Ejercicio 1

|  |  |
| --- | --- |
| Nombre | Interprete |
| Propósito | Admite la interpretación de instrucciones o leguaje ya definido para un propósito especifico, la notación debe de ser precisa y definirse en términos de gramática. |
| Problema | Utilizar el patrón interprete para el manejo de archivos XML, a la hora de guardar y leer el archivo. |
| Solución | Se crea una estructura especifica en el archivo XML que es un árbol donde el nodo padre contiene muchos nodos hijos y estos nodos hijos tienen nodos que son los atributos. Con la implementación de patrón se creó una estructura y gramática especifica. |
| Participantes y colaboradores | Cliente (XmlFile): Construye el árbol y es el encargado de disparar la acción al interprete.  Contexto (Contexto): Objeto que es usado para almacenar información global quesera utilizada por el interpretador.  AbstractExpression (Expresion): Interfaz con la estructura mínima de una expresión  TerminalExpression (Flotilla, Chofer, Ruta, RutaDeBus): se denomina a las expresiones que no tienen más continuidad y al ser evaluadas terminan la ejecución en la rama, además implementa una operación Intérprete asociada a cada símbolo terminal. |
| Consecuencia | Es fácil de poder cambiar la gramática, porque el patrón usa clases que representan reglas gramaticales, podemos usar herencia para cambiar o extender la gramática. Además, adicionar nuevas formas de interpretar expresiones. El patrón Intérprete hace fácil el evaluar una expresión de una forma nueva. |
| Implementación | Para la implementación del patrón se creó un Cliente que sería el que nos va a leer y guardar los archivos XML en el sistema, con esto el cliente tiene una instancia del contexto que ocupa para el patrón y llama al interprete para que se evaluado. Además, se crea una expresión abstracta para que todas las expresiones terminales puedan implementarla. Cuando el cliente dispara al interprete, llega a la expresión terminal se evalúa y devuelve o inicializa el objecto desea para la interpretación. |

Ejercicio 2

|  |  |
| --- | --- |
| Nombre | Observador |
| Propósito | Define una relación entre los objetos, haciendo que cuando uno cambia su estado, todos los demás son notificados en consecuencia. |
| Problema | Observar cada uno de los buses para saber cuándo es la siguiente revisión, que se hace cada 5000 km, y en caso de los 25000 km y 50000 km son revisiones mayores que son urgentes. |
| Solución | Con la implementación de este patrón, logramos tener un observador por cada uno de los buses, y cuando el kilometraje cambie en cualquiera de los buses, se va a notificar a si se acerca revisiones que se acercan. |
| Participantes y colaboradores | Sujeto (Sujeto): Define la interfaz que es implementada por los sujetos concretos para subscribir los observadores.  Sujeto Concreto (flotilla): almacena el estado de interés para sus observadores y envía notificaciones a sus observadores cuando su estado cambia.  Observador (Observador): define la interfaz para los observadores concretos, a los que se le debe notifica cuando un sujeto cambia de estado.  Observador Concreto: (Observado): implementa al observador, almacena el estado del sujeto que le resulta de interés e implementa la interfaz de Observador para mantener su estado consistente con el del sujeto. |
| Consecuencia | Con este patrón tenemos un observador por cada uno de los buses que se encuentran notificando cuando se acerque las revisiones, haciendo una abstracción en el acoplamiento entre el sujeto y el observador, logrando una mayor independencia |
| Implementación | Con la implementación de este patrón lo que se realizó fue una interfaz que contuviera las declaraciones de los métodos que subscriben Observadores, con ello se creó a el sujeto concreto que es nuestra clase flotilla y que esta clase implementara esta interfaz. Con ello se creó la interfaz para los observadores que es con la que se notifica cuando el sujeto cambia. Y por último se crea los observadores concretos que tienen una referencia a los datos del sujeto que son de su interés. |

Ejercicio 3

|  |  |
| --- | --- |
| Nombre | Estrategia |
| Propósito | El patrón define una familia de algoritmos, permite al cliente seleccionar un algoritmo de la familia y pueden ser intercambiables y brinda una manera sencilla de acceder a ellos. |
| Problema | Para la asignación de rutas con buses no se encuentra definida por algoritmo en específico si no que varía dependiendo de la estrategia que quiera el dueño de la empresa. |
| Solución | Con la implementación del patrón, se ofrecen en el sistema tres diferentes tipos de estrategias (escoger un bus aleatoriamente, un bus con poco kilometraje, un bus con mucho kilometraje). Con ello hacemos que el cliente pueda utilizar cualquier algoritmo he intercambiar la estrategia de la ruta en cualquier momento. |
| Participantes y colaboradores | Contexto (ContextoEstrategia): se mantiene una referencia de la interfaz de la estrategia.  Estrategia (Ruta Estrategia): Declara la interfaz común para los algoritmos soportados.  Estrategia concreta (Aleatoria, BusMayorKilometraje, BusMenorKilometraje): Define una implementación a un algoritmo mediante la interfaz de Estrategia. |
| Consecuencia | Define la implementación a un algoritmo mediante la interfaz de Estrategia. Hay gran variedad de estrategias para un solo comportamiento, por lo hay que conocer cómo funciona cada una. Es una alternativa a las subclases, encapsulando los algoritmos en subclases separadas hace que dichos algoritmos sean totalmente independientes del contexto, facilitando la comprensión, entendimiento y extensión. |
| Implementación | Para la implementación del patrón lo que lo que realizó fue un contexto que tuviera una referencia a la interfaz de estrategia, para dependiendo de la decisión del cliente, crear una estrategia en específico. Como se mencionó, se crea una interfaz que es la que tiene que la definición de método que será implementado por las estrategias concretas, y se crean las estrategias concretas que son las que desarrollan un algoritmo para lograr la estrategia. |