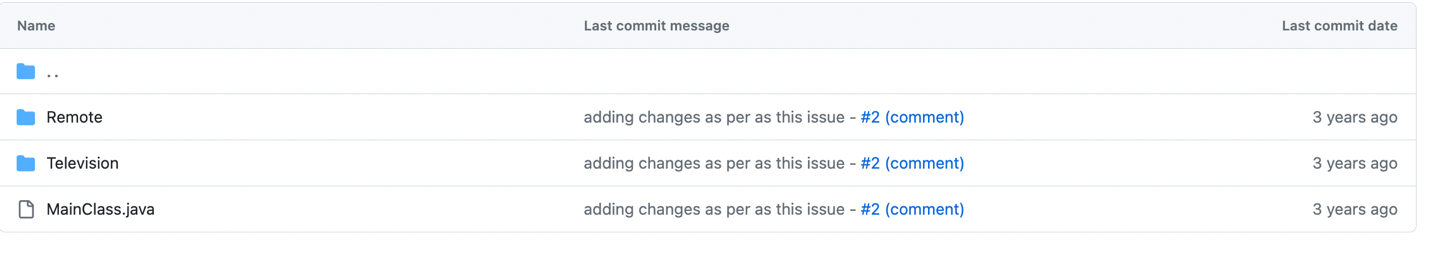
Sofia Vasquez- 202123910

**Taller 5: Documento de Análisis del Proyecto Design-Patterns**

Información general del proyecto:

Este proyecto se enfoca estudiar todos los patrones de diseño de "Head First Design Patterns". Cada patrón de diseño incluye código en Java junto con explicaciones y recursos adicionales para estudiar. En este caso, elegí el patrón Bridge entonces nos vamos a enfocar en lo relacionado a este. El patrón Bridge nos permite separar la abstracción de la implementación. Es un patrón de diseño estructural. En este proyecto, la estructura de general del diseño consiste en una empresa de controles remotos que debe producir controles remotos para televisores de todas las compañías y todos los modelos. Este se divide en dos, “Remote” y “Televisión". Comenzamos con RemoteControl, es una clase abstracta que define los métodos básicos que debe tener todo control remoto. RemoteSimple y RemoteSmart son 2 implementaciones concretas de la clase abstracta anterior. Ambos controles remotos definen algún comportamiento en los clics respectivos de los botones. ITelevision es una interfaz que define algunos comportamientos que debe tener toda televisión. Por último, SamsungSmartTelevision, SonyTelevision, XiaomiTelevision son implementaciones concretas de esta interfaz. Podemos observar que la implementación de algunos métodos en estas clases respectivas es diferente. Los principales desafíos de diseño que enfrenta este proyecto son que patrón de diseño escoger ya que toca tener en cuenta su propia complejidad. En este caso el problema fue como realizar la interfaz. Ya que si se hacía una de control remoto común y un conjunto de clases que implementan la interfaz, esto habría resultado en un gran número de clases. También habría significado que un cambio en la implementación de los televisores Sony habría llevado a cambios en todas las clases de combinación TV-Control remoto que involucraban Sony TV. Por lo tanto, era importante desacoplar las dos entidades: televisores y controles remotos.

URL: <https://github.com/piyush6348/Design-Patterns/tree/master/Structural%20Patterns/Bridge%20Pattern>

Información y estructura:

La estructura de este proyecto proporciona una clara separación entre la abstracción (RemoteControl) y la implementación (Television), cumpliendo así con los principios del patrón Bridge como lo podemos ver en la captura de pantalla.

Información general sobre el patrón:

El patrón que elegí es Bridge, este es un patrón de diseño estructural que permite dividir una clase grande o un grupo de clases relacionadas en dos jerarquías separadas (abstracción e implementación) que pueden desarrollarse independientemente la una de la otra. Para resolver este problema, el patrón Bridge cambia de la herencia a la composición del objeto. Esto significa que una de las dimensiones se extrae en una jerarquía de clases diferente, lo que significa que las clases originales solo referencian un objeto de la nueva jerarquía en lugar de tener todo su estado y sus funciones dentro de una clase. La abstracción (también conocida como interfaz) es una capa superior de control de una entidad. Esta capa no debe realizar ningún trabajo real por sí sola; en cambio, debe delegar el trabajo a la implementación (Bridge). Se utiliza el patrón Bridge cuando dos componentes de software están conectados directamente, sin embargo, existe una alta probabilidad de que uno de ellos cambie, lo que puede requerir la modificación del otro componente (*Bridge*).

Información del patrón aplicado al proyecto:

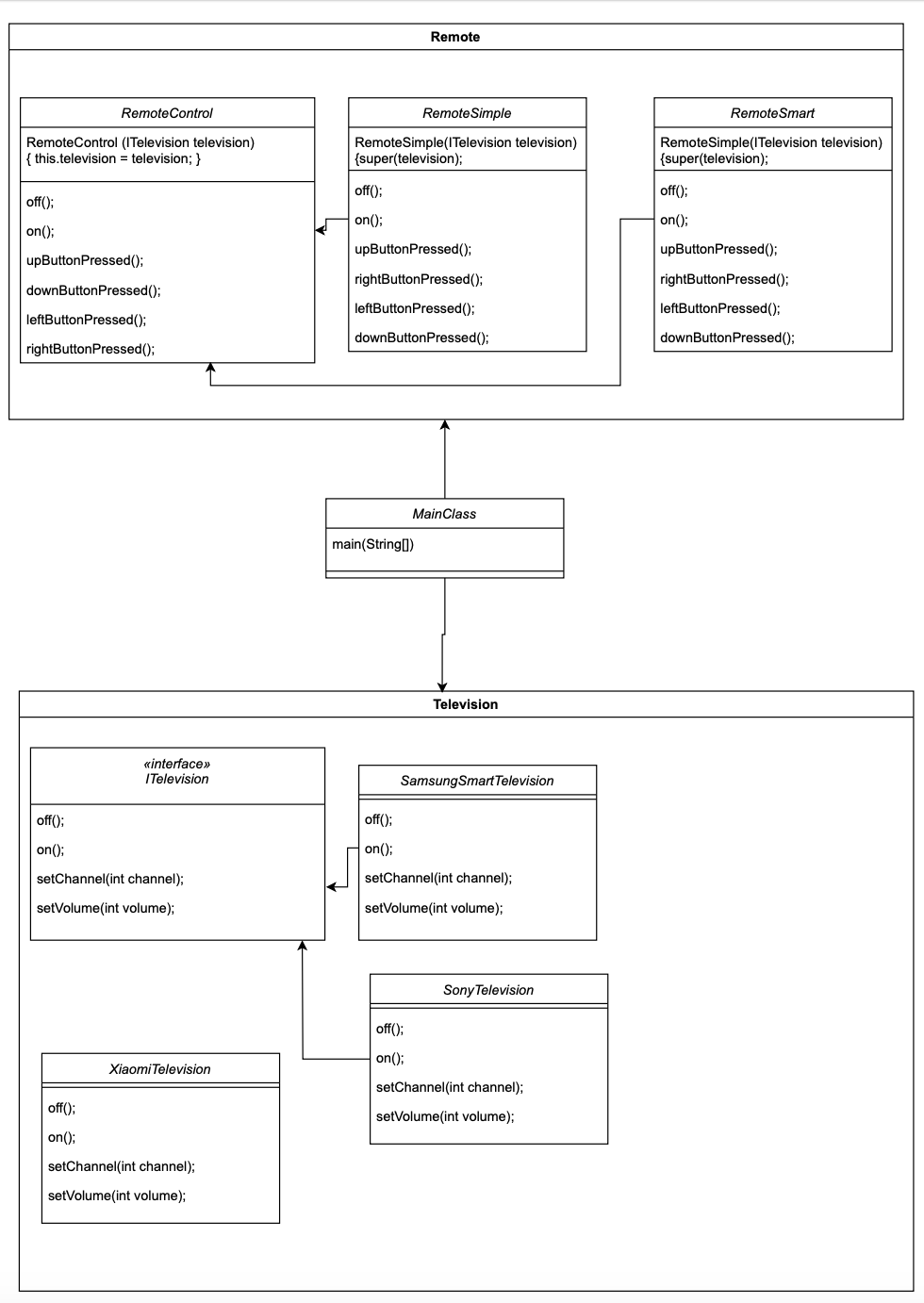
Como mencione anteriormente, una interfaz de control remoto común y un conjunto de clases que implementan la interfaz, algo así como SonyTelevisionSimpleRemote, SonyTelevisionSmartRemote, XiaomiTelevisionSimpleRemote, XiaomiTelevisionSmartRemote, etc es lo primero que se me hubiera ocurrido para resolver la idea de este proyecto. Pero esto habría resultado en un gran número de clases. También habría significado que un cambio en la implementación de los televisores Sony habría llevado a cambios en todas las clases de combinación TV-Control remoto que involucraban Sony TV. Por lo tanto, en este proyecto se decidido desacoplar las dos entidades: televisores y controles remotos. De esta forma se está utilizando el patrón Bridge porque desacoplo la abstracción de su implementación de manera que las dos puedan ser modificadas por separado sin necesidad de modificar la otra. El patrón Bridge se utiliza para separar la abstracción (RemoteControl) de su implementación (ITelevision). La interfaz ITelevision es la implementación de la abstracción que define las operaciones básicas de un televisor, y las clases RemoteSimple y RemoteSmart son las abstracciones que se conectan a través de la interfaz ITelevision.

¿Por qué tiene sentido haber utilizado el patrón en ese punto del proyecto?

Tiene sentido haber utilizado el patrón Bridge en este proyecto ya que era necesario desacoplar los controles remotos de las televisiones, permitiendo que ambas puedan variar independientemente sin depender de la otra. Las ventajas de haber utilizado este patrón son que, en primer lugar, presente flexibilidad. Al separar la abstracción de la implementación, cada uno puede cambiar sin afectar a la otra. También ayuda en la reutilización, debido a que las implementaciones de abstracción no están vinculadas a una implementación específica y se pueden reutilizar. No obstante, existen algunas desventajas. El patrón Bridge puede hacer que el código sea más complicado, especialmente si solo se requieren implementaciones simples. Además, puede haber un costo de aprendizaje para los desarrolladores que no están familiarizados con el patrón Bridge para comprender cómo funciona.

En cuanto a otras formas de resolver el problema, se me ocurren varias alternativas. Por ejemplo, se podria usar el patrón de diseño Adapter si sólo es necesario convertir una interfaz en otra. Si es necesario cambiar el comportamiento de un objeto en tiempo de ejecución, se podria usar el patrón de diseño Decorator. Sin embargo, ninguna de estas alternativas ofrece la misma flexibilidad para cambiar tanto las abstracciones como las implementaciones de forma independiente como lo hace el patrón Bridge.

Diagrama UML:



Bibliografia:

Bridge. (n.d.). <https://refactoring.guru/es/design-patterns/bridge>

*Bridge*. Reactive programming. (n.d.). <https://reactiveprogramming.io/blog/es/patrones-de-diseno/bridge>