Programación II

Nombre: Isias Mateo

Matricula: 2018-6500

Profesor: Viela Reyes

Patrones creacionales

- **Abstract Factory**: Nos provee una interfaz que delega la creación de un conjunto de objetos relacionados sin necesidad de especificar en ningún momento cuáles son las implementaciones concretas.
- **Factory Method:** Expone un método de creación, delegando en las subclases la implementación de este método.
- **Builder**: Separa la creación de un objeto complejo de su estructura, de tal forma que el mismo proceso de construcción nos puede servir para crear representaciones diferentes.
- **Singleton:** limita a uno el número de instancias posibles de una clase en nuestro programa, y proporciona un acceso global al mismo.
- **Prototype:** Permite la creación de objetos basados en «plantillas». Un nuevo objeto se crea a partir de la clonación de otro objeto.

Patrones estructurales

- **Adapter**: Permite a dos clases con diferentes interfaces trabajar entre ellas, a través de un objeto intermedio con el que se comunican e interactúan.
- **Bridge**: Desacopla una abstracción de su implementación, para que las dos puedan evolucionar de forma independiente.

- **Composite**: Facilita la creación de estructuras de objetos en árbol, donde todos los elementos emplean una misma interfaz. Cada uno de ellos puede a su vez contener un listado de esos objetos, o ser el último de esa rama.
- **Decorator**: Permite añadir funcionalidad extra a un objeto (de forma dinámica o estática) sin modificar el comportamiento del resto de objetos del mismo tipo.
- **Facade**: Una facade (o fachada) es un objeto que crea una interfaz simplificada para tratar con otra parte del código más compleja, de tal forma que simplifica y aísla su uso. Un ejemplo podría ser crear una fachada para tratar con una clase de una librería externa.
- **Flyweight**: Una gran cantidad de objetos comparte un mismo objeto con propiedades comunes con el fin de ahorrar memoria.
- **Proxy**: Es una clase que funciona como interfaz hacia cualquier otra cosa: una conexión a Internet, un archivo en disco o cualquier otro recurso que sea costoso o imposible de duplicar.

Patrones de comportamiento

- **Command**: Son objetos que encapsulan una acción y los parámetros que necesitan para ejecutarse.
- **Chain of responsibility**: se evita acoplar al emisor y receptor de una petición dando la posibilidad a varios receptores de consumirlo. Cada receptor tiene la opción de consumir esa petición o pasárselo al siguiente dentro de la cadena.
- **Interpreter**: Define una representación para una gramática, así como el mecanismo para evaluarla. El árbol de sintaxis del lenguaje se suele modelar mediante el patrón Composite.
- **Iterator**: Se utiliza para poder movernos por los elementos de un conjunto de forma secuencial sin necesidad de exponer su implementación específica.
- **Mediator**: Objeto que encapsula cómo otro conjunto de objetos interactúan y se comunican entre sí.
- **Memento**: Este patrón otorga la capacidad de restaurar un objeto a un estado anterior
- **Observer**: Los objetos son capaces de suscribirse a una serie de eventos que otro objetivo va a emitir, y serán avisados cuando esto ocurra.
- **State**: Permite modificar la forma en que un objeto se comporta en tiempo de ejecución, basándose en su estado interno.

- **Strategy**: Permite la selección del algoritmo que ejecuta cierta acción en tiempo de ejecución.
- **Template Method**: Especifica el esqueleto de un algoritmo, permitiendo a las subclases definir cómo implementan el comportamiento real.
- **Visitor**: Permite separar el algoritmo de la estructura de datos que se utilizará para ejecutarlo. De esta forma se pueden añadir nuevas operaciones a estas estructuras sin necesidad de modificarlas.