Taller 5 - Patrones de Diseño

Karen Andrea Fuentes Barreto - 202122467

Patrón escogido: *Adapter*

link del proyecto: https://github.com/bumptech/glide

1. *Información general del proyecto*

El proyecto escogido para realizar este análisis es Glide. Este repositorio cuenta con 139 contribuidores y es desarrollado por Bump Technologies. Este proyecto de código abierto ubicado en github, cuyo propósito es ser "una biblioteca de carga de imágenes rápida y eficiente para Android centrada en el desplazamiento suave." El funcionamiento de este consiste en ofrecer una API flexible, que permite la obtención, decodificación y visualización de imágenes fijas de vídeo, imágenes y GIF animados.

Su enfoque principal es hacer que el desplazamiento de cualquier tipo de lista de imágenes sea lo más suave y rápido posible, también es efectivo para casi cualquier caso en el que se necesite buscar, cambiar el tamaño y mostrar una imagen remota. (*Glide v4 : Fast and Efficient Image Loading for Android*, 2015)

La estructura general del diseño consiste de un *RequestBuilder*, que se usa para solicitudes a la hora de cargar imágenes. *RequestManager*, que administra estas solicitudes y maneja su almacenamiento en caché. *ModelLoader* el cual maneja diferentes fuentes de imágenes, y finalmente un *ResourceDecoder*, que se encarga de decodificar las imágenes.

Los grandes retos a los que se enfrenta este proyecto es cargar las imágenes de manera eficiente, independientemente de la fuente de la que provengan. Por otro lado, debe optimizar y hacer un trade-off entre el almacenamiento de caché y memoria que ocupa y la carga rápida de las imágenes. Adicionalmente, debe realizar transformaciones de imágenes (como cambiar el tamaño) manteniendo la calidad visual, y finalmente, manejar varios formatos de imagen (png, jpeg y gif) haciendo una decodificación precisa de estas.

2. *Información y estructura del fragmento del proyecto donde aparece el patrón*

La estructura principal del fragmento que incorpora el patrón Adapter gira en torno a la clase ImgurImageAdapter, que es una clase dentro de la clase MainActivity y que hereda de RecyclerView.Adapter<ViewHolder>

En esta estructura:

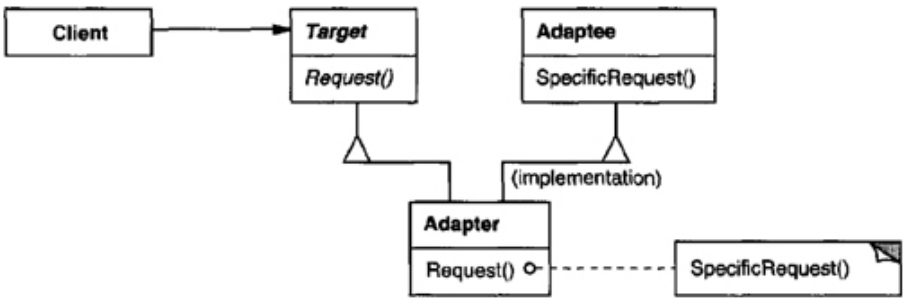
- *MainActivity* actúa como la actividad principal de la aplicación y utiliza ImgurImageAdapter para cerrar la brecha entre RecyclerView y la lista de datos de imagen (List<Image>).
- *ImgurImageAdapter* sirve como adaptador para RecyclerView y maneja varios métodos para vincular los datos de la lista de imágenes (List<Image>) a las views en la clase ViewHolder, que representa elementos individuales en RecyclerView. Expande el diseño de cada elemento, establece la imagen y el título en función del objeto de imagen correspondiente en el conjunto de datos y determina el recuento de elementos.
- Al extender *RecyclerView.Adapter<ViewHolder>*, la clase *ImgurImageAdapter* sigue el patrón del adaptador, lo que le permite adaptar los datos de imagen a RecyclerView y facilitar la visualización de imágenes y títulos en la lista desplegable de tarjetas. Encapsula las complejidades de administrar el conjunto de datos y conectarlo a las views dentro de RecyclerView, cerrando la brecha entre los datos de las imágenes y la interfaz de usuario.

3. *Información general sobre el patrón*

Según el libro Design Patterns: Elements of Reusable Object-Oriented Software, el patrón Adapter es un patrón estructural. Esto quiere decir que se ocupa de la manera en la cual las clases y objetos se componen para crear una estructura mayor. Estos se caracterizan por usar la herencia para componer interfaces o implementaciones.

En particular, el Adapter “convierte la interfaz de una clase a otra interfaz que el cliente necesita” permitiendo así que clases con interfaces incompatibles trabajen juntas. Este patrón es muy útil ya que aborda situaciones en las que una clase diseñada para la reutilización no es reutilizable debido a una discordancia entre su interfaz y la interfaz específica requerida por una aplicación. En el libro se representa el adapter de la siguiente manera:

A class adapter uses multiple inheritance to adapt one interface to another:



Usualmente este patrón se usa, como su nombre lo indica, para adaptar una interfaz que no es compatible con el programa a una que sí lo es. Una analogía comúnmente usada para describir este patrón es el adaptador de corriente, que permite usar las diferentes tomas de electricidad de diferentes países con el conector que se tiene. Un caso aplicado mencionado en el libro es el de un editor gráfico, donde se hace que una clase existente, *TextView,* sea compatible con la interfaz Shape requerida por el editor. El adaptador (TextShape) permite la integración de interfaces incompatibles, para integrar las funcionalidades de edición de texto con el editor de dibujos.

4. *Información del patrón aplicado al proyecto*

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.recyclerview.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/recycler_view"
    android:paddingBottom="20dp"
    android:paddingTop="20dp"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

```
private final class ImgurImageAdapter extends RecyclerView.Adapter<ViewHolder> {

  private List<Image> images = Collections.emptyList();

  public void setData(@NonNull List<Image> images) {
    this.images = images;
    notifyDataSetChanged();
  }

  @Override
  public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    return new ViewHolder(
        LayoutInflater.from(parent.getContext()).inflate(R.layout.image_card, parent, false));
  }

  @Override
  public void onBindViewHolder(RecyclerView.ViewHolder holder, int position) {
    ViewHolder vh = (ViewHolder) holder;
    Image image = images.get(position);
    vh.title.setText(TextUtils.isEmpty(image.title) ? image.description : image.title);

    ImgurGlide.with(vh.imageView).load(image.link).into(vh.imageView);
  }

  @Override
  public int getItemCount() {
    return images.size();
  }


  private final class ViewHolder extends RecyclerView.ViewHolder {

    private final ImageView imageView;
    private final TextView title;

    ViewHolder(View itemView) {
      super(itemView);
      imageView = (ImageView) itemView.findViewById(R.id.image);
      title = (TextView) itemView.findViewById(R.id.title);
    }
```

Al observar estos snippets de código podemos identificar el RecyclerView. Este es un componente de interfaz de usuario proporcionado por la biblioteca de AndroidX. Se utiliza para mostrar de manera eficiente grandes listas o cuadrículas de elementos mediante el reciclaje y la reutilización de views de elementos.

El componente *RecyclerView* de la biblioteca de AndroidX sirve como interfaz central para mostrar imágenes en una aplicación de Android. Sin embargo, para

conectar RecyclerView con los datos reales y definir la apariencia de cada elemento, se requiere un adaptador que implemente la interfaz RecyclerView.Adapter.

En este proyecto en particular, la clase *ImgurImageAdapter* actúa como adaptador para *RecyclerView*. Al extender la clase *RecyclerView.Adapter<ViewHolder>*, *ImgurImageAdapter* adapta la interfaz de la biblioteca de AndroidX (RecyclerView.Adapter) a una implementación más específica adaptada a las necesidades del proyecto.

ImgurImageAdapter hace Override a los métodos necesarios de la interfaz RecyclerView.Adapter, como *onCreateViewHolder*, *onBindViewHolder* y *getItemCount*, (como se ve muestra en el código) y vincula los datos a las views de elementos y define cuántos elementos se deben mostrar. También define una clase ViewHolder personalizada que contiene las views individuales de cada elemento en RecyclerView.

Al implementar la interfaz RecyclerView.Adapter, ImgurImageAdapter establece una conexión entre RecyclerView y la fuente de datos (en este caso, una lista de imágenes). Maneja la creación de views, vinculando los datos a las views y determinando la cantidad de elementos que se mostrarán. Esta adaptación de la interfaz permite que RecyclerView muestre de manera efectiva las imágenes de la fuente de datos utilizando ImgurImageAdapter como puente entre los dos.

5. *¿Por qué tiene sentido haber utilizado el patrón en ese punto del proyecto? ¿Qué ventajas tiene?*

Tiene sentido utilizar el patrón Adapter ya que este cierra la brecha entre el componente *RecyclerView* de la biblioteca de AndroidX y las fuentes de datos específicas de las imágenes, lo cuál es el propósito de este patrón estructural enfocado en integrar interfaces incompatibles.

Las ventajas de implementar este patrón es que este proporciona flexibilidad y reutilización para las diferentes páginas o aplicaciones ya que permite una fácil integración de diferentes fuentes de datos o incluso cambiar entre ellas sin afectar la estructura general de RecyclerView. El patrón Adapter abstrae las complejidades de trabajar con diversas fuentes de datos, proporcionando una interfaz unificada con la que RecyclerView puede interactuar. Otra de sus ventajas es que se mejora el

mantenimiento del código base y facilita la ampliación o modificación del proyecto en el futuro. Además, el patrón de adaptador permite encapsular la lógica de manejo de datos dentro de la clase de adaptador, lo que permite una mejor organización del código y separación de responsabilidades.

6. *¿Qué desventajas tiene haber utilizado el patrón en ese punto del proyecto?*

Una posible desventaja de usar el patrón de adaptador en este proyecto es la complejidad adicional y la sobrecarga que presenta la clase de adaptador. El adaptador actúa como intermediario entre RecyclerView y la fuente de datos, lo que puede aumentar el tamaño y la complejidad general de la base de código y su capacidad de memoria. Además, pueden haber casos de interfaces en los que la fuente de datos y RecyclerView tengan una compatibilidad más directa. Por otro lado, el adaptador se encarga de hacerle override a métodos del RecyclerView, lo cual asigna responsabilidades adicionales al adaptador.

7. *¿De qué otras formas se le ocurre que se podrían haber solucionado, en este caso particular, los problemas que resuelve el patrón?*

Considero que otra forma de solucionar el problema que resuelve el patrón era utilizando el patrón de diseño Decorator. Ya que, a diferencia del patrón Adapter, cuyo objetivo principal es brindar compatibilidad entre interfaces, el patrón Decorator permite agregar nuevas funcionalidades sin comprometer la simplicidad del diseño. En caso de que se necesitarán incorporar funcionalidades adicionales al proyecto, el uso del patrón Decorator hubiera sido más apropiado, ya que está diseñado específicamente para asignar responsabilidades a objetos individuales en lugar de toda una clase, lo cual facilita que múltiples clases puedan aprovechar dichas responsabilidades.

Referencias

Glide v4 : Fast and efficient image loading for Android. (2015). Github.io.

https://bumptech.github.io/glide/

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1993). Design patterns: Abstraction and reuse of object-oriented design. Capitulo 4. https://learning-oreilly-com.ezproxy.uniandes.edu.co/library/view/design-patterns-elements/0201633612/ch04.html#page_139