**Análisis del patrón de diseño: Iterador**

**Enlace:** <https://github.com/design-pattern-list/iterator-pattern.git>

**Información General:**

El proyecto fue creado por Michael Waanen. El cual según sus repositorios se dedica a crear proyectos basados en los patrones de diseño. El proyecto seleccionado es una especie de manejador de contactos. Este manejador de contactos le permite al usuario crear el contacto. El cual se va a almacenar en la estructura señalada. En este caso, vamos a encontrar varias estructuras usadas y divididas en varias secciones. Luego, le permite al usuario mirar su memoria de contactos organizados ascendentemente. También, le muestra al usuario la misma lista de contactos, pero esta vez organizada por la fecha de nacimiento. Esta fecha se va a organizar descendentemente, es decir, va a mostrar a los mayores primero. Todo esto es iterado u organizado o recorrido a través de la información que puede ingresar el usuario. Cabe aclarar que el proyecto usa un tipo de estructuración de tipo MVC, la cual permite dividir en ciertas partes todo lo que tiene que ver con el usuario, la consola y la memoria. Este no es un proyecto tan grande, pero cumple con la implantación del patrón de diseño seleccionado.

**Información Estructural:**

Texto

Descripción generada automáticamente Ahora bien, luego de haber visto una información general del proyecto, pasamos al patrón de diseño. Como se había dicho al principio, del documento, el patrón de diseño escogido fue el iterador. Mas adelante se va a tocar más a fondo lo que significa y para qué sirve. Ahora se va a mostrar cómo es su implementación y función en el proyecto seleccionado. La implementación la encontramos en la carpeta, del repositorio, llamada: iterators. En esta carpeta se encentran tres archivos.java diferentes, donde en cada uno se ve implementado el patrón. El primer archivo se llama: Icontacbookiterator. En este podemos ver una primera implementación del patrón. En este caso, el código es pequeño, pero cumple con la implementación. En la imagen, podemos ver la parte más fundamental del código. En este podemos ver que utiliza la función de hasNext. La cual permite saber si el elemento actual tiene un siguiente un elemento siguiente. Luego, define el siguiente elemento como tipo Contacts. Este tipo Contacts, hace parte de los objetos creados en el proyecto. Para contextualizar, esta clase Contacts tiene el nombre, tiene el año y el número. Después, reinicia la búsqueda en los elementos en el arreglo que esta. Además, podemos ver que en esta implementación llama a esta clase de tipo interfaz. Lo cual es indicativo que lo va a utilizar en otros ordenamientos. En segundo lugar, tenemos el siguiente archivo llamado: AscendingNumbersIterator. Este como podemos ver es una implantación de la interfaz Icontacbookitterator. 

Texto

Descripción generada automáticamenteEsto significa que va a tener ciertos valores iguales a los definidos en esta clase. En la imagen podemos ver que sobrescribe los métodos que fueron definidos en la clase interfaz. EN este caso, vemos que en el método hasNext, define una comparación entre la posición actual del objeto y el tamaño de la lista de contactos. En el método contacto, podemos ver que, si existe el método hasnext, retorna el elemento que pertenece a la posición más 1 del objeto. Luego, en el método de reset lo que hace es poner la posición actual en cero.

Texto

Descripción generada automáticamente

Por último, en el método sortAscendingByNameorganiza de la manera tradicional con el for tradicional. Donde va haciendo comparaciones con los elementos para organizarlos. En ultimo lugar, esta el tercer archivo llamado: OldestContactIterator. Aquí nuevamente sobrescribe, como lo había hecho antes, la clase interfaz. Y nuevamente vuelve a organizar los contactos por su parámetro de edad y año dados en la interfaz gráfica.

**Información Patrón:**

El patrón que se usa es un iterador este patrón es usado por la mayoría de programadores y es de los primeros que se estudia cuando se empieza con algún lenguaje de programación, esto se debe a que su funcionamiento es fácil de comprender y cumple una función muy bastante usada, esta función consiste en recorrer una estructura de datos que puede ser simple o compleja, normalmente lo que se recorre con un iterador es una lista y se puede mostrar por consola lo que esta lista tiene, estas no son las únicas funciones que tienen un iterador dentro del mismos se puede llevar a cabo una variedad de opciones dentro de las que existen quitar elementos o cambiarlos. Algo muy bueno en cuanto a los iteradores se refiere es la buena capacidad de encapsulamiento que tienen ya que de por si cuando se use un iterador al usuario no le interesa su implementación solo que este recorra la estructura designada. Por último, una ventaja que tiene este patrón es que permite recorrer bastantes de diversas formas estructuras, lo que lo convierte en una herramienta bastante maleable.

**Información Patrón Aplicado:**

En el caso del proyecto el patrón se usa de una manera muy útil. Esta manera es para la escénica del patrón y es recorrer una estructura. Para este caso se le pregunta a la estructura, de cierta manera, si existe un elemento siguiente en el arreglo o estructura. Esto hecho para lograr saber si se llego al final y no seguir recorriendo. O a su vez, para poder continuar recorriendo la estructura. Después, de haberle preguntado al programa si existía un elemento siguiente o ya se había finalizado la estructura. Luego de esto define, el elemento siguiente como elemento del tipo de la clase que se necesita y lo define como elemento actual del sistema. Es decir, que si existe un elemento siguiente se pasa a esa posición y se posiciona como el elemento actual del sistema. Por último, lo que hace en el método reste es lograr reiniciar de cierta manera la posición de la estructura. Es decir, lograr mantener su posición de manera relativa en la estructura. Ahora bien, luego de conocer las funcionalidades del código, se tiene que hablar de cómo se usan en el proyecto. En este proyecto se usan para dos casos. El primer caso, se utiliza para poder recorrer toda la lista de contactos que se tenga, sacando un valor de cada uno y comparándolo con otros valores de otros contactos, para este caso. En el primero lo que se utiliza es, que cuando recorra la lista usando el iterador saque de cada contacto su nombre e ir organizándolo ascendentemente. Luego, en el segundo caso. Nuevamente recorre la lista peor esta vez sacando de cada contacto el valor de año de nacimiento para comprarlo con los otros que aparecen en la lista. De esta manera organizándolos de mayor a menor. Es decir, en este caso el iterador es usado para recorrer la estructura de contactos, extraer valores de cada elemento y compararlos para crear una nueva estructura organizada.

**Ventajas:**

Ahora bien, ¿tiene sentido utilizar el patrón en el proyecto? La respuesta a esta pregunta realmente resulta un poco obvia por lo que se ha descrito anteriormente. Sin embargo, resulta necesario explicar el porqué es importantísimo usar este patrón y otros más. El patrón resulta importante en esta parte del proyecto, ya que permite hacer la funcionalidad de recorrer sobre una estructura. En este caso, permite recorrer una estructura de contactos. Este patrón permite, saber si existe un siguiente y continuar el recorrido. Además, de permitir definir el elemento actual como el siguiente de la lista. Teniendo esto en cuenta, se puede ver que el patrón es usado de una forma óptima y que tiene sentido, lo que se logra en ambos casos es darle uso a una iteración para lograr ambos objetivos y resolver el problema planteado.

Una desventaja y la principal que tiene los iteradores es cuando estos revisan una estructura muy extensa, dependiendo de la estructura que se use el iterador puede demorarse demasiado tiempo y ser tedioso esto depende de cuales sean los objetivos del problema y el código que se le dé al mismo, debido a esto es que el iterador es de las primeras formas que se aprende para recorrer estructuras de datos, este patrón te da una vista general y simple, pero usarlo siempre no va a ser lo más optimo en cuanto a un programa se refiere.

**Otras Alternativas:**

Otro método para solucionar el mismo problema en el cual se desenvuelve el patrón iterador es escoger algún tipo de estructura de datos abstracta para almacenar la información pertinente del proyecto y definir a partir de esta la forma como se realizarán los recorridos del manejador de contactos. No obstante, esta estrategia podría no resultar la más apropiada en todos los casos, siendo que en ciertas circunstancias es más conveniente realizar el recorrido de una manera específica o de otra, y sin tener una estructura generalizada, se dificulta mucho esta tarea. Por otro lado, se podrían utilizar otros patrones como Factory Method para la creación de objetos en superclase que se van modificando a partir de subclases para el manejo de los datos del manejador, mientras que se podría aplicar Visitor paras la separación del algoritmo respecto a los objetos encontrados en este facilitando así el recorrido de los elementos.

**Mapa UML de Estructura del Programa**

Diagrama

Descripción generada automáticamente

En la imagen anterior tenemos el mapa UML del patrón implementado que en general cumple con la estructura base para un iterador. Se posee una clase iterador, en este caso, *IContactBookIterator* que define las operaciones que debe ser posible realizar sobre una colección (en este caso los contactos). Por otro lado, se tienen los iteradores concretos que implementan las operaciones definidas en la interfaz. Estas están dadas por *OldestContactIterator* y *AscendingNameIterator* que se usan para la creación de las funciones que permiten realizar tales operaciones. También se tiene una clase de tipo agregador que permite adicionar objetos al iterador y que en este caso está dada por *IContactBook.* Por otro lado, se posee la clase que implementa las opciones dadas por esta clase de tipo agregador definida en *ContactBook,* y los propios objetos se crean a partir de la clase *Contact.*

**Modelo General para la Estructura del Iterador**

**Diagrama

Descripción generada automáticamente**

**[1]**

**Referencia:**

[1] Programación en Castellano… (2021). Patrones de Diseño (XVII): Patrones de Comportamiento – Iterator. Recuperado de

https://programacion.net/articulo/patrones\_de\_diseno\_xvii\_patrones\_de\_comportamiento\_iterator\_1020