Taller 5: Bridge Design Pattern

Juan Camilo Beltran

**Introducción: ¿Que es el patrón Bridge?**

El patrón de diseño Bridge es un patrón estructural del grupo GoF que nos permite desacoplar o separar dos o más abstracciones de una clase de su implementación en jerarquías diferentes. Esto con el objetivo de obtener mas control y flexibilidad a la hora de manejar ambas categorías, la clase y su respectiva implementación, además de aumentar considerablemente el nivel de independencia de cada uno. Generalmente se usa cuando la herencia entre clases no es suficiente o se vuelve extremadamente complicado manejarla de esta manera, otros casos en los que el uso del patrón Bridge es preferible incluye proyectos en los que no quieres que una clase este permanentemente atada a su implementación, hay una extensa cantidad de subclases y sub-subclases o una jerarquía muy compleja o sencillamente quieres compartir una implementación entre múltiples clases.

**Proyecto elegido: Shopping Aplication**

Este proyecto consiste en una aplicación móvil para Android en la que un solo usuario puede realizar compras de varios productos, aunque parte de la aplicación, principalmente de interfaz y funcionamiento en Android, está construida en herramientas como Gradle, toda la lógica esta hecha en Java. Esta cuenta con varios paquetes de los cuales cada uno tiene una función específica, el paquete Fragment construye parte de la interfaz y conecta con el resto de la lógica, el paquete Messages se encarga de que toda la información necesaria sea compartida con el usuario en forma de mensajes temporales, el paquete Observer revisa las acciones del usuario en la interfaz, Order tiene como función mantener el registro de las ordenes y toda la información al respecto, PaymentStrategy maneja toda la lógica de un pago aunque no este completamente funcional, ProductObserver, al igual que Observer, revisa las acciones pero más específicamente al momento de manejar productos, por último, el paquete Recycler inicializa la mayoría de la aplicación y crea adaptadores para acoplar correctamente todo el proyecto. Entre los mayores retos que veo en este proyecto se encuentra la correcta estructuración, ya que las funcionalidades son relativamente simples, parecen desconectadas y puede haber muchas problemáticas al momento de pensar en el ciclo de vida de cada elemento, y la correcta implementación de todas las funcionalidades de la forma mas conveniente.

**Patron Bridge en Shopping Application.**

Diagrama

Descripción generada automáticamenteDentro del Proyecto Shopping Application, el patrón Bridge se presenta en el paquete messages de esta manera:

Como se puede observar, dentro del proyecto existe la necesidad de enviar diferentes mensajes al usuario, como los EmailMessage y ToastMessage, cada uno tiene su implementación, EmailMessageSender y ToastMessageSender respectivamente, estas clases contienen implementaciones diferentes de su único método, también componen otras clases muy diferentes dentro del proyecto. El patrón Bridge es utilizado para desacoplar la clase abstracta Message y sus subclases de su implementación iMessageSender y sus herederos, creando efectivamente un puente entre ellas. Este es un uso útil del patrón Bridge ya que tiene varias ventajas, ya que los mensajes son usados por variadas clases, tener una implementación que no afecte directamente la clase es muy útil, además es un momento oportuno para usarlo ya que puede que en un futuro tenga la necesidad de crear mas tipos de mensajes, por dar un ejemplo, un tipo especial de mensaje para mostrar errores o para indicar al usuario que una acción no es posible, de esta forma, dependiendo de lo que necesite puedo solo crear una implementación diferente y no cambiar toda mi estructura. Una desventaja de este patrón es que puede ser un poco más difícil de entender la relación entre sus diferentes participantes a primera vista, ya que las implementaciones no son directas.

En este caso particular, la relación y uso de las clases es lo suficientemente simple como para que simplemente heredar de las clases (ver el próximo uml) sea una buena solución para este problema, sin embargo, como fue mencionado anteriormente, nos permite tener una mejor estructura para un futuro en el que haya mas elementos presentes y las implementaciones diverjan más.

Diagrama

Descripción generada automáticamente

**LINK:** [**https://github.com/burakgomec/Shopping\_Application\_Design\_Patterns.git**](https://github.com/burakgomec/Shopping_Application_Design_Patterns.git)