Taller 5: Patrones

Factory Method

Creado por:

Nicolás Prada Quintero

202113867

[n.pradaq@uniandes.edu.co](mailto:n.pradaq@uniandes.edu.co)

Sección 1 DPOO

Información del proyecto:

[Link del Proyecto](https://github.com/vishwanathdc/BankApplication.git)

Este proyecto simula el funcionamiento de un banco. La aplicación permite que un cuanta bancaria pueda depositar y extraer dinero de la cuenta, mirar la cantidad de dinero en la cuenta y ver la transacción previa. Para esto la aplicación crea un objeto BankAccount con las especificaciones de la cuenta como: Nombre del usuario, id de la cuenta, saldo de la cuenta, transacción previa, etc. Este objeto, a su vez, contiene los métodos para ejecutar cada una de estas funciones por medio de un menú que muestra en consola.

Empleo del patrón:

La parte del proyecto que nos importa es la clase BankAccountFactory. Esta clase se encarga de crear los objectos BankAccount que usa la aplicación. Para esto se emplea el patrón que se está estudiando: Factory Method. El patrón es un método, llamado createAccount que se encarga de crear el objeto que se necesita.

Información general del Patrón:

Este patrón se define como una interfaz diseñada para crear un objeto, de forma que les deje a las subclases escoger qué clase instanciar. Este patrón se utiliza comúnmente cuando se quiere crear un nuevo objeto, pero, dependiendo de las circunstancias de la aplicación, este objeto puede ser de diferentes clases. Un ejemplo de esto es una aplicación que crea usuarios, pero hay diferentes tipos de usuarios como administradores, jugadores, moderadores, etc. Cada uno con un acceso a diferentes interfaces, funcionamientos y libertades.

Información implementación del Patrón:

En este caso el patrón no se está usando por que haya diferentes clases de cuentas bancarias. La implementación del patrón impide que la aplicación haga un llamado directo del constructor, en vez obliga a la aplicación a hacer el llamado del Factory Method. Esto permite que el método compruebe que ninguno de los datos de la cuenta bancaria esté vacío a la hora de crear el objeto. Por otro lado, dependiendo del notificationType con el que se llame al método se le asigna un observador diferente a la cuenta.

Para este proyecto tiene sentido haber implementado el patrón porque permite encapsular funcionamientos de la aplicación y agrega facilidad para escalar el proyecto en un futuro. La implementación del patrón permite encapsular la creación de las cuentas. Esto facilita la comprensión del proyecto, pues se le asigna un rol y tarea específica a la fábrica encarga de crear estos objetos. Por otro lado, implementar este patrón permite que en un futuro la aplicación crezca para permitir distintos tipos de cuentas bancarias. Por ejemplo, se podría hacer para que existan cuentas bancarias de personas naturales y empresariales, cuentas para menor de edad o cuentas enfocadas a la inversión. Estos tipos diferentes de cuentas siguen siendo en esencia cuentas bancarias, pero con funcionamientos, requerimientos y restricciones muy distintas. El patrón que se implementó serviría para que el Factory Method se encargue de crear cada una de estas cuentas diferentes a pesar de que sean clases de objetos diferentes.

Aún así, la implementación del patrón crea unas desventajas. Como el patrón únicamente está implementado para comprobar los datos de la cuenta se hace más complicado el seguimiento de procesos de la aplicación. En vez simplemente llamar el constructor de la clase BankAccount hay una implementación de una interfaz designada a crear estos objetos (BankAccountFactory) y un interfaz de cuenta bancaria (BankAccountInterface) que abstrae el funcionamiento de la cuenta. Hubiera sido mucho más simple haber comprobado los valores de los atributos de la cuenta por medio de condicionales dentro del constructor o creando un contrato con el constructor. También, para asignar diferentes tipos de observadores a la cuenta se pudo haber hecho cambios en el constructor de la clase, agregando parámetros y condicionales dentro de la ejecución.

Sin embargo, estas desventajas son muy mínimas comparadas con los beneficios que trae la implementación del patrón, como las anteriores mencionadas. Asimismo, las alternativas ponen muchas restricciones sobre el proyecto, más que todo sobre el constructor de la clase BankAccount y la abstracción de esta. Expandir el proyecto en un futuro implicaría una serie de cambios significativos sobre el proyecto y la clase BankAccount.