Tervezési minták

MVC (Model-View-Controller) mintázat

Az **MVC** mintázat a teljes alkalmazás szerkezetében felismerhető. Ez egy **architekturális minta**, amely az adatokat, a logikát és a megjelenítést különválasztja. Az MVC szerkezet tudatos tervezési minta nélkül is jól elkülöníti az adatokat (Model), a megjelenítést (View) és a vezérlést (Controller).

Hogyan jelenik meg a projektben?

- **Model**: Az adatok (pl. board állapot) és a játék működéséhez szükséges logika található a Model osztályban.
- **View**: A felhasználói felület megjelenítése a View osztályban történik.
- **Controller**: A játék vezérléséért és az események kezeléséért a Controller osztály felel.

Példa a kódból:

- A **Controller** irányítja a folyamatot, pl. amikor egy játékos lép:

```
public void playerMove(int col) {
    if (!model.isPlayerTurn()) return;

    // Player makes a move
    if (model.dropPiece(col, 1)) {
        view.updateBoard(model.getBoard());

        // Check if the player has won
        if (model.checkWin(1)) {
            JOptionPane.showMessageDialog(view, "Player wins!");
            dbManager.incrementScore(view.getPlayerName()); // Increment
the player's score
            view.disableBoard();
            return;
        }

        // Switch turn to the computer
        model.setPlayerTurn(false);

        // Check if the board is full before computer moves
        if (model.isBoardFull()) {
            JOptionPane.showMessageDialog(view, "It's a draw!");
            view.disableBoard();
            return;
        }

        // Computer makes a random move
        computerMove();
    } else {
        JOptionPane.showMessageDialog(view, "Column is full! Choose another
one.");
```

```
    }
}
```

- A **View** frissíti a felületet az új állapottal:

```
// Method to update the game board with the new state
public void updateBoard(int[][] boardState) {
    for (int row = 0; row < numRows; row++) {
        for (int col = 0; col < numCols; col++) {
            if (boardState[row][col] == 1) {
                boardButtons[row][col].setBackground(Color.YELLOW); //
Player's move
            } else if (boardState[row][col] == 2) {
                boardButtons[row][col].setBackground(Color.RED); //
Computer's move
            } else {
                boardButtons[row][col].setBackground(Color.WHITE); // Empty
            }
        }
    }
}
```

Command minta

A menüelemek kezelése (pl. “New Game”, “Save Game”, “Exit”) **Command** mintaként értelmezhető, mert minden menüelemhez külön logikát rendelsz az ActionListener-ek segítségével.

Hogyan jelenik meg a projektben?

- Minden menüelem (JMenuItem) rendelkezik saját művelettel, amelyet egy ActionListener segítségével hajt végre.

Példa a kódból:

```
// Add action listener for New Game menu item
newMenuItem.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        resetGame(); // Reset and update the game board when "New Game" is
clicked
    }
}
```

Ez a Command minta egyszerűbb változata, ahol:

- A **Command** a resetGame() metódus.
- A **Invoker** a JMenuItem (pl. “New Game”).
- Az ActionListener működése a Command mintát idézi, mivel elválasztja a parancs végrehajtását a hívótól.

Factory minta

A játék inicializálása során részben felismerhető a **Factory** minta. A Model példányosítása egyfajta **Factory Method**-ként működik, mivel a létrehozása dinamikus, és az oszlopok/sorok számától függ.

Hogyan jelenik meg a projektben?

A Main osztályban a Model példányosítása dinamikusan történik a View által megadott beállítások alapján (Ez ugyan nem egy explicit Factory minta, de a dinamikus példányosítás miatt annak egyszerűbb formájaként értelmezhető):

```
// Create the Model with the row and column sizes from the View
Model model = new Model(view.getNumRows(), view.getNumCols());
```

Data Access Object (DAO) minta

A **DatabaseManager** osztályban egyértelműen megjelenik a **DAO** (Data Access Object) minta. Ez egy **szerkezeti minta**, amely az adatbázis műveleteket egyetlen osztályba szervezi, ezzel elválasztva az adatkezelést az alkalmazás többi részétől.

Hogyan jelenik meg a projektben?

A DatabaseManager biztosítja az összes adatbázis-műveletet, pl. játékállapot mentése és betöltése:

```
// Save game state (board)
public void saveGameState(String boardState) {
    String sql = "INSERT OR REPLACE INTO game_state (id, board) VALUES (1,
?)";

    try (Connection conn = connect();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, boardState);
        pstmt.executeUpdate();
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
}
```

A DatabaseManager:

- Elkülöníti az adatbázis-kezelés logikáját.
- Könnyen lecserélhető más adatforrásra, ha szükséges (pl. fájl vagy REST API).

Template Method minta

A updateGameBoard() metódusban felismerhető a Template Method minta, amely egy algoritmus sablont határoz meg, de bizonyos részeket alosztályokra vagy más metódusokra bíz.

Hogyan jelenik meg a projektben?

A játékpanel frissítése egy sablon szerint történik, ahol az egyes cellák logikája külön hívásban van megadva:

```
// Method to update the game board based on the current numRows and numCols
private void updateGameBoard() {
    // Remove the current panel and re-create the grid
    if (gamePanel != null) {
```

```
        remove(gamePanel);
    }

    gamePanel = new JPanel();
    gamePanel.setLayout(new GridLayout(numRows, numCols));
    boardButtons = new JButton[numRows][numCols];

    // Add buttons for each cell in the grid
    for (int row = 0; row < numRows; row++) {
        for (int col = 0; col < numCols; col++) {
            JButton cellButton = new JButton();
            cellButton.setPreferredSize(new Dimension(50, 50)); // Size of
each cell
            cellButton.setBackground(Color.WHITE); // Default background
color
            cellButton.addActionListener(new ColumnListener(col)); //
Action listener for moves
            boardButtons[row][col] = cellButton;
            gamePanel.add(cellButton);
        }
    }

    // Add the updated game panel to the main window
    add(gamePanel, BorderLayout.CENTER);

    // Refresh the window to reflect changes
    revalidate();
    repaint();
}
```