# CONNECT FOUR TERVEZÉSI MINTA

## MVC (MODEL-VIEW-CONTROLLER)

A Connect Four játék tervezése az **MVC (Model-View-Controller)** mintát követi nagyobb részben, amely a következőképpen valósul meg:

1. **Model**: Az alkalmazás adatai és logikája, ide tartoznak az osztályok, amelyek a játék működését szabályozzák:
    - **Board**: A tábla állapotának kezelése, például korongok elhelyezése és győzelmi feltételek ellenőrzése. Az adatok kezeléséért és a játéklogika implementációjáért felel. Ez az osztály egy logikai táblát tartalmaz, és a szabályok betartásával valósítja meg a győzelmi feltételek ellenőrzését.
    - **Disc**: A játékban használt korongok tulajdonságait definiálja. Bár maga az osztály nem egy önálló Factory implementáció, az enum lehetővé teszi a játékban használt értékek (szimbólumok) szabványosított létrehozását és kezelését.

```
package classes.connectfour;

public enum Disc {  80 usages  Dorina8212
    YELLOW( symbol: "Y"),  32 usages
    RED( symbol: "R");  29 usages

    private final String symbol;  2 usages

    Disc(String symbol) {  4 usages  Dorina8212
        this.symbol = symbol;
    }

    @Override  Dorina8212
    public String toString() {
        return symbol;
    }
}
```

    - **GameState**: A játék állapotának teljes kezelése, mint például a játékosok váltása, lépések végrehajtása. Ez követheti a **State mintát** is, mivel a játék állapotát (pl. melyik játékos következik, győzelem történt-e) külön változókkal és metódusokkal kezeli.

2. **View**: A megjelenítésért felelős rész, amely a játék állapotát jeleníti meg a felhasználónak.
    - A displayBoard metódus például a tábla aktuális állapotát rajzolja ki a konzolra, vizuálisan is megjelenítve a játék helyzetét.

```
public static void displayBoard(Disc[][] grid) {  2 usages  Dorina8212
    for (int row = 0; row < grid.length; row++) {
        for (int col = 0; col < grid[row].length; col++) {
            if (grid[row][col] == null) {
                System.out.print("- ");
            } else if (grid[row][col] == Disc.YELLOW) {
                System.out.print("Y ");
            } else if (grid[row][col] == Disc.RED) {
                System.out.print("R ");
            }
        }
```

    -

3. **Controller**: Az interakciók vezérlése, amely lehetővé teszi a felhasználó számára, hogy utasításokat adjon, valamint kezeli a bemeneteket.
    - A Main osztály feladata, hogy a játékfolyamatot irányítsa, például a játékos lépéseit kezelje és továbbítsa az utasításokat a megfelelő modellekhez. A Main koordinálja a Model (Board és GameState) és View (pl. konzolra kirajzolás) rétegek közötti kommunikációt. Ezen kívül kezeli a mentések és visszatöltések metódusait, melyek a Momento tervezési mintát követik.

```
public static void saveBoardState(Board board, String filename) {  1 usage  Dorina8212
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {
        for (int row = 0; row < board.getRows(); row++) {
            for (int col = 0; col < board.getColumns(); col++) {
                Disc disc = board.getGrid()[row][col];
                writer.write(disc == null ? "-" : (disc == Disc.YELLOW ? "Y" : "R"));
                writer.write( str: " ");
            }
            writer.newLine();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public static GameState loadGameState(String filename) {  1 usage  Dorina8212
    try (Scanner scanner = new Scanner(new File(filename))) {
        Board board = new Board( rows: 6,  cols: 7);

        int row = 0;
        while (scanner.hasNextLine() && row < 6) {
            String line = scanner.nextLine();
            String[] tokens = line.split( regex: " ");
            for (int col = 0; col < tokens.length; col++) {
                if ("Y".equals(tokens[col])) {
                    board.getGrid()[row][col] = Disc.YELLOW;
                } else if ("R".equals(tokens[col])) {
                    board.getGrid()[row][col] = Disc.RED;
                }
            }
            row++;
        }
```