Taller 5

Natalia Ortega - 201814519

Link: https://github.com/landy8530/DesignPatterns.git

Link del patrón específico:
https://github.com/landy8530/DesignPatterns/tree/master/design-patterns-basic/src/main/java/org/landy/builder

Este es un ejemplo específicamente para el patrón de diseño Builder del tipo creacional, en este ejemplo se construye para modelar la integración de datos de una empresa con una estructura flexible y escalable.

La estructura de diseño incluye: (i) las clases "Member", "Order", "SelectedPlans" y "UserProfile" son clases de dominio de los diferentes componentes del objeto de integración de datos; (ii) la clase "IntegrationData" representa el objeto de integración de datos y almacena los componentes construidos; (iii) la interfaz "IntegrationBuilder" que define los métodos necesarios para construir cada componente del objeto de integración de datos, como los mencionados en el primer punto; (iv) la clase "IntegrationDirector" que coordina la construcción del objeto de integración de datos utilizando el Builder. Invoca los métodos del Builder para construir cada componente y devuelve el objeto de integración de datos final; (v) la clase "XHFIntegrationBuilder" que implementa la interfaz "IntegrationBuilder" y proporciona la implementación concreta para construir cada componente del objeto de integración de datos. Inicializa una instancia de "IntegrationData" en su constructor y almacena los componentes construidos en él; y (vi) la clase "Test", esta es la clase principal que contiene el método main().

Algunos de los retos de diseño que enfrenta el proyecto está diseñar una estructura que permita una construcción incremental de objetos de integración de datos complejos, brindando flexibilidad y escalabilidad. Por esto se implementa el patrón Builder, ya que este proporciona una separación clara entre el proceso de construcción y los objetos resultantes, ya que esto permite agregar y configurar componentes de manera modular, lo que facilita la adaptación a diferentes escenarios y requisitos cambiantes.

El patrón Builder se encuentra en las siguientes clases:

- IntegrationBuilder (Interfaz)
- IntegrationDirector (Clase): con su método "buildIntegrationData()" coordina la construcción de los diferentes componentes utilizando el constructor proporcionado.
- XHFIntegrationBuilder (Clase): implementa la interfaz "IntegrationBuilder" y proporciona la implementación concreta de cada método de construcción de componentes.
- IntegrationData (Clase): almacena los componentes construidos, de tal manera que representa el objeto de integración de datos.

El patrón Builder es un “Creational Pattern” que se utiliza para construir objetos complejos paso a paso. El propósito principal del patrón Builder es separar la construcción de un objeto complejo de su representación y permitir la creación de diferentes representaciones utilizando el mismo proceso de construcción. Esto brinda flexibilidad y reusabilidad al construir objetos complejos, ya que se pueden crear diferentes variaciones de un objeto sin cambiar su estructura interna.

Usualmente, el patrón Builder se utiliza cuando:

- El algoritmo para crear un objeto complejo debe ser independiente de las partes que componen el objeto y cómo se ensamblan.
- El proceso de construcción debe permitir diferentes receptaciones para el objeto que está construido.

El patrón Builder se define mediante la interfaz “IntegrationBuilder”, que proporciona los métodos necesarios para construir cada componente del objeto de integración de datos. Los métodos en esta interfaz representan los pasos de construcción del objeto. Luego, la clase “IntegrationDirector” actúa como el director del proceso de construcción. Su método buildIntegrationData() toma un objeto de tipo “IntegrationBuilder” y utiliza los métodos de construcción proporcionados por el builder para construir el objeto de integración de datos. Por otro lado, la clase “XHFIntegrationBuilder” implementa la interfaz “IntegrationBuilder” y proporciona la implementación concreta para construir cada componente del objeto de integración de datos. Cada método de construcción crea el objeto respectivo, establece sus propiedades y lo almacena en el objeto de integración de datos (“IntegrationData”). Por último, la clase “IntegrationData” representa el objeto de integración de datos y almacena los componentes construidos. Tiene métodos de acceso (getters) y establecimiento (setters) para cada componente y proporciona un método toString() para obtener una representación en forma de cadena del objeto de integración de datos.

El usar este patrón tiene la ventaja que proporciona una construcción paso a paso, flexibilidad, reusabilidad, ocultamiento de la lógica de construcción, facilidad de evolución y mantenimiento, y mejora la legibilidad y comprensión del código.

Sin embargo, se pueden presentar desventajas dependiendo del contexto o las necesidades del proyecto en que se podría presentar una complejidad adicional, aumento de la cantidad de clases y sobrecarga en el código.

Otra forma en la que se podría resolver el problema sería Utilizar métodos de configuración individuales, en lugar de tener un proceso de construcción completo en el builder, se podrían haber proporcionado métodos individuales de configuración en la clase IntegrationData para cada componente.