Taller 5

David Leonardo Manrique Lesmes- 201913129

Link del repositorio: https://github.com/design-pattern-list/iterator-pattern.git

Información general del proyecto:

El autor del proyecto es Michael Waanen, quien, según sus repositorios, se especializa en la creación de proyectos basados en patrones de diseño. El proyecto en cuestión se presenta como un gestor de contactos, brindando al usuario la capacidad de crear un nuevo contacto que será almacenado en la estructura designada. En este contexto, se observan diversas estructuras utilizadas y segmentadas en secciones específicas. Posteriormente, se habilita al usuario para revisar su lista de contactos organizada de manera ascendente y, además, visualizar la misma lista ordenada según la fecha de nacimiento, con esta última disposición en orden descendente para mostrar primero a los contactos de mayor edad.

El proceso completo implica la iteración, organización y recorrido de la información proporcionada por el usuario. Es relevante destacar que el proyecto adopta un enfoque de estructuración de tipo Modelo-Vista-Controlador (MVC), lo que posibilita la segmentación de las diferentes responsabilidades relacionadas con el usuario, la interfaz de consola y la gestión de la memoria. Aunque el proyecto no es de gran envergadura, cumple con la implementación del patrón de diseño seleccionado.

Información y estructura del fragmento del proyecto donde aparece el patrón:

La implementación del patrón iterador se encuentra en la carpeta denominada "iterators" en el repositorio, la cual contiene tres archivos .java distintos, cada uno demostrando la aplicación del patrón. El primer archivo, titulado "Icontacbookiterator", presenta una implementación inicial del patrón. Aunque el código es conciso, satisface los requisitos de implementación. En la Figura 1, se muestra el código correspondiente, evidenciando el uso de la función hasNext para determinar si el elemento actual tiene un sucesor. A continuación, se define el siguiente elemento como un objeto del tipo Contacts, perteneciente a los objetos creados en el proyecto. Para contextualizar, la clase Contacts incorpora el nombre, el año y el número. Posteriormente, se reinicia la búsqueda en los elementos del arreglo. Además, se observa que esta implementación invoca a esta clase como una interfaz, indicando su posible utilización en otros esquemas de ordenamiento.

```java
package com.michielswaanen.iterators;

import com.michielswaanen.objects.Contact;

public interface IContactBookIterator {

    boolean hasNext();
    Contact next();
    void reset();
}
```

Figura 1

Ahora se analiza el archivo llamado: AscendingNumbersIterator. El código de este es el siguiente:

```java
public class AscendingNameIterator implements IContactBookIterator {

    private List<Contact> sortedContactBook;
    private int currentPosition;

    public AscendingNameIterator(List<Contact> contactList) {
        this.sortedContactBook = this.sortAscendingByName(contactList);
        this.currentPosition = 0;
    }

    private List<Contact> sortAscendingByName(List<Contact> contactList) {
        List<Contact> sortedContactBook = new LinkedList<>(contactList);

        for(int i = 0; i < sortedContactBook.size(); ++i) {
            for(int j = i + 1; j < sortedContactBook.size(); j++) {
                if(sortedContactBook.get(i).getName().compareTo(sortedContactBook.get(j).getName()) > 0) {
                    Contact tempContact = sortedContactBook.get(i);
                    sortedContactBook.set(i, sortedContactBook.get(j));
                    sortedContactBook.set(j, tempContact);
                }
            }
        }

        return sortedContactBook;
    }

    @Override
    public boolean hasNext() {
        return this.currentPosition < this.sortedContactBook.size();
    }

    @Override
    public Contact next() {
        if(this.hasNext())
            return this.sortedContactBook.get(this.currentPosition++);
        return null;
    }

    @Override
    public void reset() {
        this.currentPosition = 0;
    }
```

Lo inicial que se destaca es que esta categoría incorpora la interfaz Icontacbookitterator. En la representación visual, se evidencia que sobrescribe los métodos que han sido previamente definidos en la interfaz. Con respecto al método hasNext, se observa que establece una comparación entre la posición actual del objeto y el tamaño de la lista de contactos. En cuanto al método contacto, se aprecia que, si el método hasNext está presente, devuelve el elemento correspondiente a la posición más 1 del objeto. Posteriormente, en el método de reset, su función consiste en restablecer la posición actual a cero. Finalmente, en el método sortAscendingByName, se lleva a cabo la organización de los elementos.

Información general sobre el patrón:

El concepto de patrón iterador implica la acción de recorrer una estructura de datos, siendo comúnmente aplicado para atravesar listas y visualizar su contenido en la consola. Sin embargo, las funciones de un iterador no se limitan únicamente a la presentación de datos; también abarcan diversas opciones, como la eliminación o modificación de elementos dentro de la estructura. Una ventaja significativa de los iteradores radica en su eficaz capacidad de encapsulamiento. En este sentido, al utilizar un iterador, al usuario no le resulta relevante conocer los detalles de su implementación; simplemente se espera que el iterador recorra la estructura designada de manera efectiva.

Información del patrón aplicado al proyecto:

Este enfoque del iterador resulta especialmente valioso en el contexto del proyecto, ya que facilita la navegación y manipulación de las estructuras de datos. La capacidad de verificar la existencia de elementos siguientes, así como de definir y actualizar el elemento actual según sea necesario, brinda una flexibilidad significativa en el procesamiento de la información almacenada.

En la práctica del proyecto, se aplican dos casos específicos de uso del iterador. En el primer escenario, se emplea para recorrer la lista de contactos, extrayendo los nombres y organizándolos en orden ascendente. Este proceso permite una clasificación ordenada y accesible de la información. Por otro lado, en el segundo caso, el iterador se vuelve a utilizar para recorrer la lista de contactos, esta vez extrayendo los años de nacimiento. Esta operación posibilita la organización de la lista de contactos de manera descendente según la edad, contribuyendo a un análisis más detallado de la información.

En conclusión, el patrón iterador no solo mejora la eficiencia del recorrido de estructuras, sino que también se implementa de manera estratégica en el proyecto para realizar operaciones específicas de extracción y organización de datos, proporcionando así una herramienta fundamental para la gestión eficaz de la información.

Ventajas y desventajas:

Este diseño permite determinar la existencia de elementos siguientes y continuar el proceso de recorrido, además de facilitar la definición del elemento actual como el siguiente en la lista. Al considerar esto, se evidencia que el patrón se utiliza de manera eficaz y coherente. En ambos casos, se emplea la iteración para alcanzar los objetivos planteados y resolver la problemática en cuestión.

No obstante, una desventaja destacada del uso de iteradores surge cuando se aplican a estructuras extensas. La eficiencia del iterador puede deteriorarse considerablemente, dependiendo de la complejidad de la estructura y el enfoque de implementación del código. Esta desventaja es más notable en situaciones donde se requiere un procesamiento extenso, lo que hace que el iterador pueda volverse lento y tedioso. Por lo tanto, aunque el iterador proporciona una visión general y sencilla para recorrer estructuras de datos, no siempre resulta la opción

óptima en términos de eficiencia en un programa, dependiendo de los objetivos específicos y la complejidad del código asociado.

Otras Alternativas:

Otra alternativa para abordar la misma problemática abordada por el patrón iterador consiste en seleccionar algún tipo de estructura de datos abstracta para almacenar la información relevante del proyecto. A partir de esta elección, se define la manera en que se llevarán a cabo los recorridos en el manejador de contactos. Sin embargo, esta estrategia puede no ser la más adecuada en todos los escenarios, ya que en ciertas circunstancias es más conveniente realizar el recorrido de una manera específica. Sin una estructura generalizada, esta tarea puede volverse complicada. Por otra parte, se podrían explorar otros patrones como el Factory Method para la creación de objetos en una superclase, los cuales se van modificando a partir de subclases para el manejo de los datos del manejador. Asimismo, se podría considerar el uso del patrón Visitor para separar el algoritmo con respecto a los objetos encontrados, facilitando de esta manera el recorrido de los elementos.