Tervezési Minták

1. MVC (Model-View-Controller) Minta

Az **MVC** egy olyan architekturális tervezési minta, amely az alkalmazás különböző funkcionális részeit elkülöníti:

- **Model:** Az adatok kezelése és a logikai működés itt történik.
- **View:** Az adatok megjelenítéséért felelős.
- **Controller:** Az események kezelése és az alkalmazás irányítása tartozik ide.

Ez a minta segít az alkalmazás áttekinthetőségében és karbantarthatóságában.

Megvalósítás a projektben:

- **Model:** Az osztály felelős például a játéktábla állapotának tárolásáért és a játék működéséhez szükséges logikáért.
- **View:** A felhasználói felületet hozza létre és frissíti.
- **Controller:** Az események kezelése és a játékirányítás itt történik.

Példa a projektből

- **A Controller irányítja az eseményeket:**

```
public void playerMove(int col) {
    if (!model.isPlayerTurn()) return;
    if (model.dropPiece(col, 1)) {
        view.updateBoard(model.getBoard());
        if (model.checkWin(1)) {
            JOptionPane.showMessageDialog(view, "Player wins!");
            dbManager.incrementScore(view.getPlayerName());
            view.disableBoard();
            return;
        }
        model.setPlayerTurn(false);
        if (model.isBoardFull()) {
            JOptionPane.showMessageDialog(view, "It's a draw!");
            view.disableBoard();
            return;
        }
        computerMove();
    } else {
        JOptionPane.showMessageDialog(view, "Column is full! Choose another
one.");
    }
}
```

- **A View frissíti a játék megjelenítését:**

```
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);
            } else if (boardState[row][col] == 2) {
                boardButtons[row][col].setBackground(Color.RED);
            } else {
                boardButtons[row][col].setBackground(Color.WHITE);
            }
        }
    }
}
```

- **Command Minta**

Az alkalmazás menüelemei, például a „**New Game**”, „**Save Game**” és „**Exit**”, a **Command minta** logikáját követik. Minden menüponthoz külön művelet társul, amelyeket **ActionListener-ek** segítségével valósítunk meg.

- Megvalósítás a projektben:

- A JMenuItem (pl. "New Game") meghatározza az adott parancsot.
- Az **ActionListener** a hozzá tartozó művelet végrehajtását kezeli.
- **Példa a projektből**

```
new MenuItem.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        resetGame();
    }
});
```

- **Command:** A resetGame() metódus.
- **Invoker:** A JMenuItem komponens.

- **Factory Minta**

Az alkalmazás inicializálásánál felismerhető a **Factory minta** használata, amely az objektumok dinamikus létrehozására szolgál.

- **Megvalósítás a projektben:**

A Model példányosítása a View által meghatározott paraméterek alapján történik, például a sorok és oszlopok számától függően:

Model model = new Model(view.getNumRows(), view.getNumCols());

2. Data Access Object (DAO) Minta

A **DAO minta** lehetővé teszi az adatbáziskezelési logikák egyetlen osztályban való elkülönítését. Ez az adatbázis-műveletek karbantarthatóságát és az adatok forrásának lecserélhetőségét biztosítja.

- **Megvalósítás a projektben:**

A DatabaseManager osztály felelős az adatbázis-műveletekért, például a játékállapot mentéséért:

```
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());
   }
}
```

3. Template Method Minta

A **Template Method minta** egy algoritmus sablont határoz meg, miközben bizonyos lépéseket alosztályokra vagy külön metódusokra bíz.

- **Megvalósítás a projektben:**

A updateGameBoard() metódus példája:

```
private void updateGameBoard() {
   if (gamePanel != null) {
      remove(gamePanel);
   }
   gamePanel = new JPanel();
   gamePanel.setLayout(new GridLayout(numRows, numCols));
   boardButtons = new JButton[numRows][numCols];
   for (int row = 0; row < numRows; row++) {
      for (int col = 0; col < numCols; col++) {
         JButton cellButton = new JButton();
         cellButton.setPreferredSize(new Dimension(50, 50));
         cellButton.setBackground(Color.WHITE);
         cellButton.addActionListener(new ColumnListener(col));
         boardButtons[row][col] = cellButton;
         gamePanel.add(cellButton);
      }
   }
   add(gamePanel, BorderLayout.CENTER);
   revalidate();
   repaint();
}
```