Introducción Patrones de Diseño

Diseñar software orientado a objetos es difícil, y aún lo es más diseñar software orientado a objetos reutilizable. Hay que encontrar los objetos pertinentes, factorizarlos en clases con la granuralidad adecuada, definir interfaces de clases de jerarquías de herencia y establecer las principales relaciones entre esas clases y objetos. Nuestro diseño debe ser específico del problema que estamos manejando, pero también lo suficientemente general para adecuarse a futuros requisitos y problemas. Los diseñadores experimentados de software orientado a objetos nos dirán que es difícil, sino imposible, lograr un diseño flexible y reutilizable la primera vez.
Sin embargo, estos diseñadores experimentados consiguen hacer buenos diseños, mientras que los diseñadores novatos se ven abrumados por las opciones disponibles y tienden a recurrir a las técnicas no orientadas a objetos que ya usaron antes, y esto es un futuro FRACASO. Es evidente que los diseñadores experimentados saben algo que los novatos no, ¿qué es?.
Algo que los expertos saben que NO hay que hacer es resolver cada problema partiendo de cero. Por el contrario, reutilizan soluciones que ya le han sido útiles en el pasado. Cuando encuentran una solución buena la usan una y otra vez. Esa experiencia es parte de lo que los convierte en expertos.
Por lo tanto, nos encontramos con patrones recurrentes de clases y comunicaciones entre objetos en muchos sistemas orientados a objetos. Estos patrones resuelven problemas concretos de diseño y hacen que los diseños orientados a objetos sean aún más flexibles, elegantes y reutilizables. Los patrones ayudan a los diseñadores a reutilizar buenos diseños, al basar los nuevos diseños en la experiencia previa. Un diseñador familiarizado con dichos patrones puede aplicarlos inmediatamente en los problemas de diseño sin tener que re descubrirlos.
Todos sabemos el valor de la experiencia en el diseño. ¿Cuántas veces hemos tenido un *deja-vu* de diseño -esa sensación de que ya hemos resuelto ese problema antes, pero no sabemos exactamente dónde ni cómo?. Si pudieramos recordar los detalles del problema anterior y de como lo resolvimos, podríamos valernos de esa experiencia sin tener que reinventar la solución. Sin embargo, no solemos dedicar a dejar constancia de nuestra experiencia en el diseño de software para que la usen otros.
Parte del propósito de este artículo es acceder a la documentación de la experiencia del diseño de software orientado a objetos en forma de patrones de diseño (veremos alguno de ellos). El objetivo de los patrones de diseño es representar esa experiencia.
Los patrones de diseño hacen que sea más fácil reutilizar buenos diseños y arquitecturas; nos ayudan a elegir las alternativas de diseño que hacen que un sistema sea reutilizable.

¿Qué es un patrón de diseño?

Según Christopher Alexander, *"cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, así como la solución a ese problema, de tal modo que se pueda aplicar esta solución un millón de veces, sin hacer lo mismo dos veces"*. Aunque Alexander se refería a patrones en Ciudades y edificios, lo que dice también es válido para patrones de diseño orientado a objetos. Nuestras soluciones se expresan en términos de objetos e interfaces, en lugar de paredes y puertas, pero en la esencia de ambos tipos de patrones se encuentra una solución a un problema dentro de un contexto.
En general, un patrón tiene cuatro elementos esenciales:

1. El **nombre** del patrón permite describir, en una o dos palabras, un problema de diseño junto con sus soluciones y consecuencias. Al dar el nombre a un patrón inmediatamente estamos incrementando nuestro vocabulario de diseño, lo que nos permite diseñar con mayor abstracción. Tener un vocabulario de patrones nos permite hablar de ellos con otros colegas,

mencionarlos en nuestra documentación y tenerlos nosotros mismos en cuenta. De esta manera, resulta más fácil pensar en nuestros diseños y transmitirlos a otros, junto con sus ventajas e inconvenientes.

2. El **problema** describe cuando aplicar el patrón. Explica el problema y su contexto. Puede describir problemas concretos de diseño (por ejemplo, cómo representar algoritmos como objetos), así como las estructuras de clases u objetos que son sintomáticas de un diseño inflexible. A veces el problema incluye una serie de condiciones que deben darse para que tenga sentido aplicar el patrón.
3. La **solución** describe los elementos que constituyen el diseño, sus relaciones, responsabilidades y colaboraciones. La solución no describe un diseño o una solución en concreto, sino que el patrón es más bien como una plantilla que puede aplicarse en muchas situaciones diferentes. El patrón proporciona una descripción abstracta de un problema de diseño y cómo lo resuelve una disposición general de elementos, en nuestro caso clases y objetos.
4. Las **consecuencias** son los resultados así como las ventajas e inconvenientes de aplicar el patrón. Principalmente la reutilización suele ser uno de los factores de los diseños orientados a objetos, las consecuencias de un patrón incluyen su impacto sobre la flexibilidad, extensibilidad y portabilidad de un sistema; incluir estas consecuencias de modo explícitos nos ayudará a comprenderlas y evaluarlas.

La elección del lenguaje de programación es importante. Nuestros patrones presuponene características de los lenguajes como Smalltalk y C++, y esa elección determina lo que puede implementarse o no fácilmente. Pero, nosotros vamos a utilizar Java, ¿entonces no podemos aplicar los patrones de diseño?, si, tranquilos, Java *"es un lenguaje orientado a objetos"*, y a pesar de sus particularidades o limitaciones dentro de este Paradigma, nos es posible diseñar una solución de software orientada a objetos, además aplicando los patrones de diseño en caso de desearlo, muy recomendado.

Los patrones pueden tener un propósito de **creación**, **estructural** o de **comportamiento**. Los patrones de creación tienen que ver con el proceso de creación de objetos. Los patrones estructurales tratan con la composición de clases u objetos. Los de comportamiento caracterizan el modo en que las clases y objetos interactúan y se reparten la responsabilidad.
A continuación el listado de los principales patrones de diseño catalogados en la clasificación mencionada anteriormente:

Creación: Factory Method, Abstract Factory, Builder, Prototype y Singleton.
Estructural: Adapter, Bridge, Composite, Decorator, Facade, Flyweighty Proxy.
Comportamiento: Interpreter, Template Method, Chain of Responsability, Command, Iterator, Mediator, Memento, Observer, State, Strategy y Visitor.

Nosotros estudiaremos algunos de ellos, no todos.

Esta introducción, fue extraída y adaptada del libro: ***Design Patterns***, autores: Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides.

ES MUY RECOMENDABLE SU LECTURA.