ПОРІВНЯННЯ ІМПЕРАТИВНИХ, ФУНКЦІОНАЛЬНИХ ТА ЛОГІЧНИХ МОВ ПРОГРАМУВАННЯ ПРИ РЕАЛІЗАЦІЇ ГРИ У ШАШКИ

Презентація до курсової роботи

Хоменко Максим Олександрович, ІПЗ-3
Горборуков Вячеслав Вікторович

Мета дослідження

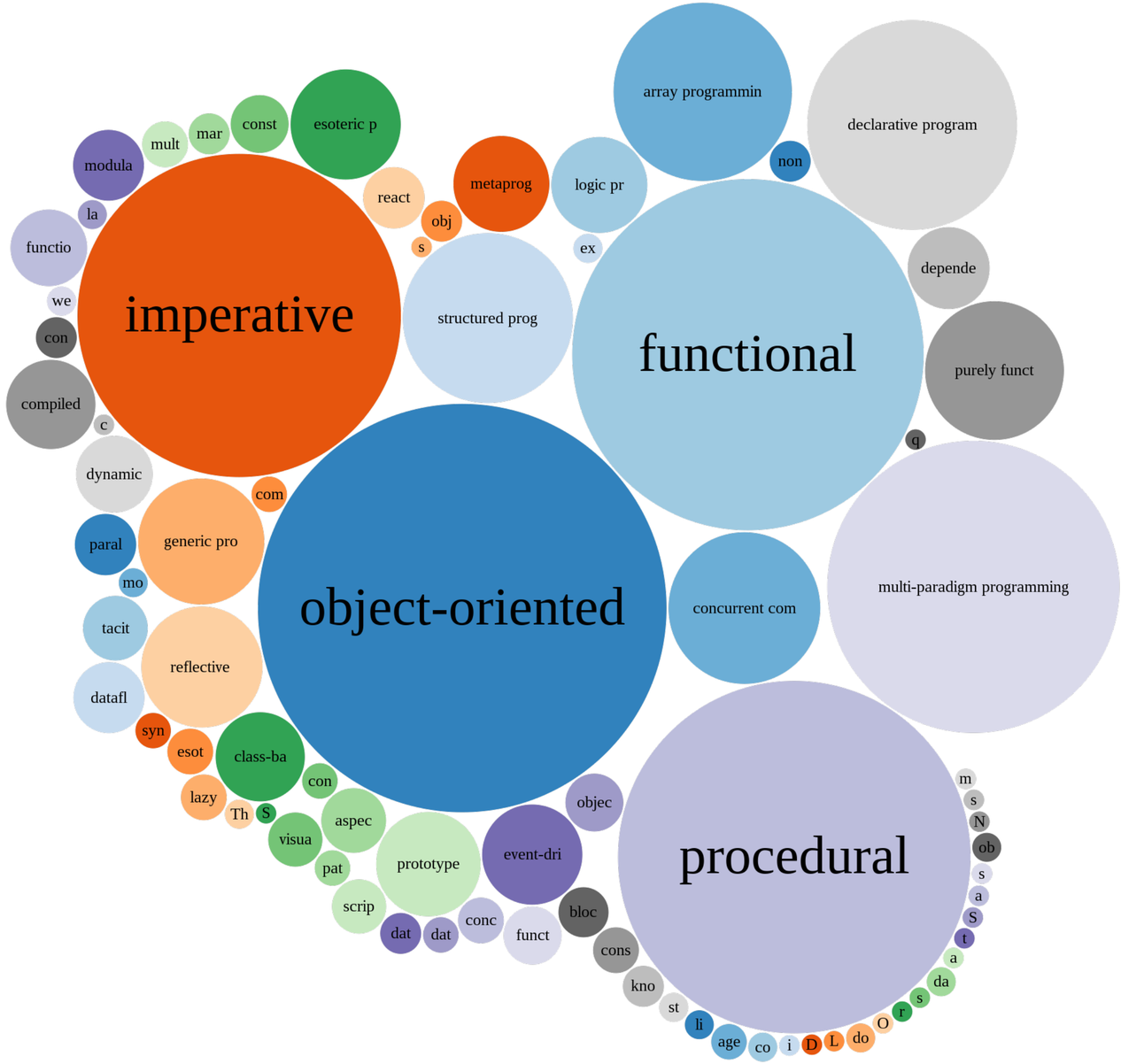
Порівняти три мови програмування, що належать до різних парадигм - імперативної, функціональної та логічної - шляхом реалізації гри в шашки, порівняти їхні переваги та недоліки в контексті певних завдань.

Теоретична основа

Що таке парадигма?

Теоретична основа

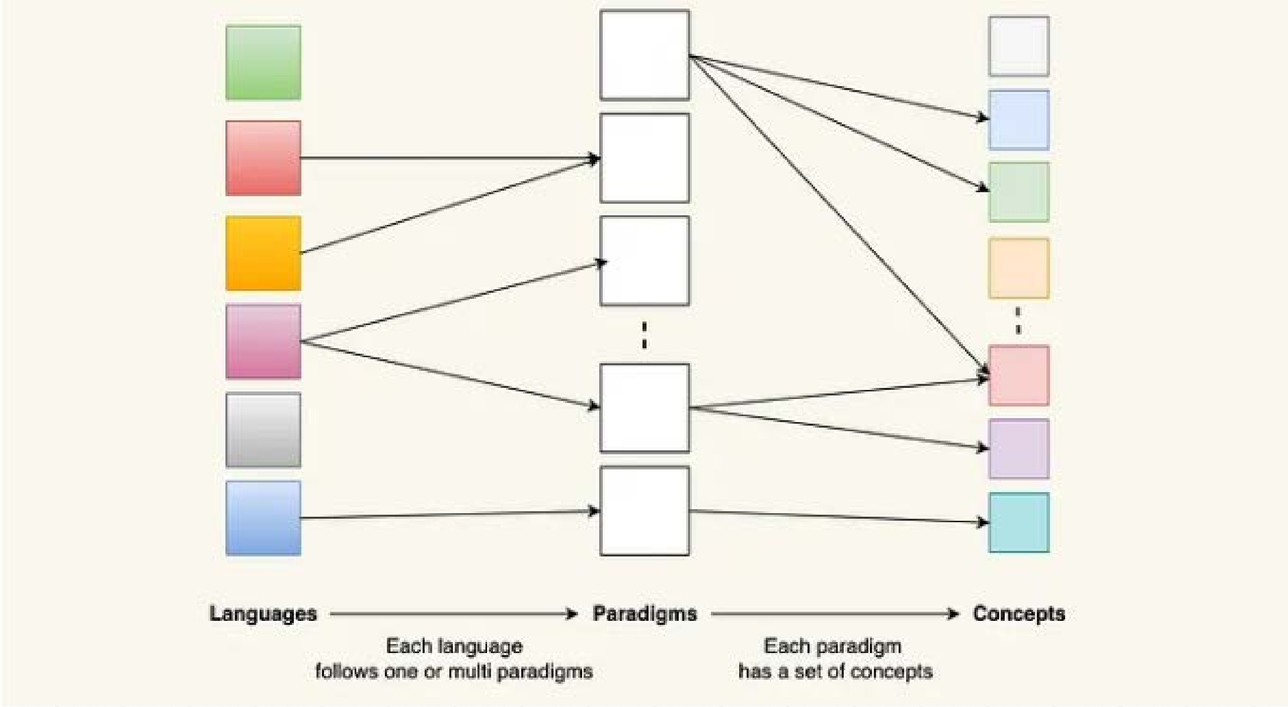
- **Парадигма** - спосіб структурування й концептуалізації комп'ютерної програми
- Парадигма задає те, як працює програма і як виглядатиме програмний код

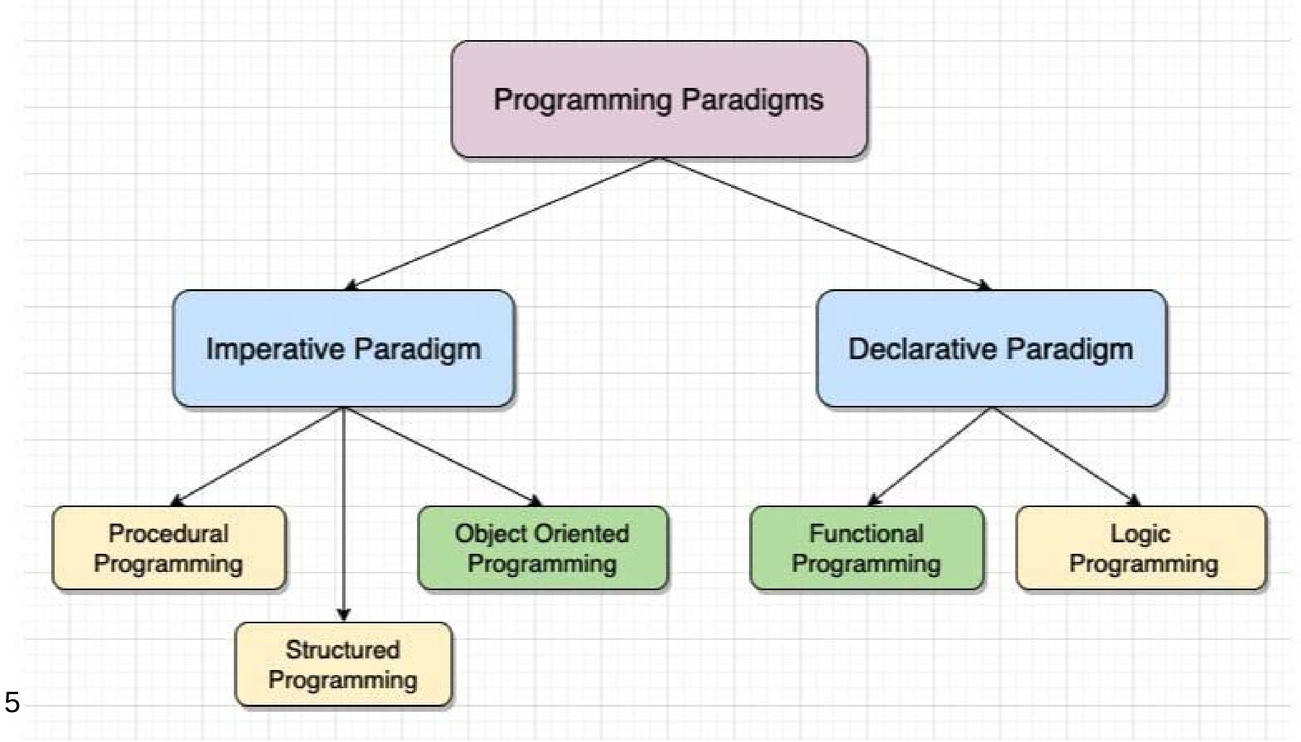


Що таке парадигма?

Теоретична основа

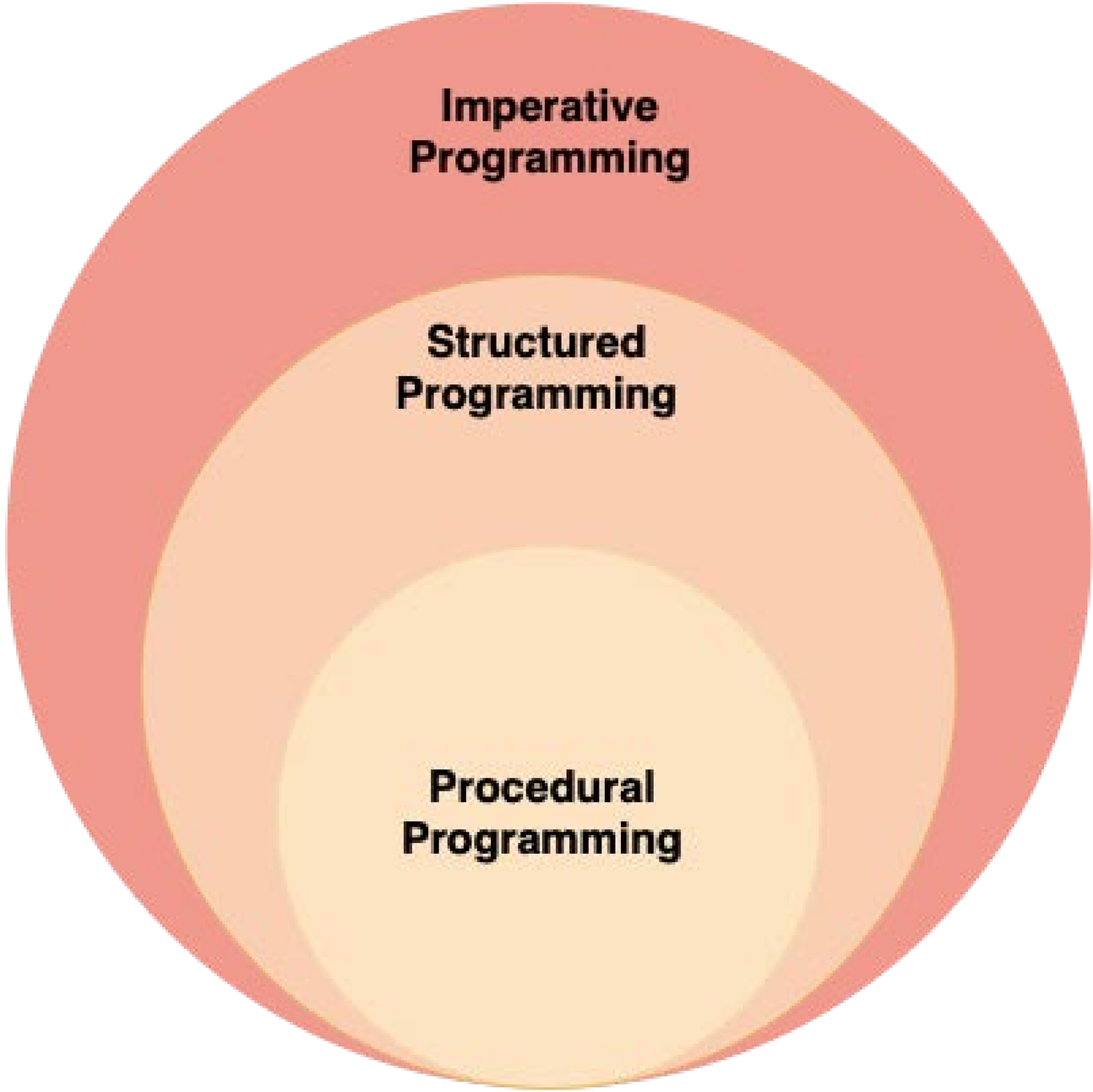
- Парадигм існує багато
- Та всі застосовують певні концепти
- До основних можна віднести **імперативну** й **декларативну**
  - До декларативної ж можна віднести **функціональну** й **логічну**





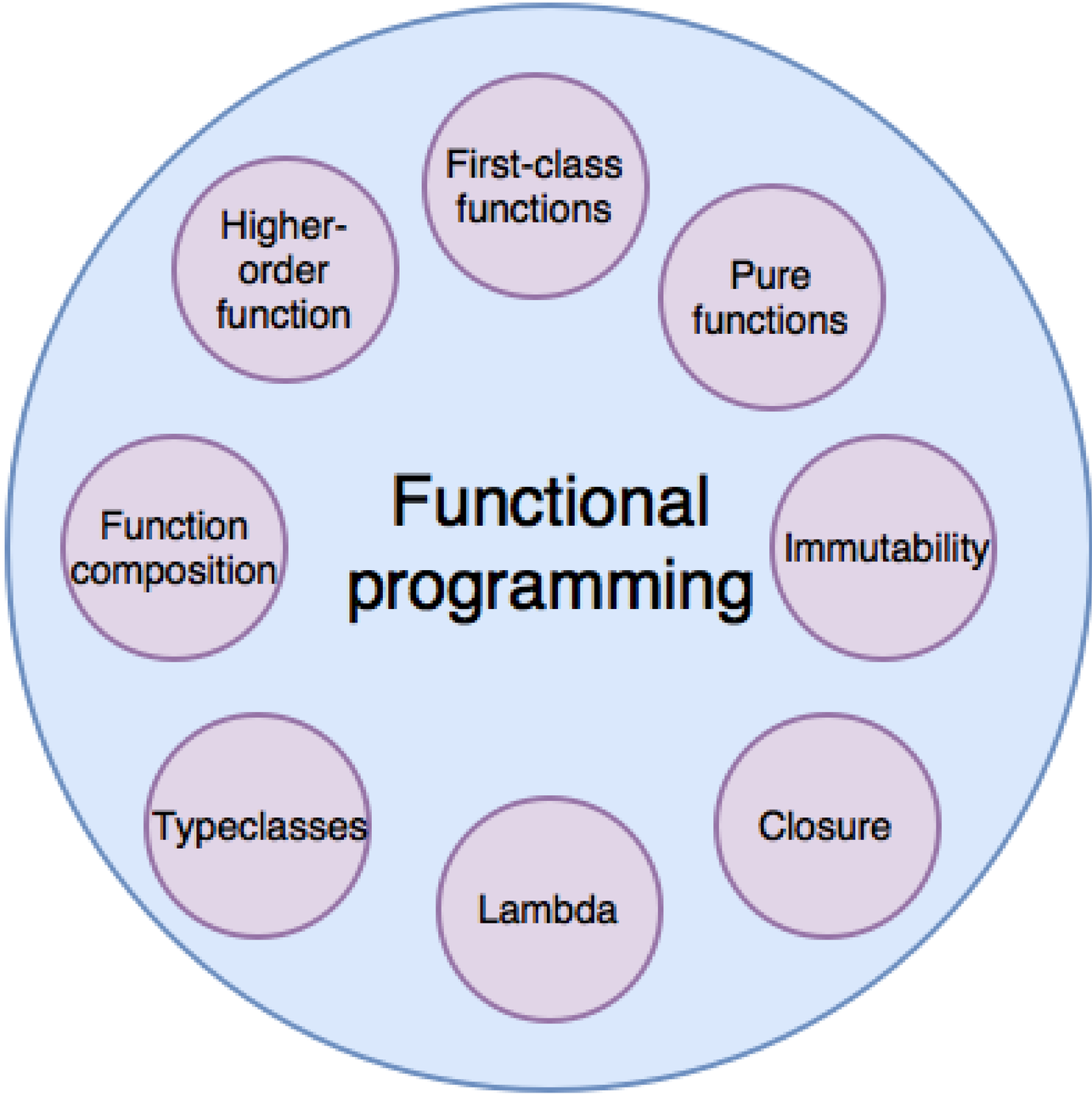
Імперативна парадигма
Теоретична основа

- Найстаріша парадигма
- Наближена до системи
- Вказує, **як** щось зробити
- Підтипи
  - Структурований
  - Процедурний
  - Модульний



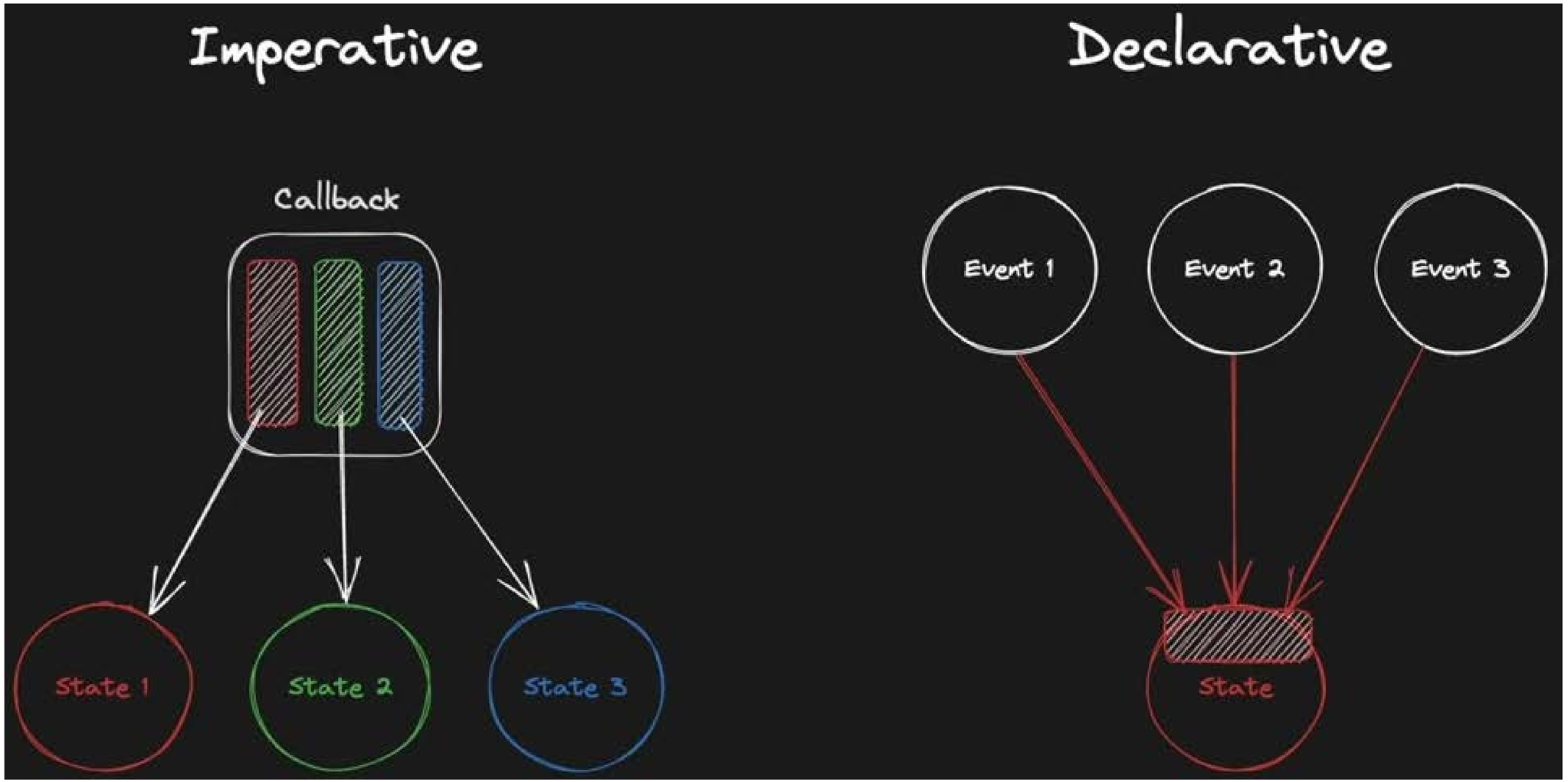
Декларативна парадигма
Теоретична основа

- Вказує, **що** зробити
- Декларативними є
  - Функціональна
  - Логічна
- **Функціональна**
  - Основна сила - незмінність **(immutability)**
- **Логічна**
  - Основна сила - логіка на базі правил **(rule-based logic)**



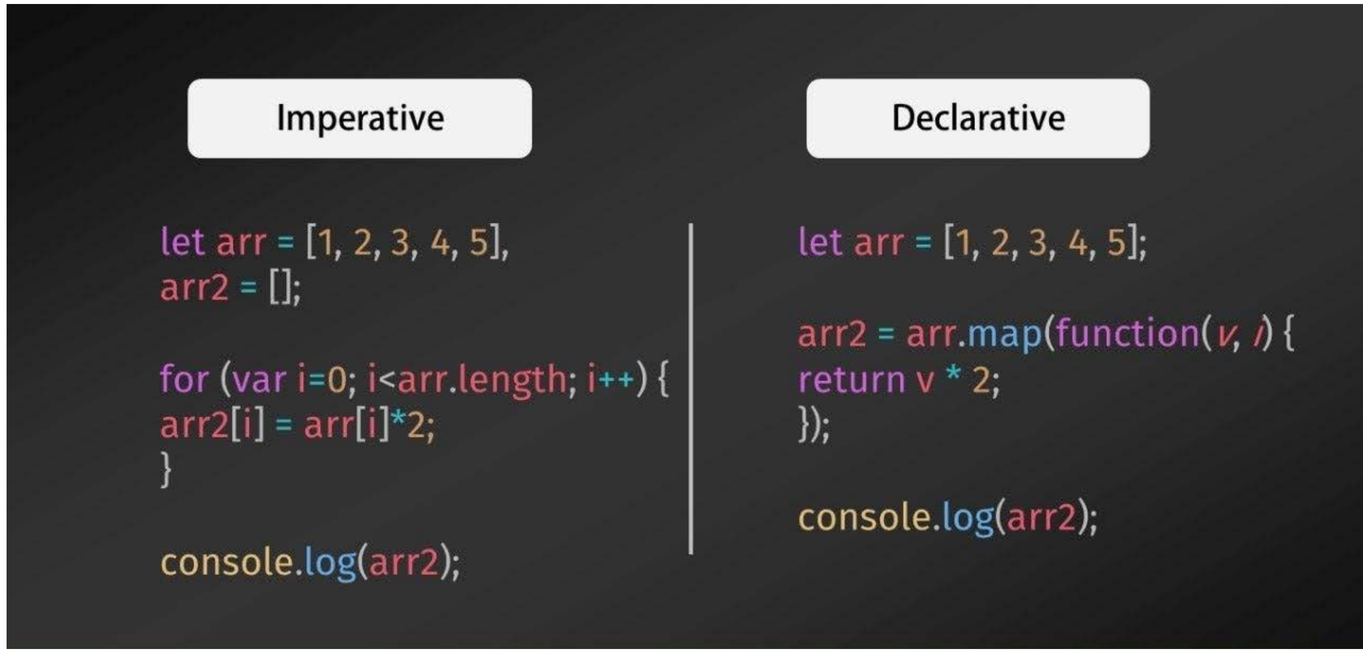
Порівняння парадигм

Теоретична основа



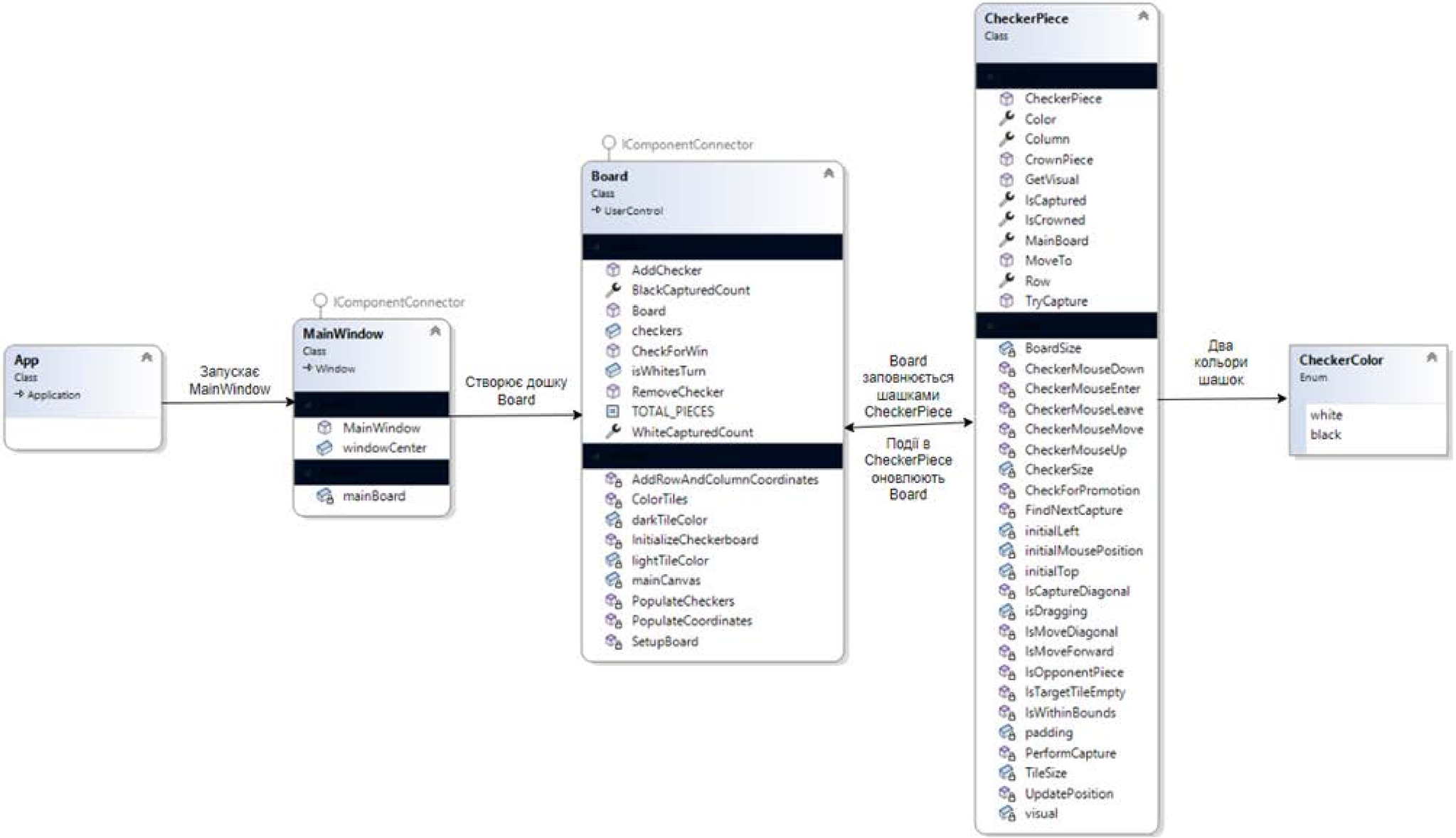
Багатопарадигмальні мови
Теоретична основа

- Мови, які підтримують кілька парадигм
- Об'єднують концепції з них, щоб максимізувати функціональність
- Багатопарадигмальна мова - своєрідний "мультитул"



Реалізація ігор в шашки

Реалізація гри в шашки на базі імперативної парадигми
(Мова програмування C#)



Реалізація гри в шашки на базі функціональної парадигми (Мова програмування Haskell)

app\Main.hs

Запускає програму

```
module Main where

import Gameplay

-- Launch the game
main :: IO ()
main = playGame
```

app\Gameplay.hs

Файл відповідає за ігровий процес, переміщення шашками і т.д.

Код: створює ігрову дошку, ініціалізує стан гри

```
-- Starts the game
playGame :: IO ()
playGame = do
    images <- loadImages
    let initState = GameState createInitialBoard
Nothing WhitePlayer Nothing
    play
      (InWindow "Checkers" (windowWidth,
windowHeight) (100, 100))
      white
      30
      initState
      (drawGameState images)
      handleEvent
      update
```

app\UserInterface.hs

Файл відповідає за графічні елементи гри

Код: завантажує всі потрібні зображення

```
-- Loads images used in the game
loadImages :: IO GameImages
loadImages = do
    whiteImg <- loadBMP "images/whiteChecker.bmp"
    blackImg <- loadBMP "images/blackChecker.bmp"
    crownedWhiteImg <- loadBMP
"images/crownedwhiteChecker.bmp"
    crownedBlackImg <- loadBMP
"images/crownedblackChecker.bmp"
    whiteWinsImg <- loadBMP
```

app\Types.hs

Файл відповідає за типи & структури даних

Код: ініціалізація дошки, заповнення її шашками

```
-- Function to create the initial board
configuration
createInitialBoard :: Board
createInitialBoard = [
    [Empty, Occupied BlackPiece, Empty, Occupied
BlackPiece, Empty, Occupied BlackPiece, Empty,
Occupied BlackPiece],
    [Occupied BlackPiece, Empty, Occupied
BlackPiece, Empty, Occupied BlackPiece, Empty,
Occupied BlackPiece, Empty],
    [Empty, Occupied BlackPiece, Empty, Occupied
```

app\GameLogic.hs

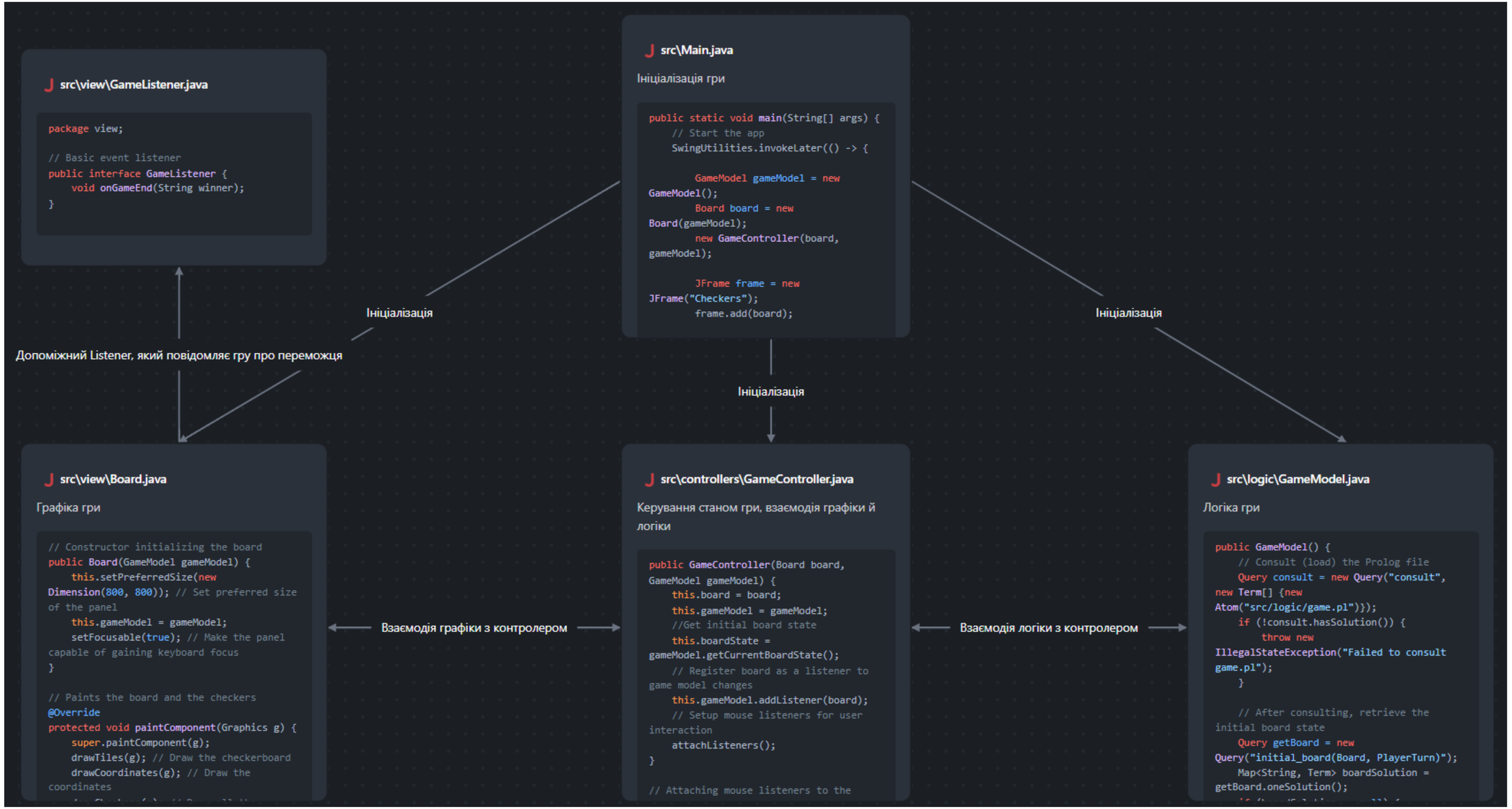
Файл відповідає за валідацію рухів і підтримку правил гри

Код: через handleEvent йде перевірка рухів / валідація і т.д.

```
-- Handles the dropping of a piece on the board
based on mouse input
dropPiece :: Point -> Maybe SelectedPiece ->
Board -> Player -> (Board, Bool, Maybe Player)
dropPiece mousePos (Just sp) brd currPlayer =
    let dropPos = pixelsToBoardCoords mousePos
        originalPos = originalPosition sp
        curPiece = piece sp
        isValid = isValidMove brd originalPos
dropPos curPiece currPlayer
    in if isValid then
        -- Check if the move involves capturing
```

Реалізація гри в шашки на базі логічної парадигми

(Мови програмування Prolog & Java)



Порівняльний аналіз

Реалізація функціоналу ігор

Порівняльний аналіз

Реалізація руху шашки на імперативній мові

```
// Moves the checker to a new position on the board.
2 references
public async void MoveTo(int newRow, int newColumn)
{
    // Inverse ifs to avoid nesting
    if (await TryCapture(newRow, newColumn))
    {
        // handle capture here
        return;
    }

    // Run move through rules to validate
    if (
        (!IsMoveDiagonal(newRow, newColumn)) ||
        !IsWithinBounds(newRow, newColumn) ||
        (!IsCrowned && !IsMoveForward(newRow)) ||
        !IsTargetTileEmpty(newRow, newColumn))
    {
        // If rules fail, restore checker's current position
        UpdatePosition(Row, Column);
        return;
    }

    // If rules pass, update player position, change turns
    UpdatePosition(newRow, newColumn);
    MainBoard.isWhitesTurn = !MainBoard.isWhitesTurn;

    // Check if a checker has reached the opposite end of the board, and crown it if it has
    CheckForPromotion();
}
```

Реалізація руху шашки на логічній (декларативній) мові

```
% Update the board state
move_checker(Board, StartRow, StartCol, EndRow, EndCol, Piece, PlayerTurn, NewBoard, NextPlayerTurn, Winner) :-
    valid_move(Board, StartRow, StartCol, EndRow, EndCol, Piece, PlayerTurn, CaptureRow, CaptureCol), % Validation
    update_board(Board, StartRow, StartCol, EndRow, EndCol, CaptureRow, CaptureCol, Piece, NewBoard), % Updating the board
    toggle_turn(PlayerTurn, NextPlayerTurn), % Changing player turns
    check_win_condition(NewBoard, Winner). % Check if one of the players wins
```

- Логічна (декларативна) реалізація:

"Виконай предикат руху з такими параметрами й поверни стан дошки при виконанні.
Залежно від правил і параметрів, поверни новий стан гри (коронуй шашку чи не перемикай хід гравця, якщо хід недійсний)."

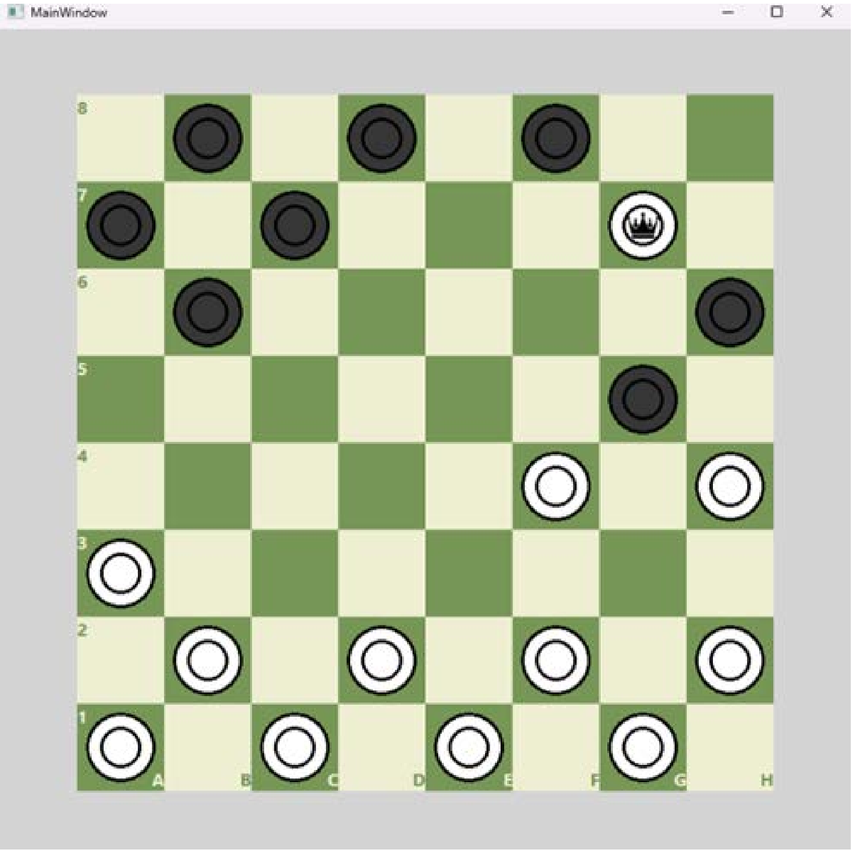
- Імперативна реалізація:

"Виконай серію перевірок умов перед виконанням руху.
Якщо умови не відповідають визначеним критеріям, анулюй рух.
Інакше, виконай рух і онови дошку."

Розробка графічних елементів ігор

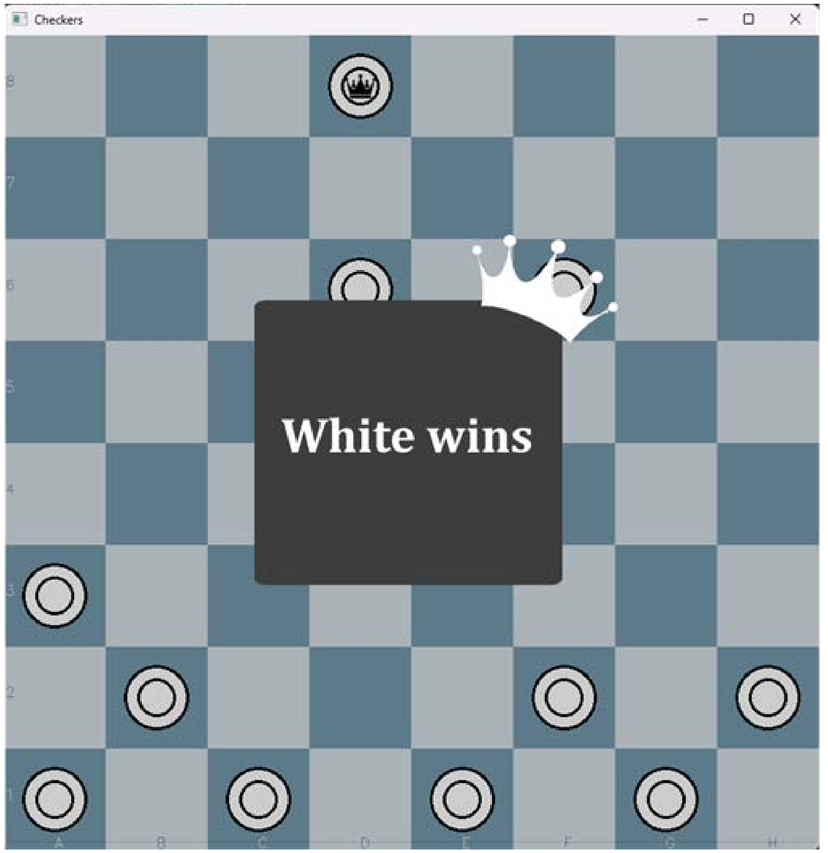
Порівняльний аналіз

Графічне представлення гри на імперативній мові C#



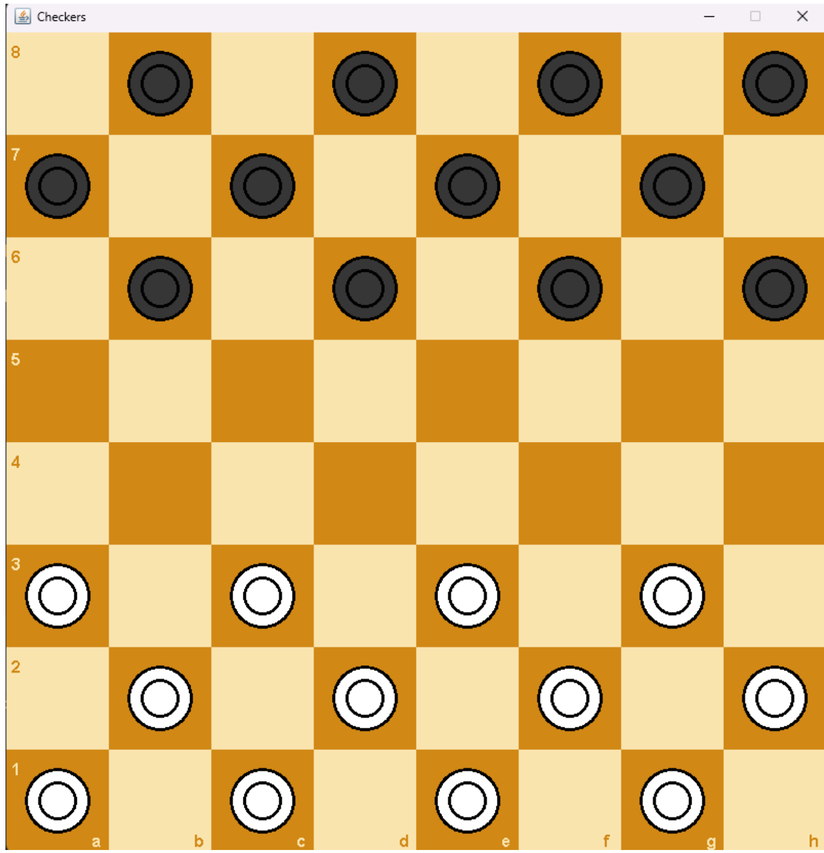
Найпростіша реалізація, адже C# і WPF тісно інтегровані. Імперативний підхід дає чіткі директиви щодо вигляду гри.

Графічне представлення гри на функціональній мові Haskell



Складніша реалізація через високу абстрактність мови Haskell, а також обмеження обраної графічної бібліотеки

Графічне представлення гри на мові Java (до логічної реалізації на мові Prolog)



Підтримка графічних компонентів у логічному програмуванні суттєво обмежена; використання іншої мови стало необхідним для розробки інтерфейсу

Підтримуваність програмного коду
Порівняльний аналіз

Реалізація координат ігрової дошки на імперативній мові
(Зображення містить реалізацію координатів лише до рядків)

```
// Adds coordinates to each tile for better user interaction.
1 reference
private void PopulateCoordinates()
{
    int rowNum = 8;
    int colNum = 8;

    Style darkTileStyle = (Style)MainBoard.FindResource("DarkTileStyle");
    Style lightTileStyle = (Style)MainBoard.FindResource("LightTileStyle");

    // Populate row coordinates (numbers)
    for (int i = 1; i <= rowNum; i++)
    {
        TextBlock rowCoordinate = new()
        {
            Text = i.ToString(),
            HorizontalAlignment = HorizontalAlignment.Left,
            VerticalAlignment = VerticalAlignment.Top
        };

        SolidColorBrush tileBackground = ((i % 2 != 0) ?
            (SolidColorBrush)darkTileStyle.Setters.OfType<Setter>().First().Value :
            (SolidColorBrush)lightTileStyle.Setters.OfType<Setter>().First().Value);
        rowCoordinate.Foreground = new SolidColorBrush((tileBackground.Color == darkTileColor) ?
            lightTileColor :
            darkTileColor);

        rowCoordinate.FontWeight = FontWeights.Bold;
        rowCoordinate.FontSize = 16;

        Grid.SetRow(rowCoordinate, rowNum - i);
        Grid.SetColumn(rowCoordinate, 0);
        MainBoard.Children.Add(rowCoordinate);
    }
```

Імперативно пояснює, **як** малювати координати рядків
Громіздка, багато шаблонного коду

Реалізація координат ігрової дошки на функціональній (декларативній) мові
(Зображення містить реалізацію координатів і до рядків, і до стовпчиків)

```
-- Draws coordinate labels on the board
drawCoordinates :: Picture
drawCoordinates = pictures $ rowCoords ++ colCoords
  where
    coordScale = 0.1
    xOffset = -5.0
    yOffset = -5.0
    halfBoard = 4 * tileSize

    rowCoords = [ translate (-halfBoard - xOffset / 2) (fromIntegral y * tileSize - halfBoard + tileSize / 2) $
                  scale coordScale coordScale $
                  color (contrastColor (0, 7 - y)) $
                  text (show (1 + y))
                | y <- [0..7] ]

    colCoords = [ translate (fromIntegral x * tileSize - halfBoard + tileSize / 2) (-halfBoard - yOffset / 2) $
                  scale coordScale coordScale $
                  color (contrastColor (x, 7)) $
                  text [colChar]
                | (colChar, x) <- zip ['A'..'H'] [0..7] ]

    -- Function to determine the contrasting color for the coordinates
    contrastColor :: (Int, Int) -> Graphics.Gloss.Color
    contrastColor (x, y) = if even (x + y) then darkTileColor else lightTileColor
```

Декларує, **що** таке координати рядків і стовпчиків
Лаконічна, більш абстрактна

Висновки

Висновки
Порівняльна таблиця

Оцінка парадигм для реалізації окремих складових гри в шашки

Парадигма	Комплексність реалізації функціоналу гри	Складність розробки графічного інтерфейсу	Підтримуваність коду, простота відладки	Потенціал до розширення функціоналу	Продуктивність гри
Імперативна					
Функціональна					
Логічна					

Зелений колір вказує, що парадигма гарно порається з завданням; жовтий вказує, що вона адекватно порається; червоний вказує, що парадигма не рекомендується.

Висновки

Яка парадигма найкраща?

- Варто розуміти, яка парадигма працює найкраще з певним завданням
- Загалом — усі парадигми мають своє призначення
- Для цієї задачі найкраща **імперативна парадигма**
  - **Явне керування станами** графічної та логічної складових спростило процес розробки
  - Громіздкість коду не виявилась великим мінусом

Q&A

Чи є питання?

Дякую за увагу!