Universidad de los Andes
Facultad de Ingeniería

Departamento de Ingeniería de Sistemas y Computación Diseño y Programación Orientada a Objetos ISIS-1226

TALLER 5 – PATRONES

Juan David Rios Nisperuza, 202215787, jd.riosn1@uniandes.edu.co

Patrón a estudiar: Iterator

Repositorio donde se implementa el patrón: https://github.com/banerjeesamrat/Virtual-Coffee-Shop.git

•Información general del proyecto: para qué sirve, cuál es la estructura general del diseño, qué grandes retos de diseño enfrenta (i.e. ¿qué es lo difícil?). Deben incluir la URL para consultar el proyecto.

Propósito: El propósito del proyecto es crear una cafetería virtual independiente donde los clientes puedan personalizar sus pedidos de café eligiendo entre las opciones de filtro y café instantáneo, seleccionando varios complementos y calculando el precio en consecuencia. El proyecto pretende simular la experiencia de pedir y personalizar café en un entorno virtual.

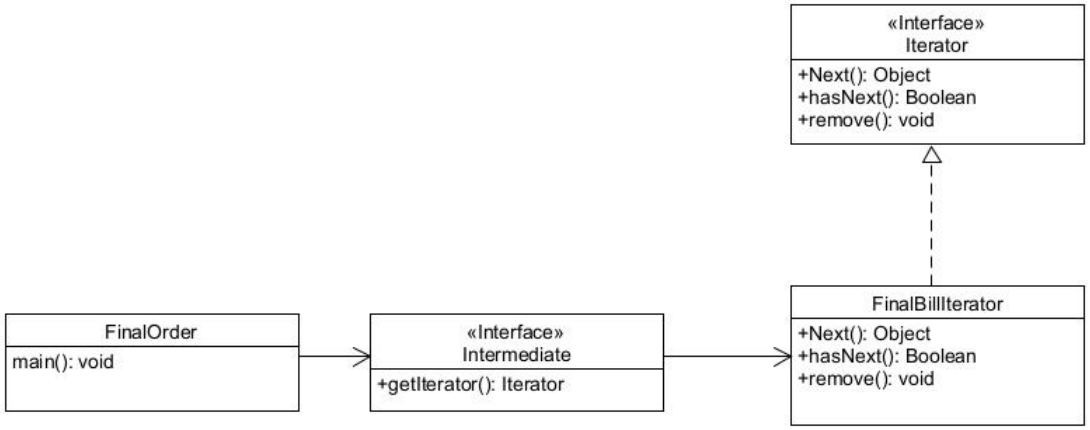
Estructura de diseño general: el proyecto utilizará el patrón del decorador para personalizar el pedido de café de cada cliente. Este patrón permite la adición dinámica de comportamientos o características a objetos de café individuales, como agregar un trago de caramelo o crema batida. Además, el patrón iterador se utilizará para recorrer el pedido del cliente, lo que permitirá un fácil acceso y manipulación de los artículos de café.

Desafíos de diseño: Los principales desafíos de diseño para este proyecto pueden incluir:

Implementar el patrón decorador de manera efectiva para manejar varias combinaciones de complementos y opciones de personalización, gestionar el proceso de pedido y calcular correctamente el coste total, incluidos los impuestos y los precios de los artículos individuales y desarrollar una interfaz fácil de usar usando JavaFX para representar la cafetería virtual y permitir que los clientes interactúen con el sistema sin problemas.

• Información y estructura del fragmento del proyecto donde aparece el patrón. No se limite únicamente a los elementos que hacen parte del patrón: para que tenga sentido su uso, probablemente va a tener que incluir elementos cercanos que sirvan para contextualizarlo.

A continuación, se muestra el fragmento el cual se hace el uso del patrón iterator:



Se presentará una breve explicación de cada una de las clases involucradas:

Iterator: Esta interfaz define los métodos que deben ser implementados en las clases que la utilizan para permitir la iteración sobre una colección de elementos de manera secuencial.

FinalBillIterator: Esta clase proporciona una implementación concreta de la interfaz Iterator. En ella, se declaran e inicializan las variables "bill" y "pos". "bill" almacena el costo total acumulado de la factura, mientras que "pos" indica la posición actual del iterador en la lista "FinalBill". La clase contiene los siguientes métodos:

- "hasNext" verifica si existen más elementos en la lista "FinalBill" para iterar.

- "next" devuelve el siguiente elemento de la lista y actualiza el costo total acumulado utilizando el método "cost" de cada objeto.

- "remove" elimina el elemento actual de la lista "FinalBill", siempre y cuando el iterador no esté antes del primer elemento.

FinalOrder: Esta clase representa un pedido final y ofrece funcionalidades relacionadas con la gestión de la lista de pedidos. La clase incluye un campo estático "FinalBill", que es un ArrayList de objetos de tipo Coffee. Este campo almacena los elementos del pedido final. Los métodos incluidos son:

- "addToList" se utiliza para agregar un objeto Coffee a la lista "FinalBill".

- "orderQuantity" muestra el tamaño de la lista "FinalBill", es decir, la cantidad de elementos en la lista.

- "clear" elimina todos los elementos de la lista "FinalBill", dejándola vacía.

- "getIterator" devuelve una instancia de FinalBillIterator, permitiendo la iteración sobre los elementos de "FinalBill".

.

• Información general sobre el patrón: qué patrón es y para qué se usa usualmente.

Iterator es un patrón de diseño de comportamiento que te permite recorrer elementos de una colección sin exponer su representación subyacente (lista, pila, árbol, etc.).

La idea central del patrón Iterator es extraer el comportamiento de recorrido de una colección y colocarlo en un objeto independiente llamado iterador.

Los iteradores implementan varios algoritmos de recorrido. Varios objetos iteradores pueden recorrer la misma colección al mismo tiempo.

Además de implementar el propio algoritmo, un objeto iterador encapsula todos los detalles del recorrido, como la posición actual y cuántos elementos quedan hasta el final. Debido a esto, varios iteradores pueden recorrer la misma colección al mismo tiempo, independientemente los unos de los otros.

Normalmente, los iteradores aportan un método principal para extraer elementos de la colección. El cliente puede continuar ejecutando este método hasta que no devuelva nada, lo que significa que el iterador ha recorrido todos los elementos.

Todos los iteradores deben implementar la misma interfaz. Esto hace que el código cliente sea compatible con cualquier tipo de colección o cualquier algoritmo de recorrido, siempre y cuando exista un iterador adecuado. Si necesitas una forma particular de recorrer una colección, creas una nueva clase iteradora sin tener que cambiar la colección o el cliente.

Estructura:

La interfaz Iteradora declara las operaciones necesarias para recorrer una colección: extraer el siguiente elemento, recuperar la posición actual, reiniciar la iteración, etc.

Los Iteradores Concretos implementan algoritmos específicos para recorrer una colección. El objeto iterador debe controlar el progreso del recorrido por su cuenta. Esto permite a varios iteradores recorrer la misma colección con independencia entre sí.

La interfaz Colección declara uno o varios métodos para obtener iteradores compatibles con la colección. Observa que el tipo de retorno de los métodos debe declararse como la interfaz iteradora de forma que las colecciones concretas puedan devolver varios tipos de iteradores.

Las Colecciones Concretas devuelven nuevas instancias de una clase iteradora concreta particular cada vez que el cliente solicita una. Puede que te estés preguntando: ¿dónde está el resto del código de la colección? No te preocupes, debe estar en la misma clase. Lo que pasa es que estos detalles no son fundamentales para el patrón en sí, por eso los omitimos.

El Cliente debe funcionar con colecciones e iteradores a través de sus interfaces. De este modo, el cliente no se acopla a clases concretas, permitiéndote utilizar varias colecciones e iteradores con el mismo código cliente.

Normalmente, los clientes no crean iteradores por su cuenta, en lugar de eso, los obtienen de las colecciones. Sin embargo, en algunos casos, el cliente puede crear uno directamente, como cuando define su propio iterador especial.

• Información del patrón aplicado al proyecto: explicar cómo se está utilizando el patrón dentro del proyecto.

El patrón iterador se está aplicando para recorrer la lista de artículos del pedido de café del cliente. Este patrón permite obtener cada artículo individualmente de manera ordenada y sin revelar la estructura interna de la lista.

La implementación del patrón iterador proporciona una interfaz que define métodos para acceder a los elementos del pedido en secuencia. Por ejemplo, el método hasNext() que verifica si hay más elementos en el pedido, un método next() que devuelve el siguiente elemento y el método remove();

Al utilizar el iterador, se puede recorrer el pedido del cliente en un bucle, accediendo a cada artículo y realizando las operaciones necesarias. Esto simplifica el código al no tener que preocuparse por los detalles de implementación de la lista de artículos.

Además, el uso del patrón iterador permite una fácil manipulación de los artículos del pedido. Se pueden realizar operaciones como la suma total del pedido, la búsqueda de un artículo específico, la eliminación de un artículo no deseado o cualquier otra manipulación requerida, ya que se puede acceder a los elementos individualmente mediante el iterador.

• ¿Por qué tiene sentido haber utilizado el patrón en ese punto del proyecto? ¿Qué ventajas tiene?

El uso del patrón Iterator en este punto del proyecto tiene sentido debido a las siguientes ventajas:

-Encapsulación de la estructura de datos: El patrón Iterator permite encapsular los detalles del trabajo con una estructura de datos compleja, como la lista de artículos del pedido de café. Esto evita que el cliente tenga que conocer los detalles internos de la estructura y le proporciona métodos simples y claros para acceder a los elementos de la colección. Esto facilita el uso y la comprensión del código por parte del cliente.

-Protección de la colección: Al utilizar un iterador, se proporciona una capa de protección adicional a la colección de artículos del pedido. El cliente no puede acceder directamente a la colección ni realizar acciones descuidadas o maliciosas sobre ella. El iterador controla el acceso a los elementos y garantiza que se realicen operaciones válidas y seguras.

• ¿Qué desventajas tiene haber utilizado el patrón en ese punto del proyecto?

La desventaja de haber utilizado el patrón Iterator en este proyecto en particular es que puede resultar innecesario debido a que la aplicación y la estructura del proyecto no son complejas, y las colecciones de datos son sencillas. En casos donde las colecciones son simples y directamente accesibles, utilizar un iterador puede agregar una capa adicional de complejidad y sobrecarga.

Además, en algunos casos, recorrer directamente los elementos de colecciones especializadas puede ser más eficiente que utilizar un iterador. El uso de un iterador puede introducir un mayor costo computacional debido a la necesidad de mantener el estado del iterador y realizar comprobaciones adicionales en cada iteración.

Dado que el proyecto no presenta una complejidad significativa en términos de estructura de datos y recorrido de elementos, podría haberse optado por soluciones más simples y directas, como utilizar bucles y operaciones básicas de manipulación de listas, en lugar de implementar el patrón Iterator.

Sin embargo, es importante tener en cuenta que el uso del patrón Iterator puede brindar flexibilidad y abstracción en proyectos más complejos, donde se requiere un recorrido más sofisticado de las colecciones y operaciones avanzadas de manipulación de elementos. En estos casos, el patrón Iterator puede ser una solución adecuada y eficiente.

• ¿De qué otras formas se le ocurre que se podrían haber solucionado, en este caso particular, los problemas que resuelve el patrón?

Cuando se trabaja en un proyecto de menor escala o complejidad, puede no ser necesario implementar un patrón específico para recorrer una lista. En su lugar, se puede acceder y manipular los elementos de la lista directamente utilizando métodos y funciones básicas.

Este enfoque más directo puede ahorrar tiempo y esfuerzo, especialmente cuando se trata de proyectos sencillos en los que no se requiere una lógica complicada o un procesamiento intensivo de datos. Simplemente se puede iterar sobre los elementos de la lista uno por uno y realizar las operaciones necesarias en cada uno de ellos.

En proyectos más complejos, donde se necesita un patrón específico para recorrer la lista o se requiere un procesamiento más avanzado, puede ser recomendable utilizar estructuras de control más sofisticadas o algoritmos específicos. Sin embargo, en proyectos simples, la capacidad de recorrer una lista directamente sin un patrón específico puede simplificar el código y facilitar su comprensión y mantenimiento.