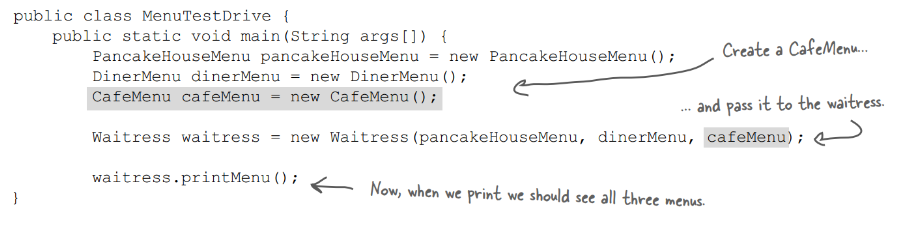
Ở class CafeMenu chúng ta chủ dụng Collection là Hashtable. Đây là Collection được Java hỗ trợ Iterate. Hashtable phức tạp hơn 1 chút so với Arraylist vì nó hỗ trợ cả key và value nhưng có thể nhận được Iterate là các value của Hashtable



1. Thêm CafeMenu vào Waitress



Bây giờ chúng ta sẽ cập nhật lại testDrive để đảm bảo mọi thứ đều hoạt động đúng.

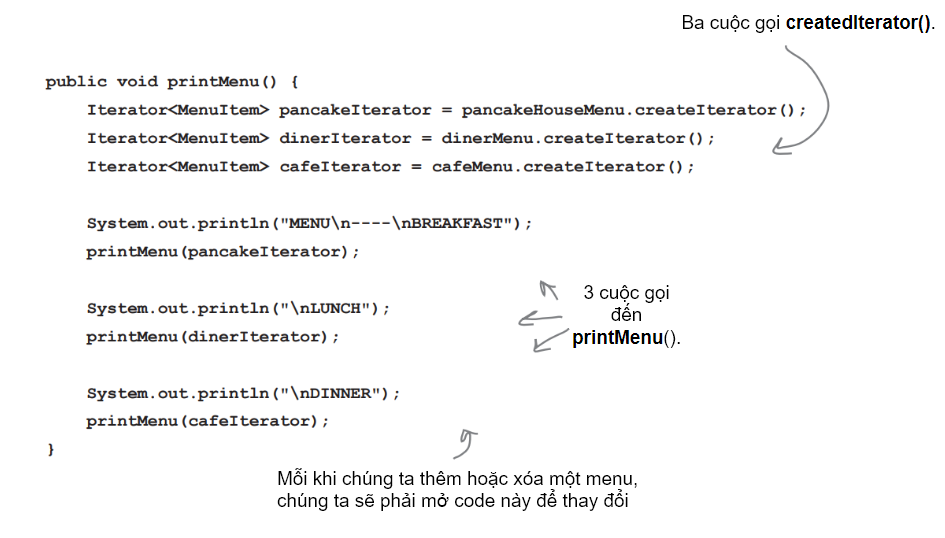




1. Cải tiến Waitress

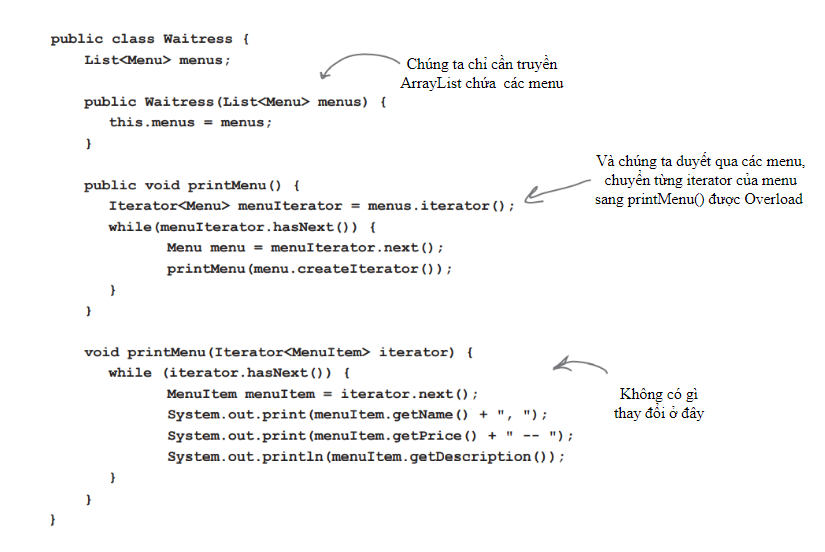
Waitress đã đi được một chặng đường dài, nhưng vẫn gặp phải vấn đề đó là có đến ba lần gọi **printMenu()**.

Hãy xem lại phương thức **printMenu()**, mỗi khi chúng ta thêm một menu mới, chúng ta sẽ phải viết thêm code mới. Điều này đã “vi phạm nguyên tắc đóng mở”.

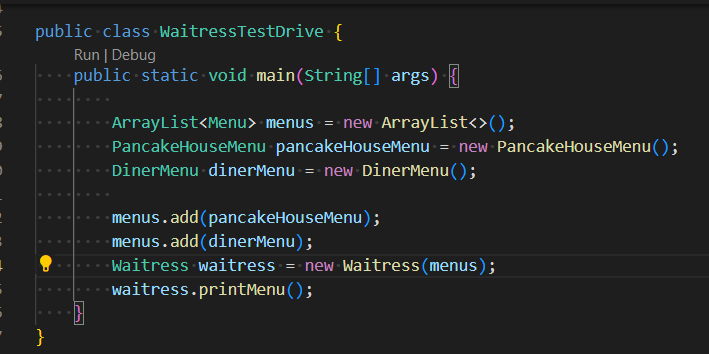


Chúng ta đã thực hiện một công việc tuyệt vời là tách rời việc thực hiện menu và đưa phép lặp vào một Iterator. Nhưng chúng ta vẫn đang xử lý các menu với các đối tượng độc lập, riêng biệt – chúng ta cần một cách để quản lý chúng cùng nhau.

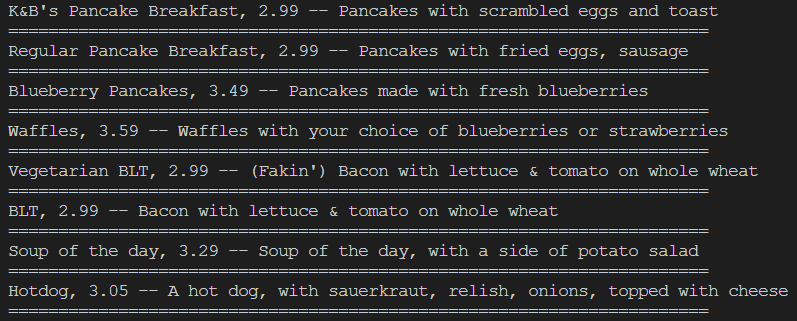
Để khắc phục việc gọi 3 lần đến **printMenu()** cần phảiđóng gói các Menu vào 1 ArrayList. Sau đó lấy Iterator của ArrayList để lặp qua từng menu. Code trong Waitress sẽ đơn giản và xử lí bất kì số lượng menu nào được thêm vào.



Sau đây là lớp TestDrive



Chương trình chạy ra kết quả

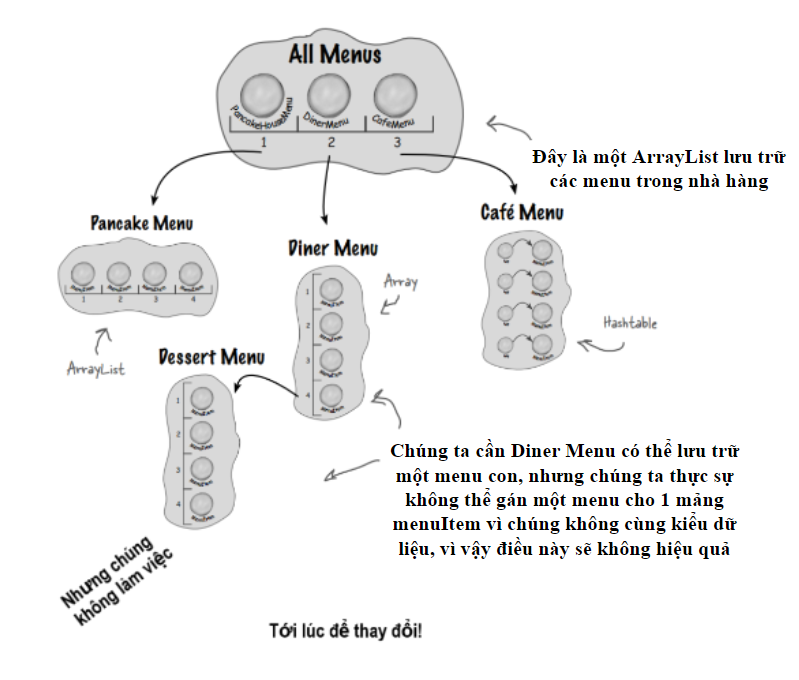


Bây giờ Waitress đã khắc phục được việc gọi **printMenu()** nhiều lần. Mặc dù các tên Menu đã mất, nhưng chúng ta vẫn có thể tự thêm tên vào mỗi menu.

1. Composite
2. Vấn đề mới

Chương trình muốn thêm một menu con, món tráng miệng (DessertMenu) vào menu DinerMenu thì phải làm thế nào ?

Sẽ thật tuyệt nếu chương trình có thể biến DessertMenu thành một phần của DinerMenu, nhưng khi đó DessertMenu sẽ không hoạt động ở mẫu Iterator.

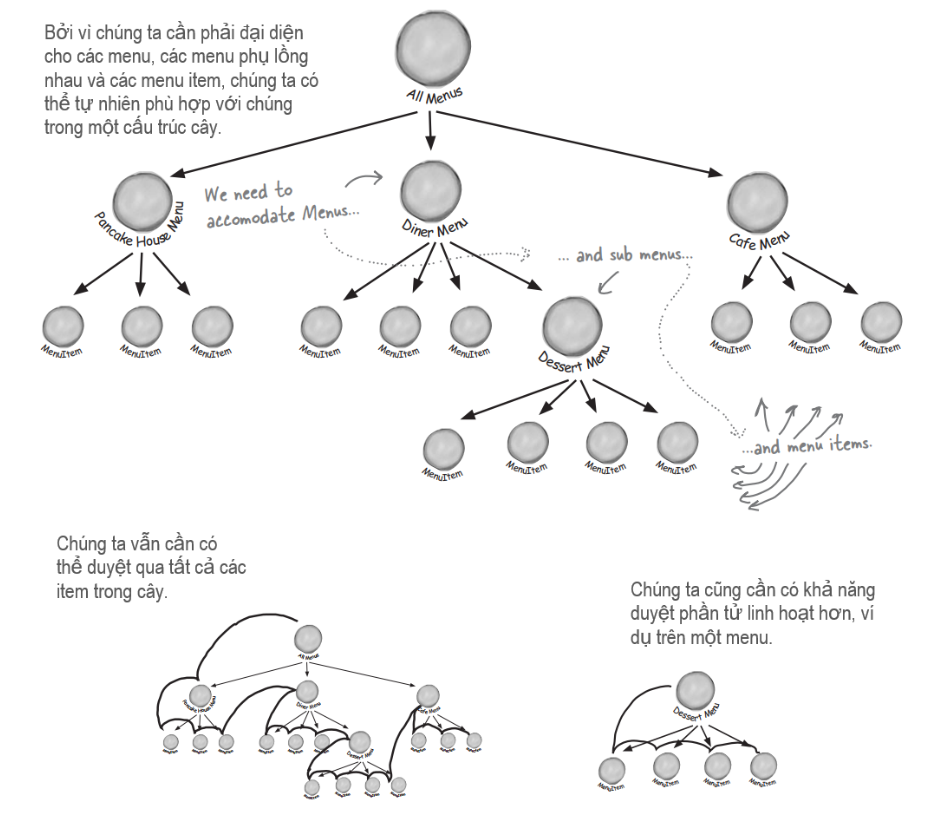


1. Cách giải quyết vấn đề ?

Đã đến lúc đưa ra quyết định để làm lại thành một thứ đủ chung để làm việc được trên tất cả các menu (cả menu phụ – sub menu).

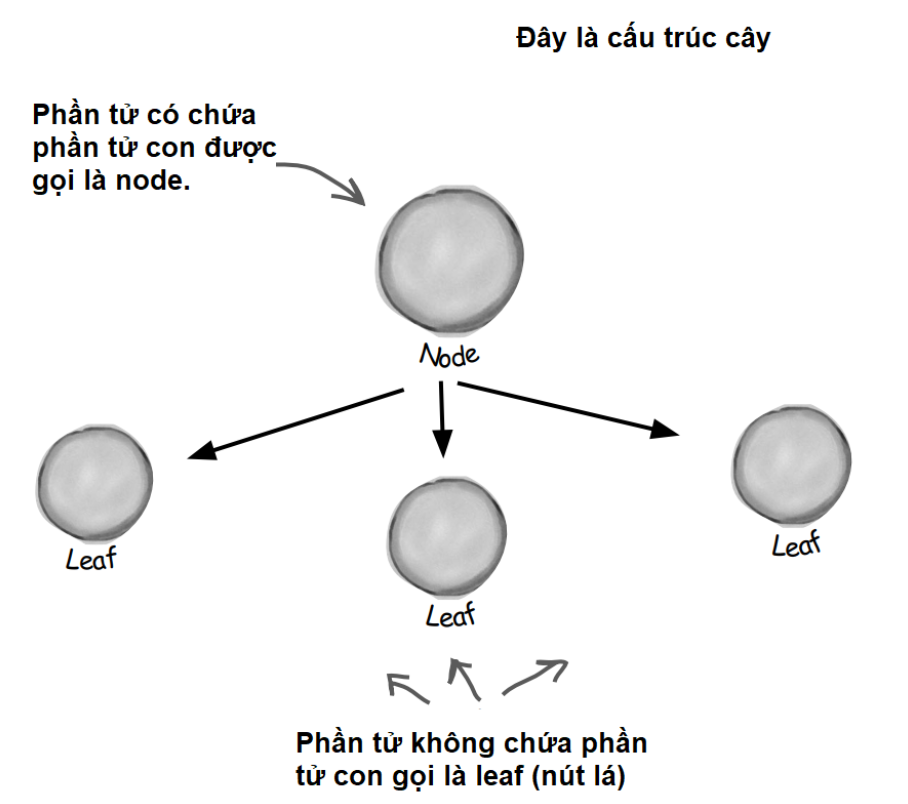
Vậy chúng ta cần thay đổi những gì?

* Chúng ta cần một số loại cấu trúc cây sẽ chứa các **menu**, **menu con** và các **menu item**.
* Chúng ta cần đảm bảo sẽ có một cách để duyệt qua các phần tử trong mỗi menu ít nhất là thuận tiện như những gì chúng ta đang làm bây giờ với các iterator.
* Chúng ta có thể cần phải duyệt qua các phần tử một cách linh hoạt hơn. Chẳng hạn, chúng ta có thể chỉ cần duyệt trên **dessert menu** của Diner, hoặc chúng ta có thể cần duyệt toàn bộ **Diner menu**, bao gồm cả **dessert submenu**.



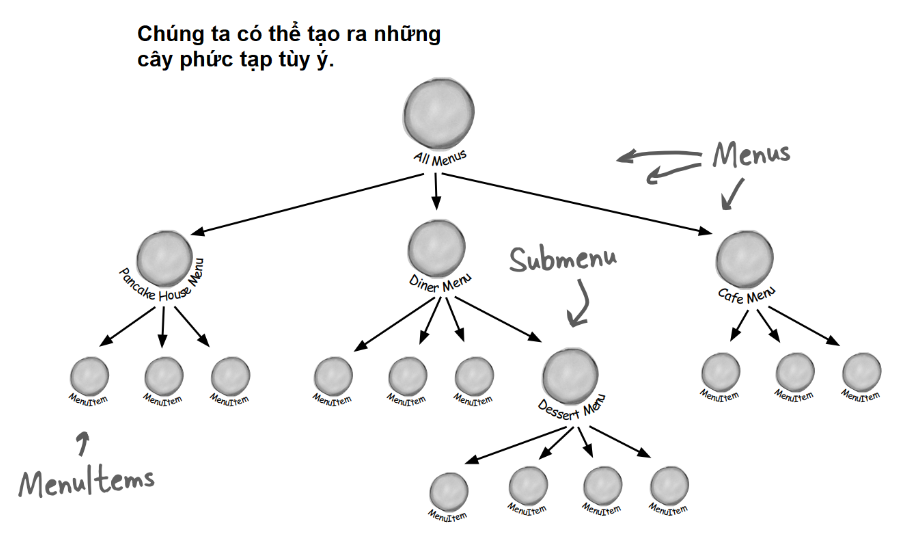
1. Định nghĩa Composite Pattern

Composite Pattern cho phép bạn kết hợp các đối tượng thành các cấu trúc cây để thể hiện hệ thống phân cấp toàn bộ. Composite cho phép client xử lý các đối tượng riêng lẻ và các thành phần của các đối tượng một cách thống nhất

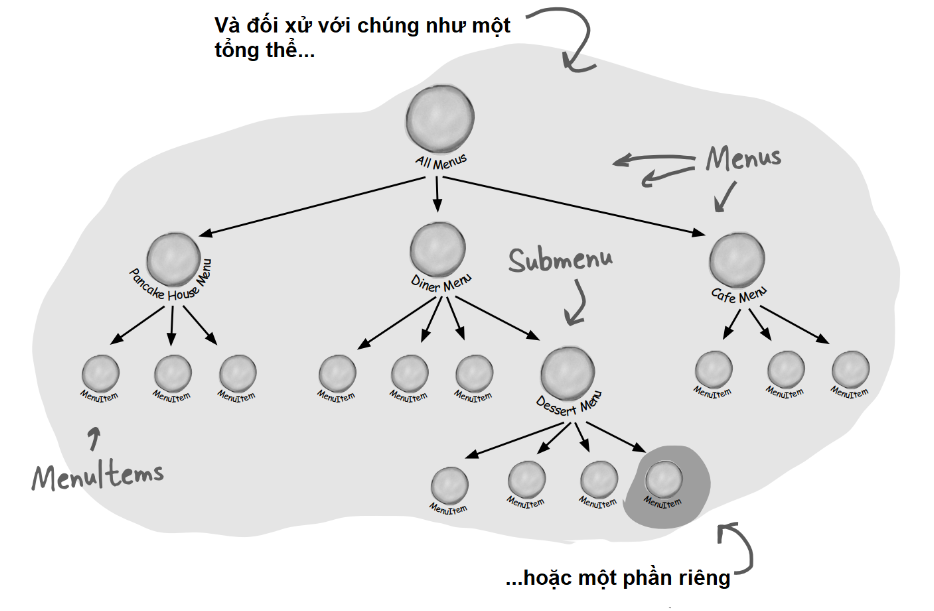


Mẫu này cho chúng ta một cách để tạo ra một cấu trúc cây có thể xử lý một nhóm các **menu** và các **menu item** lồng nhau trong cùng một cấu trúc. Bằng cách đặt các **menu** và các **item** trong cùng một cấu trúc, tạo ra một hệ thống phân cấp toàn bộ; nghĩa là, một cây có các đối tượng được tạo thành từ các thành phần (**menu** và các **menu item**) nhưng có thể được coi như một tổng thể, giống như một **menu lớn**.

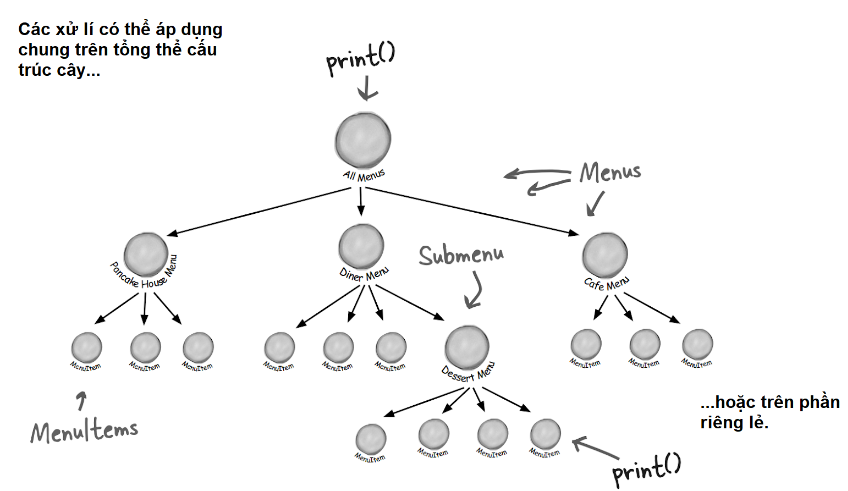




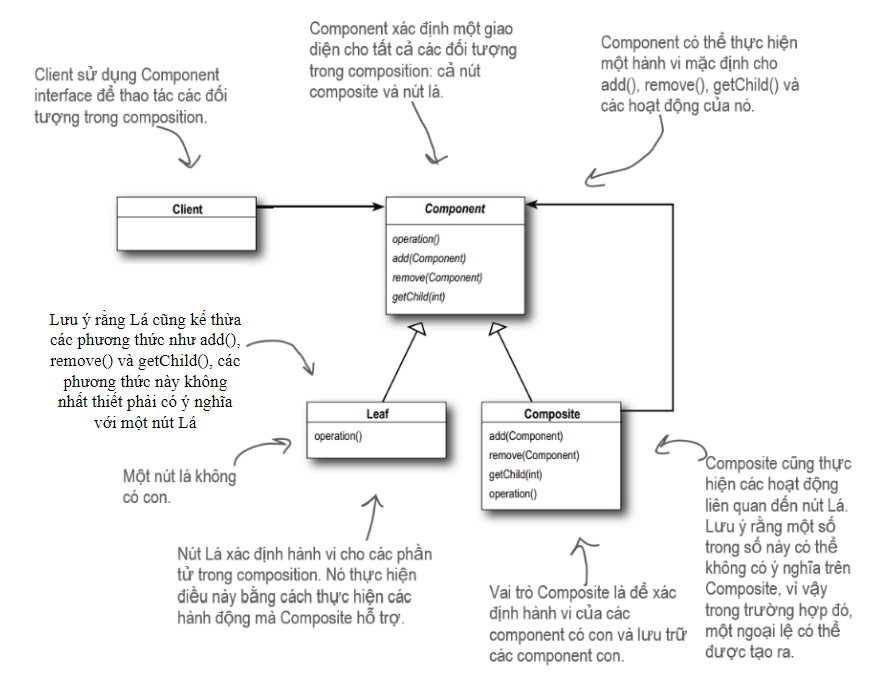
Composite Pattern cho phép chúng ta xây dựng các cấu trúc của các đối tượng ở dạng cây có chứa cả các tập hợp của các đối tượng và các đối tượng riêng lẻ dưới dạng các nút (node).



Sử dụng một cấu trúc composite, chúng ta có thể áp dụng các hoạt động tương tự trên cả đối tượng composite và các đối tượng riêng lẻ. Nói cách khác, trong hầu hết các trường hợp, chúng ta có thể bỏ qua sự khác biệt giữa các composite của các đối tượng và các đối tượng riêng lẻ.

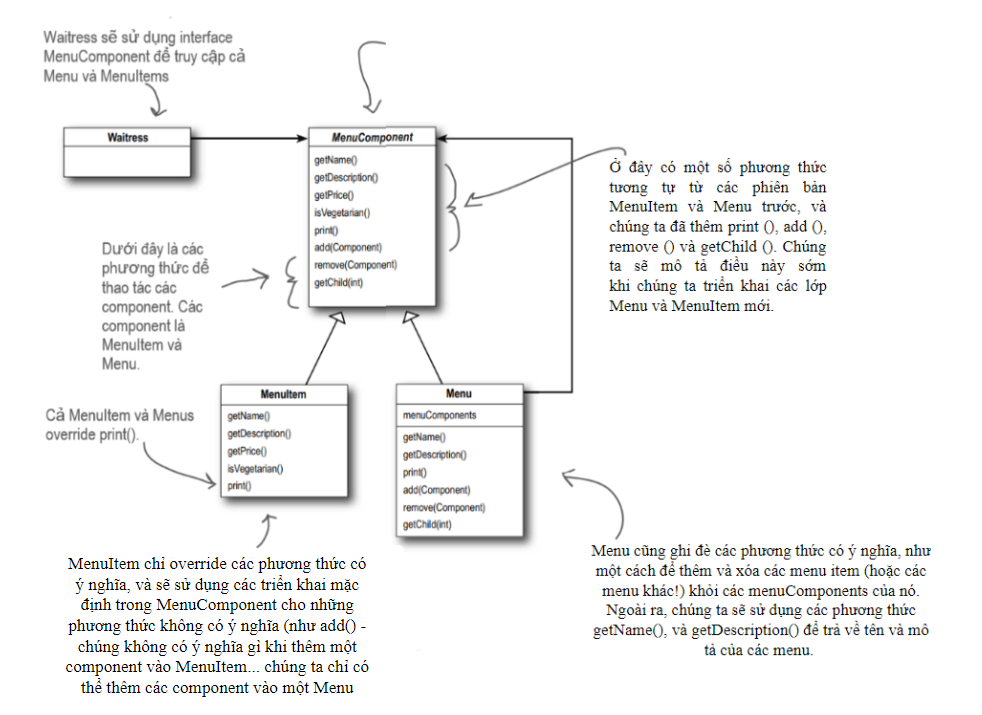


Sơ đồ Class



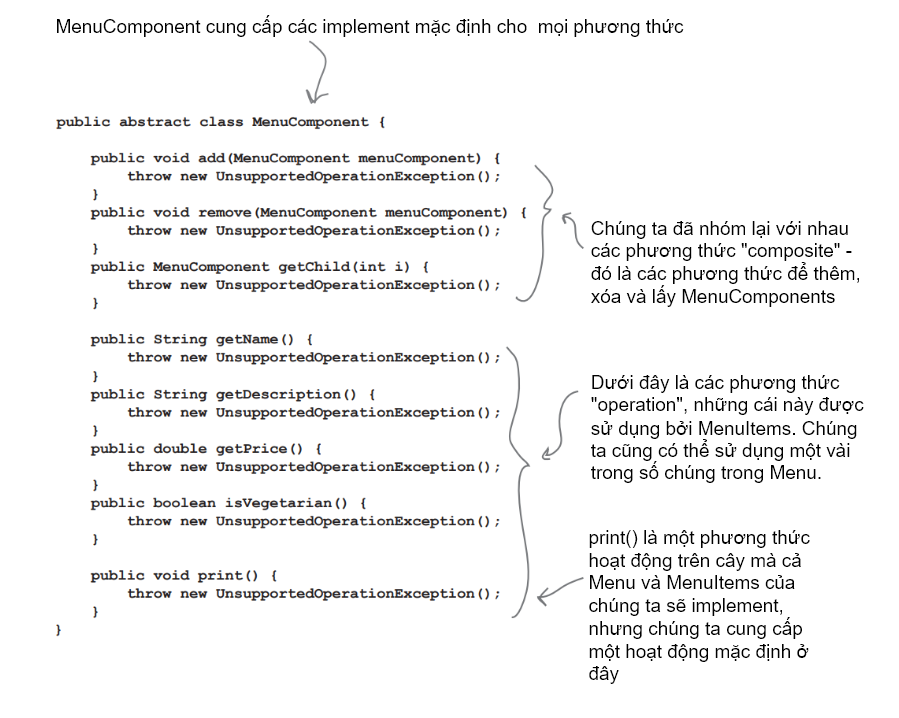
1. Thiết kế Menu với Composite pattern

Một số phương thức có thể không có ý nghĩa khi gọi trên một menu item hoặc một menu, nhưng chúng ta có thể đối phó với điều đó, và chúng ta sẽ làm trong chốc lát. Nhưng bây giờ, hãy xem qua một bản phác thảo về cách các menu sẽ chuyển thành cấu trúc Composite Pattern:



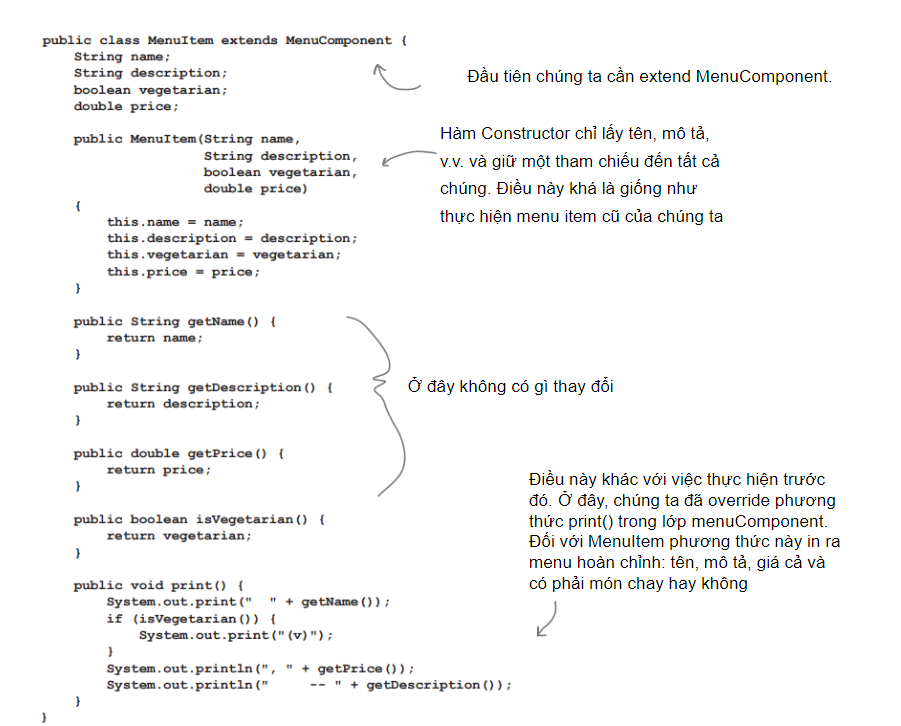
1. CÀI ĐẶT MENU COMPONENT

* Chúng ta sẽ bắt đầu với lớp trừu tượng MenuComponent; vai trò của menu component là cung cấp interface cho các nút lá và các nút composite. Hiện tại chúng tôi sẽ cung cấp một triển khai mặc định của các phương thức để nếu MenuItem (lá) hoặc Menu (composite) không muốn thực hiện một số phương thức (ví dụ nút lá sẽ không cần phương thức getChild()) chúng có thể cài đặt cho chúng một số hành vi mặc định.
* Tất cả các component phải implement giao diện MenuComponent; tuy nhiên, vì các node và leaf có vai trò khác nhau, chúng tôi không thể cài đặt cách triển khai mặc định cho mỗi phương thức đều có ý nghĩa. Đôi khi điều tốt nhất có thể làm là ném runtime exception.
* Vì một số phương thức chỉ có ý nghĩa đối với MenuItem và một số phương thức chỉ có ý nghĩa đối với Menu, nên việc triển khai mặc định sẽ là UnsupportedOperationException. Theo cách đó, nếu MenuItem hoặc Menu không hỗ trợ thao tác, chúng không phải làm gì cả, chúng chỉ có thể kế thừa việc thực hiện mặc định.



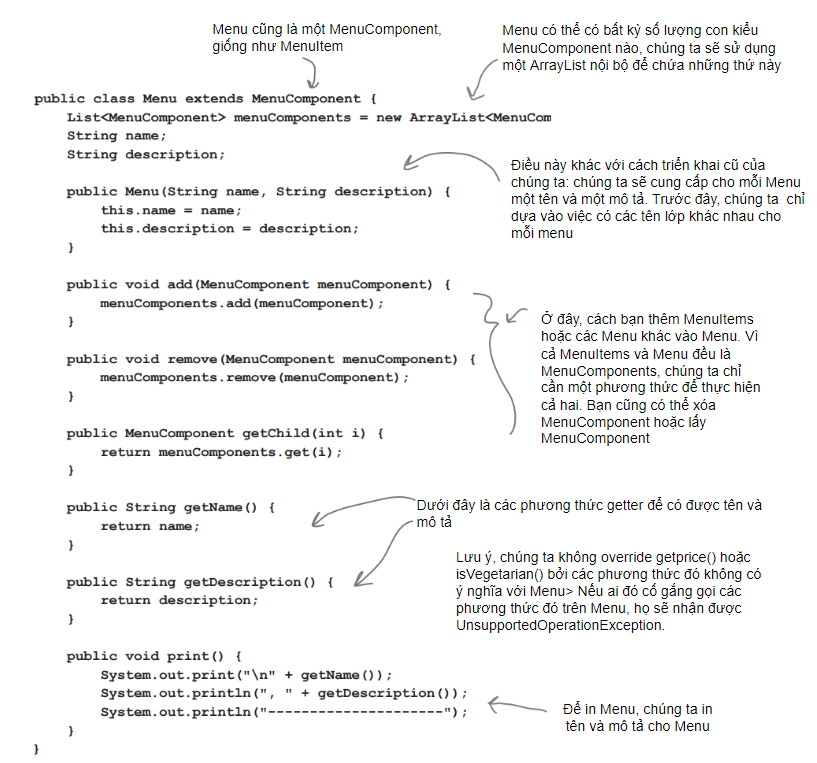
1. CÀI ĐẶT MENU ITEM

Viết cho lớp MenuItem. Hãy nhớ rằng, đây là lớp lá trong Composite diagram và nó thực hiện hành vi của các phần tử trong composite.



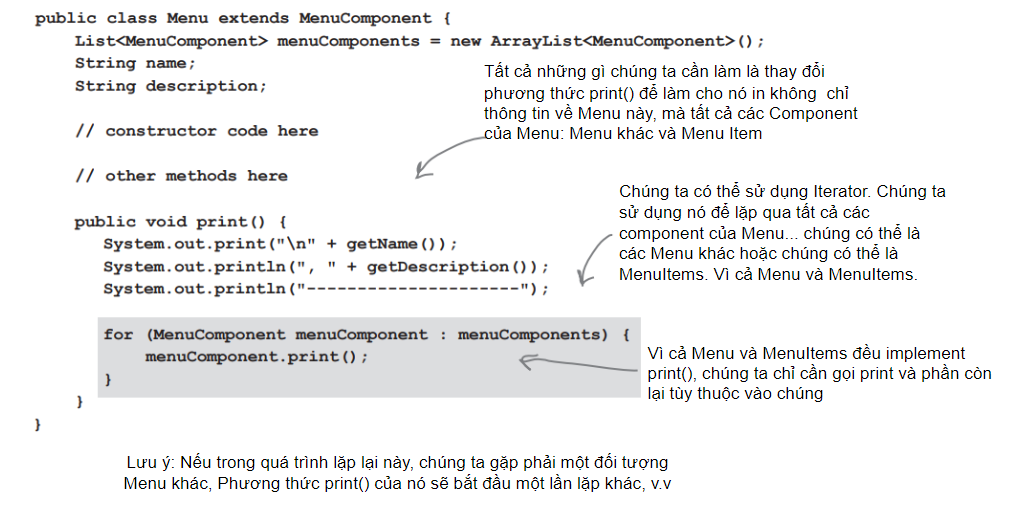
1. CÀI ĐẶT COMPOSITE MENU

Lớp composite có thể chứa MenuItem (leaf) hoặc các Menu (node) khác. Có một vài phương thức từ MenuComponent, lớp này không overide getprice() và isVegetarian(), vì những phương thức này không có nhiều ý nghĩa đối với Menu.



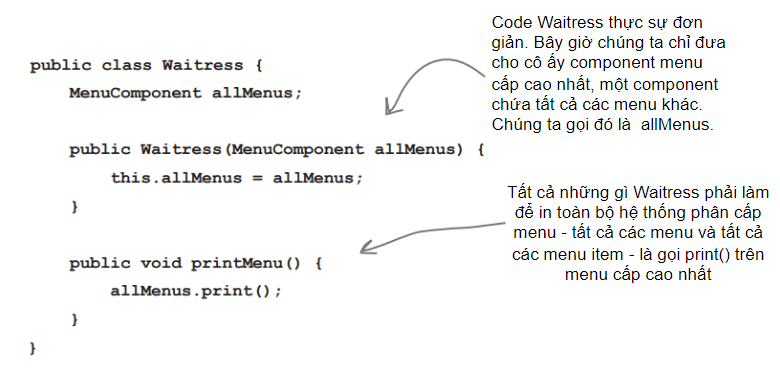
* Tuy nhiên đối với việc implement print(). Chúng ta chỉ nhận được một tên và mô tả menu đơn giản. Chúng ta không nhận được bản in của COMPOSITE.
* Vì menu là một composite và nó chứa cả Menu Item và các Menu khác, nên phương thức print() sẽ in mọi thứ mà nó chứa. Nếu không như vậy, chúng ta sẽ phải lặp lại toàn bộ composite và tự in từng mục. Điều đó sẽ làm sai mục đích của việc có một cấu trúc composite.
* Việc implement print() rất dễ dàng vì chúng ta có thể dựa vào từng component để có thể tự in.

1. SỬA PHƯƠNG THỨC PRINT()

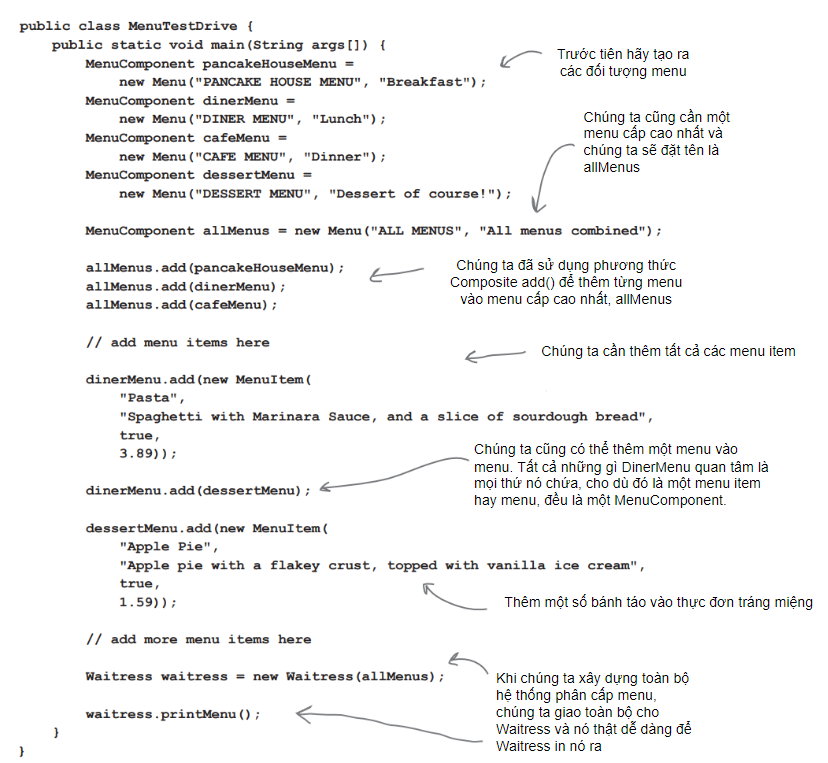


1. Sửa lại Class Waitress và test drive

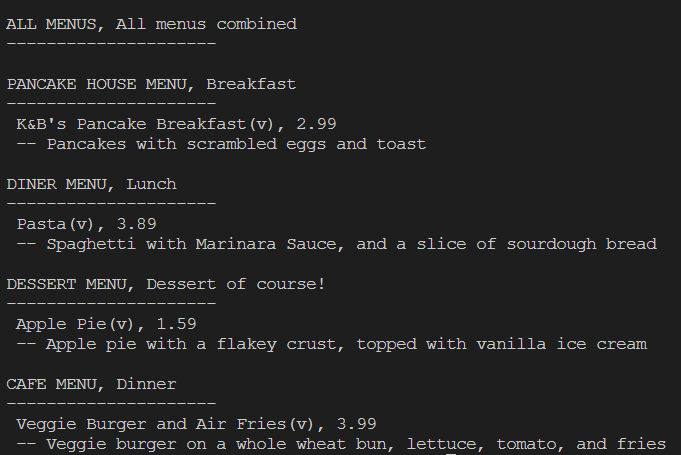
Cập nhật code Waitress trước khi chạy chương trình



Không giống như phiên bản trước, chúng ta sẽ xử lý tất cả việc tạo menu trong test drive. Chương trình có thể yêu cầu mỗi đầu bếp cung cấp cho chúng ta menu mới, nhưng hãy test chương trình trước. Đây là code:



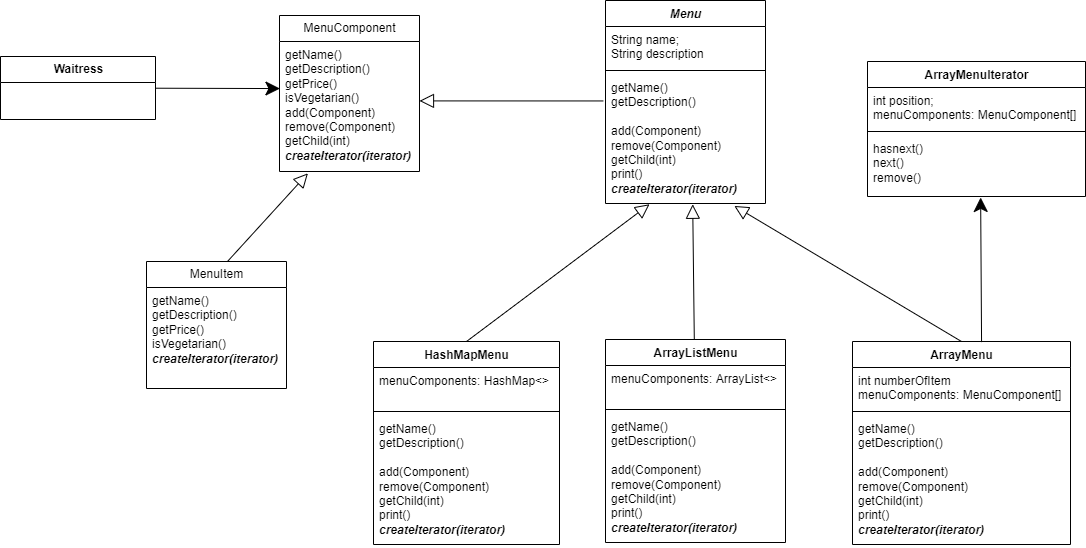
Kết quả



1. Composite kết hợp với Iterator

### 10.1 Sơ đồ lớp

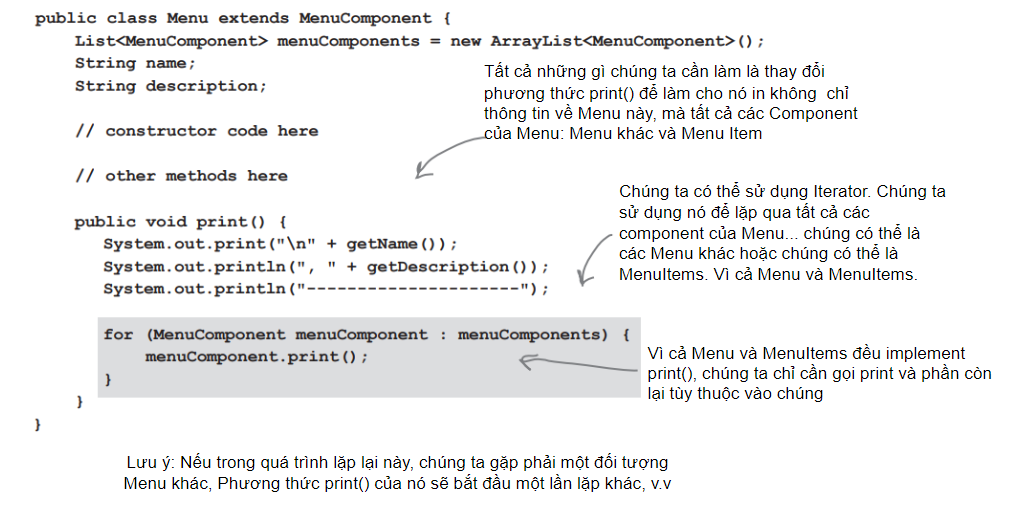
Đây là sơ đồ lớp kết hợp Composite với Iterator sử dụng 3 collection khác nhau



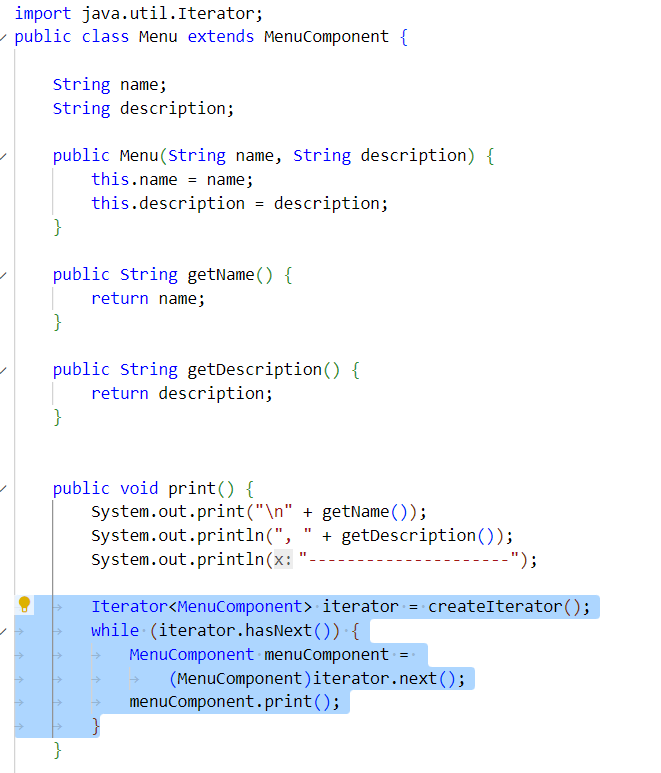
### 10.2 Sửa class Menu

Thay vì sử dụng foreach ở print() trong lớp Menu thì chúng ta sẽ sử dụng Iterator

Trước khi sử dụng Iterator



Sau khi sử dụng Iterator

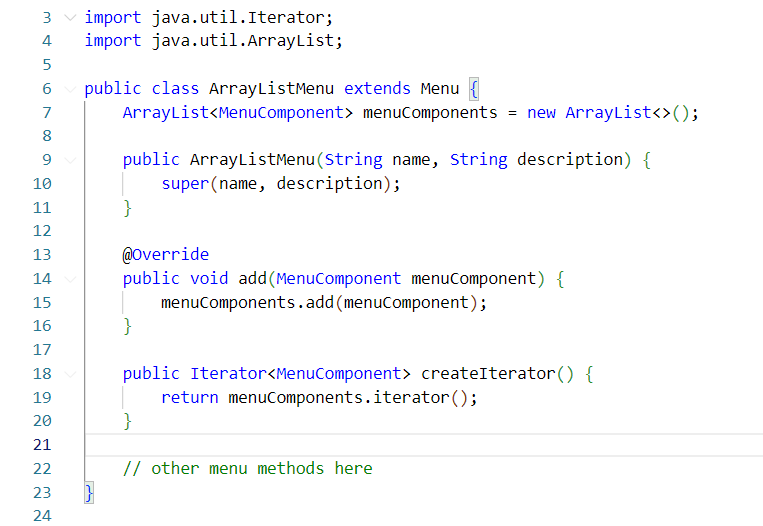


Việc thay đổi phương thức print() có sử dụng Iterator sẽ giúp chúng ta có thể duyệt phần tử của bất cứ Collection nào miễn là Collection đó tạo ra iterator.

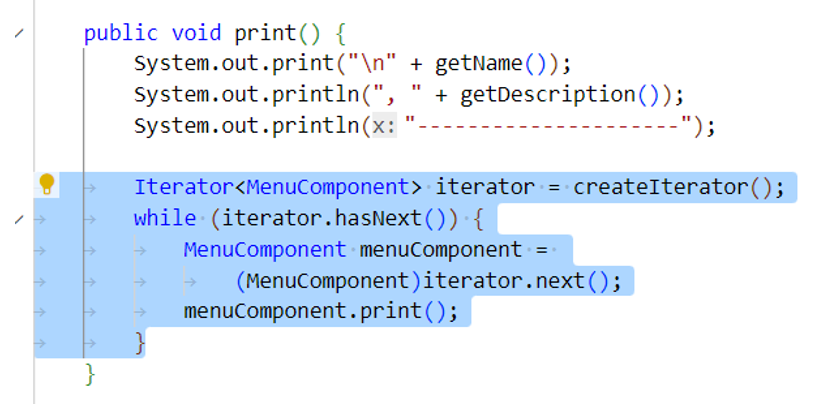
Ở đây chúng ta định nghĩa class Menu là class cha, các class sử dụng Collection sẽ thừa kế class Menu. Do đó class Menu không cần add(), remove(), getChild() vì collection ArrayList đã bị xóa . Các phương thức này sẽ override ở lớp con tùy theo Collection được sử dụng

### 10.3 Sử dụng collection ArrayList

Sau khi áp dụng Iterator cho class Menu thì bây giờ chúng ta có thể cài nhiều collection khác nhau để lưu 1 menu cụ thể. Hãy thử cài ArrayList được đặt tên class là ArrayListMenu. Class này sẽ extend class Menu và override phương thức add(). Mỗi collection sẽ có cách thêm khác nhau, do đó việc override phương thức add() sẽ tạo sự khác biệt mà class cha không có.



Contructor đã được thừa kế từ class Menu. Do đó mà khi tạo ra đối tượng, chúng ta có thể truyền tên Menu và mô tả cho Menu cụ thể. Phương thức add() sẽ sử dụng ArrayList để lưu các MenuComponent. Sau đó đưa ArrayList vào Iterator bằng createIterator(). Như vậy là chúng ta đã có iterator được tạo ra từ ArrayList. Lúc này việc in ra các phần tử trong iterator sẽ do print() của class Menu thực hiện.



### 10.4 Sử dụng collection HashMap

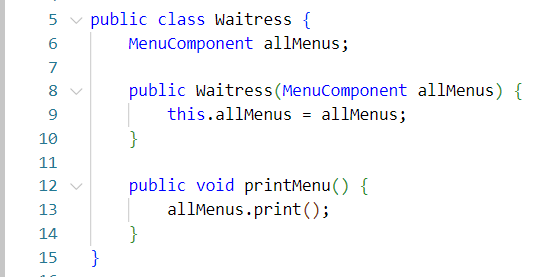
Hãy thử 1 collection HashMap để thấy được việc kết hợp mẫu Iterator vào mẫu Composite lợi hại như thế nào. Chỉ cần tạo ra class dùng HashMap cho kế thừa class Menu và thay đổi 1 chút là chúng ta đã có thể tạo ra Iterator mà không cần phải thay đổi print() của Menu.



Như vậy là xong, giờ chỉ cần test xem chương trình hoạt động tốt không.

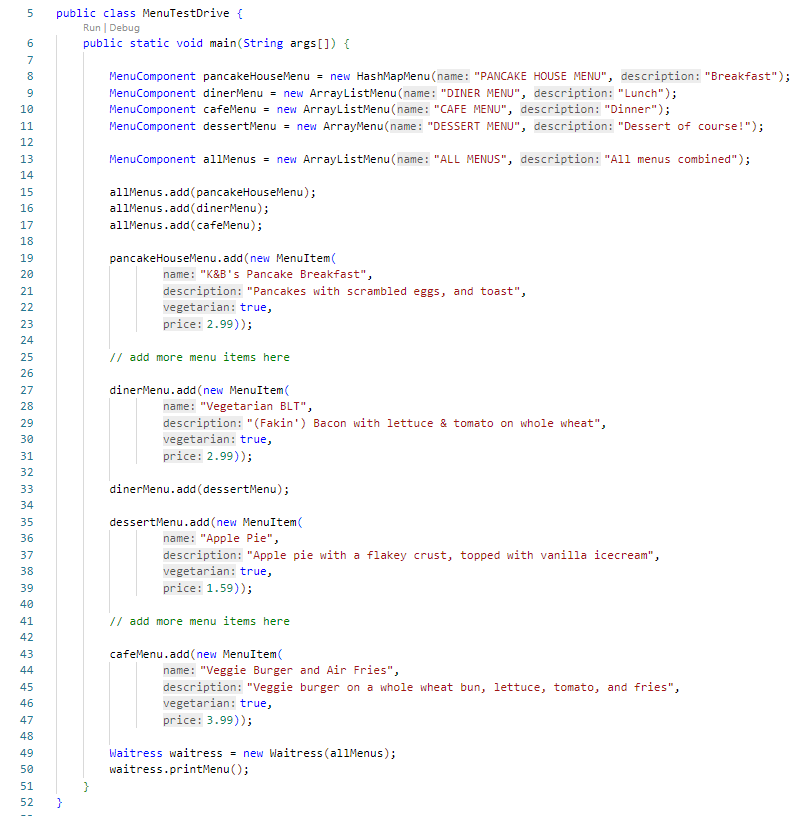
### 10.5 Test drive

Chúng ta đã thay đổi khá nhiều nhưng dù vậy thì class Waitress và test drive cũng không phải sửa code.

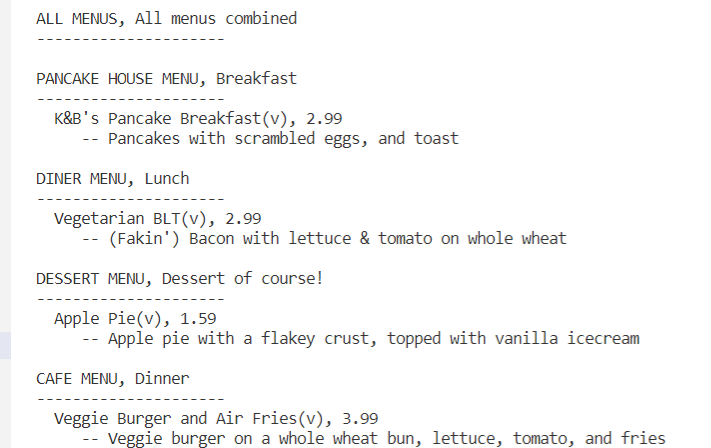


Không có sự thay đổi so với dùng foreach

Class test drive thì thay đổi 1 chút. Sự thay đổi này là do chúng ta muốn add thêm menu hoặc thay đổi collection chứa các phần tử của menu đó.

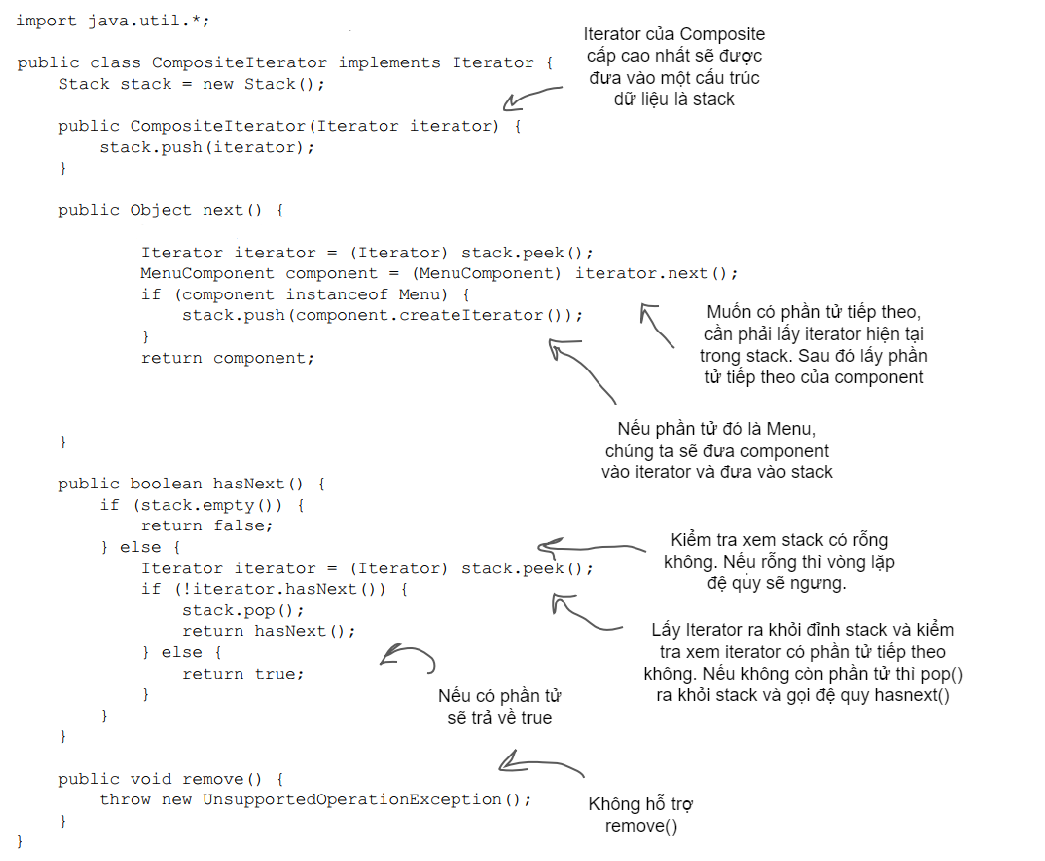


Chúng ta có thể thấy **MenuComponent pancakeHouseMenu** lưu đối tượng **MenuItem** vào **HashMap**, còn **MenuComponent** **dinnerMenu** sẽ lưu **MenuItem** vào **ArrayList**. Tức là 2 menu có tên là **Pancake House Menu** và **Dinner Menu** lưu các món ăn vào 2 **collection** khác nhau. Điều này cho thấy mẫu Iterator hoạt động trên mẫu Composite. Và sau đây là kết quả cho thấy điều đó:



### 10.6 CompositeIterator

**CompositeIterator** là một iterator HẤP DẪN. Công việc của nó là duyệt phần tử trên các **MenuItem** trong component và đảm bảo tất cả các sub-Menu (và sub-sub-Menu, v.v.) cũng được duyệt. Sau đây là code



**CompositeIterator** giúp chúng ta duyệt phần tử linh hoạt hơn. Thay vì chỉ duyệt được tất cả phần tử thì **CompositeIterator** giúp cho chương trình duyệt được các phần tử được đặt điều kiện. Chẳng hạn như in toàn bộ món chay.

Để áp dụng CompositeIterator chúng ta sẽ thêm phương thức createCompositeIterator cho MenuComponent với tham số truyền vào là Iterator



Việc áp dụng phương thức createCompositeIterator() sẽ thể hiện ở phần in toàn bộ món chay

### 10.7 In toàn bộ món chay

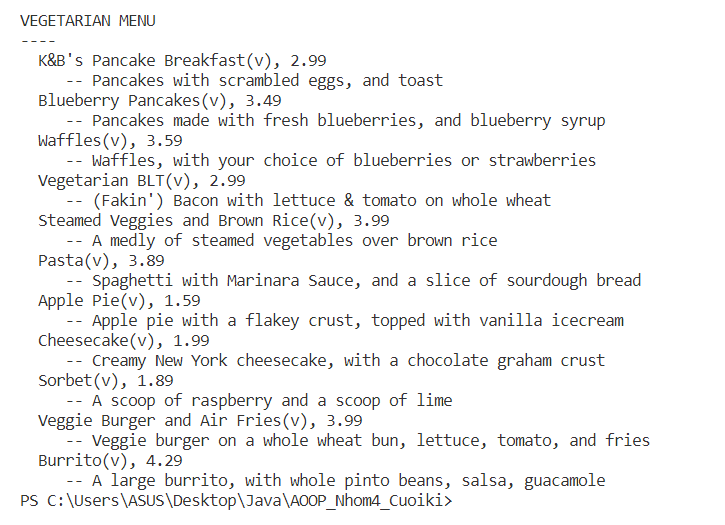
Hiện tại chúng ta chỉ mới in được toàn bộ menu bao gồm Menu (Composite) và MenuItem (Leaf). Vậy làm thế nào để class Waitress in ra được món chay của MenuItem (Leaf) mà không in ra Menu (Composite). Trước tiên chúng ta sẽ cung cấp cho Waitress 1 phương thức printVegetarianMenu() để in món chay.



Khi áp dụng CompositeIterator vào các Collection Menu sẽ giúp cho việc duyệt phần tử linh hoạt hơn, có thể trả về component chứa composite hoặc leaf

print() chỉ được gọi trên MenuItem, không bao giờ gọi trên Composite. Đó là vì class Composite không có isVegetarian(), lúc này sẽ xảy ra lỗi. Nhưng thật may mắn, chúng ta đã try catch để mỗi khi xảy ra lỗi thì sẽ bị ném ngoại lệ. Do vậy mà Waitress vẫn in ra được món chay, còn Composite xảy ra lỗi thì bị ném ngoại lệ.

Sau đây là kết quả của test drive



1. Tổng kết
2. Tóm tắt Iterator

* **Một Iterator** cho phép truy cập vào một phần tử Collection mà không làm lộ cấu trúc bên trong.
* **Một Iterator** đảm nhận công việc duyệt phần tử trên một Collection và gói nó trong một đối tượng khác.
* Khi sử dụng **Iterator**, chúng ta sẽ giảm bớt trách nhiệm cho việc truyền tải dữ liệu.
* **Iterator** cung cấp một interface chung để duyệt qua các item của Collection, cho phép bạn sử dụng đa hình khi viết code.

1. Tóm tắt Composite

* **Composite Pattern** cung cấp một cấu trúc để chứa cả các leaf object và composite con.
* **Composite Pattern** cho phép client xử lý composite và các leaf object một cách thống nhất.
* Component là bất kỳ đối tượng nào trong cấu trúc cây của Composite. Các Component có thể là **Composite** khác hoặc các **nút lá**.
* Có nhiều sự đánh đổi trong thiết kế khi cài đặt **Composite pattern**. Bạn cần phải cân bằng sự transparent và sự an toàn với nhu cầu của bạn.

**LỜI CẢM ƠN**

Nhóm em xin gửi lời cảm ơn đến Ban giám hiệu trường Đại học Gia Định vì đã tạo điều kiện cho nhóm em được tìm hiểu và thấy được tầm quan trọng của mẫu thiết kế qua bộ môn Lập trình hướng đối tượng nâng cao.

Xin cảm ơn giảng viên bộ môn – Thầy Lê Huỳnh Phước đã hướng dẫn tận tình, giúp đỡ nhóm em trong quá trình hoàn thành báo cáo này.

-HẾT-