JUSTIFICACIÓN DE LAS PRINCIPALES DECISIONES DE DISEÑO

I. Patrones de diseño:

Patrón Mediador:

- Clases involucradas:
  - PartidaAbstracta
  - Chaturanga
- Porque lo utilizamos:
  - El patrón mediador se utiliza debido a que es necesario establecer un punto intermedio entre la capa de datos y las interacciones del JugadorChaturanga, se utiliza la partida como un medio que encapsula los jugadores y el tablero para que interactúen entre ellos por medio de la misma partida. Aplicado al código es la clase Chaturanga la que se encarga de esta interacción mediante el método iniciarPartida.
  - Cabe resaltar que el método iniciarPartida no está del todo implementado debido a que el flujo del juego va empezar en este método, por lo que para este sprint se decidió enfocarnos en otras historias de usuario que no involucran todo el flujo de juego.
- Principios SOLID aplicados:
  - Single Responsibility

Método fabrica:

- Clases involucradas:
  - JugadorChaturanga
  - Chaturanga
  - FileManagerChaturanga
- Porque lo utilizamos:
  - ---------- ??? no estamos seguros
- Principios SOLID aplicados
  - :Single responsibility
  - Liskov substitution principle
  - Dependency inversion principle

Método Plantilla

- Clases involucradas:
  - JugadorChaturanga
  - Chaturanga
  - FileManagerChaturanga
- Porque lo utilizamos:
  - El patrón Método Plantilla se utiliza en estas clases ya que permite tener una clase abstracta que tiene un comportamiento base y partir de la misma permite crear clases específicas con comportamientos personalizados según las necesidades de las mismas, en este caso se

crea un jugador de chaturanga a partir de un jugador abstracto que permite definir un comportamiento específico y relacionado este juego.

- Principios SOLID aplicados:
    - Single Responsibility
    - Liskov substitution principle
    - Open-close
    - Interface segregation principle

Patrón Decorador

- Clases involucradas:
    - PiezaAbstracta, Elefante, Caballo, Peon, Rey, Barco.
- Porque lo utilizamos:
    - El patrón decorador se utiliza a la hora de implementar las clases específicas de PiezaAbstracta, ya que las clases derivadas (Elefante, Caballo, Peon, Rey y Barco) se extiende al agregar más atributos (nombre, color) y funcionalidad (la funcionalidad específica de getImageFilePath), sin necesidad de modificar la clase principal PiezaAbstracta.
    - Este patrón da paso a que se puedan implementar más piezas que deriven de PiezaAbstracta y así implementar piezas con funcionalidades diferentes sin tener que cambiar el comportamiento del juego.
- Principios SOLID aplicados:
    - Open-close

II. Respecto a la arquitectura

El programa implementa una arquitectura MVC:

- **Modelo**:

La capa de modelo la implementa la clase **Tablero**, la cual a su vez corre por debajo la clase **Casilla** y **PiezaAbstracta**, estos se consideran el modelo ya que los datos base para el funcionamiento del juego (las piezas sobre el tablero) se toman del contenido del tablero.

- **Vista**:

La capa de vista se implementa mediante la clase de **InterfazGraficaGenerica**, ya que esta es la que se encarga de implementar métodos genéricos que se encargan de dibujar la parte gráfica (lo que ve el usuario), específicamente **printImage, abrirNuevaInterfaz, agregarBoton** y **mostrarCuadroDialogo** son métodos genéricos que puede utilizar cualquier clase para dibujar algo en la vista..

- **Controlador**:

Sabemos que el controlador es aquella capa intermedia que comunica la vista con el modelo, en el programa el controlador corresponde a las clases **PartidaAbstracta** y **Chaturanga,** estas clases al implementar el patrón mediador comunica al JugadorAbstracto (clase que implementa una interfaz para JugadorChaturanga que a su vez es aquella que se encarga de recibir las jugadas del usuario) con la clase Tablero, la cual corresponde al modelo como vimos anteriormente.