INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

Tercer Parcial

Principios de Diseño Orientado a Objetos

Análisis y diseño orientado a objetos

GRUPO: 2CV14

Alumnos:

Estrada Botello Oscar Eduardo	**2019630411**
Pastrana Torres Victor Norberto	**2019630349**
Pereda Reyes Alfredo	**2019630500**
Romero Escogido Donovan	**2019630460**

PROFESORA:

Reyna Elia Melara Abarca

FECHA DE ENTREGA: lunes 21 de junio de 2021

1. **¿Qué significa the Gang of Four - GoF?**

El objetivo principal de los patrones es facilitar la reutilización de diseños y arquitecturas software que han tenido éxito capturando la experiencia y haciéndola accesible a los no expertos.

Dentro de los patrones clásicos tenemos los GoF (Gang of Four),estudiados por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides en su mítico libro Design Patterns se contemplan 3 tipos de patrones:

- **Patrones de creación:** tratan de la inicialización y configuración de clases y objetos
- **Patrones estructurales**: Tratan de desacoplar interfaz e implementación de clases y objetos
- **Patrones de comportamiento** tratan de las interacciones dinámicas entre sociedades de clases y objetos.

2. **En la Introducción de esta actividad se han enfatizado en negritas dos enunciados, proporcione una interpretación sobre estas ideas.**

Son plantillas que identifican problemas en el sistema y proporcionan soluciones apropiadas a problemas generales a los que se han enfrentado los desarrolladores durante un largo periodo de tiempo, a través de prueba y error.

Los patrones de diseño te ayudan a estar seguro de la validez de tu código, ya que son soluciones que funcionan y han sido probados por muchísimos desarrolladores siendo menos propensos a errores.

Las ventajas del uso de patrones:

- Conforman un amplio catálogo de problemas y soluciones
- Estandarizan la resolución de determinados problemas
- Condensan y simplifican el aprendizaje de las buenas prácticas
- Proporcionan un vocabulario común entre desarrolladores

3. **El espacio de los patrones está clasificado de acuerdo a su propósito, los cuáles pueden ser: Creación (Creational), Estructura (Structural) y Comportamiento (Behavioral). Indique a qué se refiere cada uno de estos propósitos.**

Patrones de creación

- **Abstract Factory**. Proporciona una interfaz para crear familias de objetos o que dependen entre sí, sin especificar sus clases concretas.

- **Builder**. Separa la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda crear diferentes representaciones.

- **Factory Method**. Define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clase instanciar. Permite que una clase delegue en sus subclases la creación de objetos.

- **Prototype**. Especifica los tipos de objetos a crear por medio de una instancia prototípica, y crear nuevos objetos copiando este prototipo.

- **Singleton**. Garantiza que una clase sólo tenga una instancia, y proporciona un punto de acceso global a ella.

Patrones estructurales

- **Adapter**. Convierte la interfaz de una clase en otra distinta que es la que esperan los clientes. Permiten que cooperen clases que de otra manera no podrían por tener interfaces incompatibles.

- **Bridge**. Desvincula una abstracción de su implementación, de manera que ambas puedan variar de forma independiente.

- **Composite**. Combina objetos en estructuras de árbol para representar jerarquías de parte-todo. Permite que los clientes traten de manera uniforme a los objetos individuales y a los compuestos.

- **Decorator**. Añade dinámicamente nuevas responsabilidades a un objeto, proporcionando una alternativa flexible a la herencia para extender la funcionalidad.

- **Facade**. Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema se más fácil de usar.

- **Flyweight**. Usa el compartimiento para permitir un gran número de objetos de grano fino de forma eficiente.

- **Proxy**. Proporciona un sustituto o representante de otro objeto para controlar el acceso a éste.

Patrones de comportamiento

- **Chain of Responsibility**. Evita acoplar el emisor de una petición a su receptor, al dar a más de un objeto la posibilidad de responder a la petición. Crea una cadena con los objetos receptores y pasa la petición a través de la cadena hasta que esta sea tratada por algún objeto.

- **Command**. Encapsula una petición en un objeto, permitiendo así parametrizar a los clientes con distintas peticiones, encolar o llevar un registro de las peticiones y poder deshacer las operaciones.

- **Interpreter**. Dado un lenguaje, define una representación de su gramática junto con un intérprete que usa dicha representación para interpretar las sentencias del lenguaje.

- **Iterator**. Proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna.

- **Mediator**. Define un objeto que encapsula cómo interactúan un conjunto de objetos. Promueve un bajo acoplamiento al evitar que los objetos se refieran unos a otros explícitamente, y permite variar la interacción entre ellos de forma independiente.

- **Memento**. Representa y externaliza el estado interno de un objeto sin violar la encapsulación, de forma que éste puede volver a dicho estado más tarde.

- **Observer**. Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambia de estado se notifica y actualizan automáticamente todos los objetos.

- **State**. Permite que un objeto modifique su comportamiento cada vez que cambia su estado interno. Parecerá que cambia la clase del objeto.

- **Strategy**. Define una familia de algoritmos, encapsula uno de ellos y los hace intercambiables. Permite que un algoritmo varíe independientemente de los clientes que lo usan.

- **Template Method**. Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos. Permite que las subclases redefinan ciertos pasos del algoritmo sin cambiar su estructura.

- **Visitor**. Representa una operación sobre los elementos de una estructura de objetos. Permite definir una nueva operación sin cambiar las clases de los elementos sobre los que opera.

4. Los los patrones de diseño a explorar serán MVC, Singleton e Iterator. ¿en qué clasificación recae cada uno de estos? ¿pueden los patrones de diseño usarse con otros patrones de diseño? Explique.

- **MVC**

  MVC es un patrón de diseño que se estructura mediante tres componentes: modelo, vista y controlador. Este patrón tiene como principio que cada uno de los componentes esté separado en diferentes objetos, esto quiere decir que los componentes no se pueden combinar dentro de una misma clase.

  Antes que nada, MVC separa la lógica de negocios y la capa de representación entre sí. Tradicionalmente se utilizaba para interfaces gráficas de usuario de escritorio. Hoy en día, la arquitectura MVC se ha hecho popular para el diseño de aplicaciones Web y aplicaciones mobile.

  Por lo tanto es un patrón estructural.

- **Singleton**

  Singleton es un patrón de diseño que permite restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto.

  Su intención consiste en garantizar que una clase solo tenga una instancia y proporcionar un punto de acceso global a ella.

  El patrón singleton se implementa creando en nuestra clase un método que crea una instancia del objeto solo si todavía no existe alguna. Para asegurar que la clase no puede ser instanciada nuevamente se regula el alcance del constructor (con modificadores de acceso como protegido o privado).

  La instrumentación del patrón puede ser delicada en programas con múltiples hilos de ejecución. Si dos hilos de ejecución intentan crear la instancia al mismo tiempo y esta no existe todavía, solo uno de ellos debe lograr crear el objeto. La solución clásica para este problema es utilizar exclusión mutua en el método de creación de la clase que implementa el patrón.

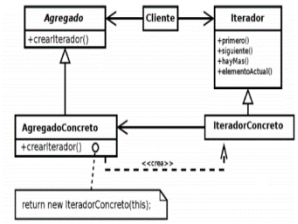
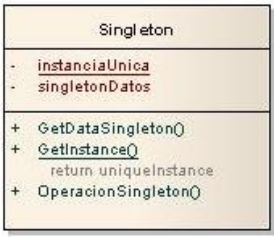
  Por lo tanto es un patrón de creación.

- **Iterator**
  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.).

  Por lo tanto es un patrón de comportamiento.

5. **Incorporar en las tablas (clic aquí) la información sobre los patrones de diseño MVC, Singleton e Iterator. Puede revisar para MVC aquí y en [8], además puede complementar con su propia investigación.**

	Iterator	Singleton	MVC
Intención	Es un mecanismo de acceso a los elementos que constituyen una estructura de datos para la utilización de estos sin exponer su estructura interna.	Consiste en garantizar que una clase solo tenga una instancia y proporcionar un punto de acceso global a ella.	Que cada uno de los componentes esté separado en diferentes objetos, esto significa que los componentes no se pueden combinar dentro de una misma clase.
Motivación	Deseo de acceder a los elementos de un contenedor de objetos (por ejemplo, una lista) sin exponer su representación interna.	En algunas ocasiones es muy importante poder garantizar que solo existe una instancia de una clase.	Clasificar la información, la lógica del sistema y la interfaz que se le presenta al usuario.
Aplicabilidad	Permite el acceso al contenido de una estructura sin exponer su representación interna.	En las que dicha clase controla el acceso a un recurso físico único (como puede ser el ratón o un archivo abierto en modo exclusivo) o cuando cierto tipo de datos debe estar disponible para todos los demás objetos de la aplicación.	El objetivo primordial del patrón es dar soporte a los modelos funcionales y mapas mentales de la información relevante para los usuarios, permitiendo un modelo que facilite la consulta y manejo de estos.

Estructura			
Colaboraciones	Un iterador concreto es el encargado de guardar la posición actual dentro de la estructura de datos, interactuando con esta para calcular el siguiente elemento del recorrido.	Los clientes acceden a la instancia única solamente a través de la operación getInstance.	Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.
Implementación	Para la creación del patrón iterador debe implementarse el control de la iteración (pudiendo ser un iterador externo que ofrece los métodos para que el cliente recorra la estructura paso a paso, o un iterador interno que ofrece un método de actuación sobre la estructura que, de manera transparente al cliente, la recorre aplicándose a todos sus elementos) y definirse el recorrido.	Creando en nuestra clase un método que crea una instancia del objeto solo si todavía no existe alguna. Para asegurar que la clase no puede ser instanciada nuevamente se regula el alcance del constructor (con modificadores de acceso como protegido o privado).	1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc.) 2. El controlador recibe la notificación de la acción solicitada por el usuario 3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario). 4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario.

Código ejemplo

```
public class Vector2 {
    public int[] _datos;

    public Vector2(int valores){
        _datos = new int[valores];
        for (int i = 0; i < _datos.length; i++){
            _datos[i] = 0;
        }
    }

    public int getValor(int pos){
        return _datos[pos];
    }

    public void setValor(int pos, int valor){
        _datos[pos] = valor;
    }

    public int dimension(){
        return _datos.length;
    }

    public IteradorVector iterador(){
        return new IteradorVector(this);
    }
}
```

```
public class Singleton{
   private static var instance:Singleton;
   private static var allowInstance:Boolean;
   public function Singleton(){
      if(!allowInstance){
         throw new Error("Debes usar getInstance()");
      }else{
         trace("Se inicializó una instancia de Singleton");
      }
   }

   public static function getInstance():Singleton{
      if(instance==null){
         allowInstance=true;
         instance= new Singleton();
         allowInstance=false;
      }else{
         trace("Se regresa la instancia existente");
      }
      return instance;
   }
}
```

```
public class MainMVC{
    public static void main(String[] args) {
      Intern model = retriveInternFromDatabase();
       InternView view = new InternView();
       InternController controller = new InternController(model, view);
       controller.updateView();
       controller.setInternName("John");
       controller.updateView();
    }

    public static Intern retriveInternFromDatabase(){
       Intern intern = new Intern();
       intern.setName("Jane");
       intern.setLastname("Doe");
       return intern;
    }
}
```

6. **En el desarrollo de su sistema de software, ¿ha detectado alguna oportunidad de utilizar patrones de diseño? Dependiendo de las características de un proyecto, MVC suele ser comúnmente utilizado en el desarrollo de aplicaciones Web. ¿Ha utilizado algún otro patrón de diseño? De ser así indique cómo y si le es posible agregue una porción del código en donde lo haya empleado.**

Decidimos implementar el patrón de diseño MVC, ya que como lo menciona en la pregunta este patrón de diseño es comúnmente utilizado en el desarrollo de páginas web. Esto se debe a que este patrón de diseño ha sido ampliamente adaptado como una arquitectura para diseñar e implementar aplicaciones web en los principales lenguajes de programación. Se han desarrollado frameworks (comerciales y no comerciales) que implementan este patrón.

7. **MVC suele ser clasificado como un estilo arquitectónico o también como una arquitectura. Explique, es un patrón, una arquitectura, un estilo arquitectónico, todas las anteriores, ninguna de las anteriores, =D.**

MVC era inicialmente un patrón arquitectural, un modelo o guía que expresa cómo organizar y estructurar los componentes de un sistema software, sus responsabilidades y las relaciones existentes entre cada uno de ellos.

Su nombre, MVC, parte de las iniciales de Modelo-Vista-Controlador (Model-View-Controller, en inglés), que son las capas o grupos de componentes en los que organizaremos nuestras aplicaciones bajo este paradigma.

Es a menudo considerado también un patrón de diseño de la capa de presentación, pues define la forma en que se organizan los componentes de presentación en sistemas distribuidos.

Aunque el auge de este término durante los últimos tiempos pueda indicar lo contrario, MVC no es un concepto nuevo, ya que el patrón fue descrito en el año 1979 por Trygve Reenskaug, hoy en día profesor emérito de informática de la Universidad de Oslo, mientras trabajaba con el equipo de Smalltalk en los laboratorios Xerox PARC.

Por tanto, teniendo en cuenta su antigüedad, es obvio que ni siquiera fue ideado expresamente para sistemas web aunque ahora se use mucho en ellas. Existen implementaciones para todo tipo de sistemas (escritorio, clientes y servidores web, servicios web, Single Page Applications o SPA, etc.) y lenguajes (Smalltalk, Java, Ruby, C++, Python, PHP, JavaScript, NodeJS, etc.).

La arquitectura MVC propone, independientemente de las tecnologías o entornos en los que se base el sistema a desarrollar, la separación de los componentes de una aplicación en tres grupos (o capas) principales: el modelo, la vista, y el controlador, y describe cómo se relacionarán entre ellos para mantener una estructura organizada, limpia y con un acoplamiento mínimo entre las distintas capas.

Bibliografía

[1] https://unpocodejava.com/2013/01/02/un-poco-de-patrones-de-diseno-gof-gang-of-four/

[2] https://profile.es/blog/patrones-de-diseno-de-software/

[3] https://medium.com/all-you-need-is-clean-code/patrones-de-dise%C3%B1o-b7a99b8525e

[4] https://refactoring.guru/es/design-patterns/iterator

[5] https://es.wikipedia.org/wiki/Singleton

[6] https://blog.nearsoftjobs.com/patr%C3%B3n-de-dise%C3%B1o-mvc-2366948b5fc7

[7] http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/202