

TUGAS BESAR

PEMROGRAMAN BERORIENTASI OBJEK

TEKNIK INFORMATIKA UNISSULA

---

Nama : Achmad Khusna Faizuddin

Nim : 32602200026

Kelas : TIF -A

**A. Konsep *Inheritance*, *Polymorphism*, *Encapsulation*, *Getter and Setter*, *Interface*.**

*Game* TicTacToe merupakan sebuah *game* yang berisikan kode program yang dibuat dengan menyertakan sebuah konsep *Inheritance*, *Polymorphism*, *Encapsulation*, *Getter and Setter*, *Interface*. Berikut adalah penjelasan lebih rinci untuk setiap konsepnya:

1. *Inheritance* (Pewarisan):

Konsep ini terlihat dalam kelas *PlayerImpl* yang merupakan *subclass* dari kelas abstrak *Player*. *PlayerImpl* mewarisi atribut dan metode dari kelas *Player* yang abstrak.

2. *Encapsulation*:

Atribut *name* dan *symbol* pada kelas *Player* di-set sebagai *private*, sehingga mereka tidak dapat diakses langsung dari luar kelas. Akses ke atribut tersebut hanya dimungkinkan melalui metode *getter* dan *setter* yang telah disediakan.

3. *Getter and Setter*:

Metode *getter* (*getName* dan *getSymbol*) dan *setter* (*setName* dan *setSymbol*) digunakan untuk mendapatkan dan mengubah nilai atribut *privat* (*name* dan *symbol*) pada kelas *Player*. Ini memastikan kontrol akses yang lebih baik terhadap atribut tersebut.

4. *Polymorphism*:

*Polymorphism* terlihat dalam penggunaan metode abstrak *makeMove()* yang diimplementasikan oleh kelas TicTacToe. Setiap pemain (*Player*) dapat mengimplementasikan metode ini sesuai dengan perilaku uniknya.

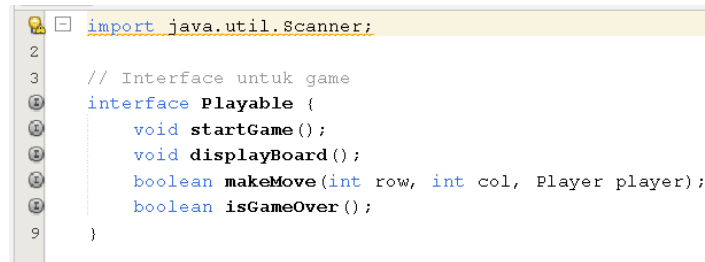
## 5. *Interface* (Antarmuka):

Ada penggunaan antarmuka *Playable*, yang menyatakan metode-metode yang harus diimplementasikan oleh kelas yang menggunakan antarmuka tersebut. Kelas *TicTacToe* mengimplementasikan antarmuka *Playable*, dan oleh karena itu, harus menyediakan implementasi untuk semua metodenya.

Demikianlah, kode tersebut mencakup beberapa konsep dasar dalam pemrograman berorientasi objek, memanfaatkan konsep-konsep seperti pewarisan, *enkapsulasi*, *polimorfisme*, *getter* dan *setter*, serta antarmuka.

## B. Penjelasan Kode Program Secara Menyeluruh

### 1. *Interface Playable*:



```
1 import java.util.Scanner;
2
3 // Interface untuk game
4 interface Playable {
5     void startGame();
6     void displayBoard();
7     boolean makeMove(int row, int col, Player player);
8     boolean isGameOver();
9 }
```

- a. ***StartGame()***: Metode ini digunakan untuk memulai permainan. Pada implementasinya, permainan akan terus berlanjut selama permainan belum berakhir.
- b. ***DisplayBoard()***: Metode ini bertanggung jawab untuk menampilkan kondisi terkini papan permainan. Isinya adalah tampilan papan permainan di layar.
- c. ***MakeMove(int row, int col, Player player)***: Metode ini memungkinkan pemain membuat langkah pada posisi tertentu di papan permainan. Posisi langkah ditentukan oleh *row* dan *col*, dan pemain yang melakukan langkah adalah *player*.
- d. ***IsGameOver()***: Metode ini memeriksa apakah permainan sudah berakhir. Permainan dianggap berakhir jika ada pemenang atau papan penuh.

## 2. *Abstract class Player:*

```
1  abstract class Player {
2      private String name;
3      private char symbol;
4
5      public Player(String name, char symbol) {
6          this.name = name;
7          this.symbol = symbol;
8      }
9
10     // Getter untuk mendapatkan nama pemain
11     public String getName() {
12         return name;
13     }
14
15     // Getter untuk mendapatkan simbol pemain
16     public char getSymbol() {
17         return symbol;
18     }
19
20     // Setter untuk memperbarui nama pemain
21     public void setName(String name) {
22         this.name = name;
23     }
24
25     // Setter untuk memperbarui simbol pemain
26     public void setSymbol(char symbol) {
27         this.symbol = symbol;
28     }
29 }
```

Pada bagian ini, sebuah kelas abstrak bernama *Player* sedang didefinisikan. Kelas ini memiliki dua variabel *instan privat*, yaitu *name* untuk menyimpan nama pemain dan *symbol* untuk menyimpan simbol pemain. Variabel-variabel ini bersifat *privat*, artinya hanya dapat diakses dan diubah dari dalam kelas *Player* itu sendiri.

### a. **Constructor** (*public Player(String name, char symbol)*)

```
public Player(String name, char symbol) {
    this.name = name;
    this.symbol = symbol;
}
```

Kode diatas adalah konstruktor dari kelas *Player*. Konstruktor ini digunakan untuk membuat objek *Player* baru dengan menginisialisasi nilai dari dua atribut *privat*: *name* (nama pemain) dan *symbol* (simbol pemain) berdasarkan parameter yang diberikan pada saat pembuatan objek. Dengan kata lain, konstruktor ini mengatur nilai awal untuk atribut-atribut tersebut.

- b. **Getter** dan **Setter** (`getName()`, `getSymbol()`, `setName(String name)`, `setSymbol(char symbol)`).

```
// Getter untuk mendapatkan nama pemain
public String getName() {
    return name;
}

// Getter untuk mendapatkan simbol pemain
public char getSymbol() {
    return symbol;
}

// Setter untuk memperbarui nama pemain
public void setName(String name) {
    this.name = name;
}

// Setter untuk memperbarui simbol pemain
public void setSymbol(char symbol) {
    this.symbol = symbol;
}
```

Kode di atas adalah implementasi *getter* dan *setter* dalam kelas *Player* :

- 1) `public String getName()` : Metode ini mengembalikan nilai dari atribut *name* (nama pemain).
- 2) `public char getSymbol()` : Metode ini mengembalikan nilai dari atribut *symbol* (simbol pemain).
- 3) `public void setName(String name)` : Metode ini digunakan untuk memperbarui nilai atribut *name* dengan nilai yang diberikan sebagai parameter.
- 4) `public void setSymbol(char symbol)` : Metode ini digunakan untuk memperbarui nilai atribut *symbol* dengan nilai yang diberikan sebagai parameter.

Secara keseluruhan, metode-metode ini menyediakan akses kontrol yang aman untuk mendapatkan dan mengubah nilai atribut *name* dan *symbol* pada objek *Player*.

### 3. Kelas TicTacToe:

- a. **Konstruktor** (`public TicTacToe(String player1Name, String player2Name)`)

```

public TicTacToe(String player1Name, String player2Name) {
    board = new char[3][3];
    player1 = new PlayerImpl(name: player1Name, symbol: 'X');
    player2 = new PlayerImpl(name: player2Name, symbol: 'O');
    currentPlayer = player1;
    initializeBoard();
}

```

Kode diatas digunakan untuk menginisialisasi objek TicTacToe dengan membuat papan permainan (*board*), dua pemain (*player1* dan *player2*), dan pemain yang sedang giliran (*currentPlayer*).

b. **Metode** initializeBoard()

```

private void initializeBoard() {
    // Inisialisasi papan permainan
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            board[i][j] = '-';
        }
    }
}

```

digunakan untuk mengisi papan permainan dengan karakter '-' yang menunjukkan bahwa seluruh papan masih kosong.

c. **Metode** StartGame()

```

public void startGame() {
    Scanner scanner = new Scanner(source: System.in);
    while (!isGameOver()) {
        displayBoard();
        System.out.println("Giliran " + currentPlayer.getName() + " (" + currentPlayer.getSymbol() + ")");
        System.out.print(s: "Masukkan baris (0-2): ");
        int row = scanner.nextInt();
        System.out.print(s: "Masukkan kolom (0-2): ");
        int col = scanner.nextInt();

        if (makeMove(row, col, player: currentPlayer)) {
            if (checkWin()) {
                displayBoard();
                System.out.println(currentPlayer.getName() + " menang!");
                break;
            } else if (isBoardFull()) {
                displayBoard();
                System.out.println(s: "Permainan seri!");
                break;
            } else {
                currentPlayer = (currentPlayer == player1 ? player2 : player1);
            }
        } else {
            System.out.println(s: "Langkah tidak valid. Coba lagi.");
        }
    }
    scanner.close();
}

```

Kode di atas adalah implementasi metode *startGame* dalam kelas TicTacToe. Berikut adalah penjelasan singkat:

- 1) `Scanner scanner = new Scanner(System.in);` : Membuat objek *Scanner* untuk menerima *input* dari pengguna.

- 2) `while (!isGameOver())` {: Memulai *loop* permainan selama permainan belum berakhir.
- 3) `displayBoard()` {: Menampilkan papan permainan ke layar.
- 4) `System.out.println("Giliran"+currentPlayer.getName()  
)+ " (" + currentPlayer.getSymbol() + ")");` : Menampilkan informasi tentang pemain yang sedang bermain (nama dan simbol).
- 5) `System.out.print("Masukkan baris (0-2): ");` dan `System.out.print("Masukkan kolom (0-2): ");` : Meminta *input* dari pengguna untuk baris dan kolom tempat mereka ingin menempatkan simbol.
- 6) `int row = scanner.nextInt();` dan `int col = scanner.nextInt();` : Membaca *input* baris dan kolom dari pengguna.
- 7) `if (makeMove(row, col, currentPlayer)){` : Memeriksa apakah langkah yang dimasukkan oleh pemain saat ini *valid*. Jika *valid*, lanjut ke langkah selanjutnya.
- 8) Dalam blok `if (checkWin())` {}, memeriksa apakah pemain saat ini menang. Jika ya, menampilkan papan, mengumumkan pemenang, dan mengakhiri permainan.
- 9) Dalam blok `else if (isBoardFull())` {}, memeriksa apakah papan permainan penuh (seri). Jika ya, menampilkan papan, mengumumkan permainan seri, dan mengakhiri permainan.
- 10) Dalam blok `else` {}, beralih ke pemain berikutnya.
- 11) `scanner.close();` : Menutup objek *Scanner* setelah permainan selesai.

Secara keseluruhan, metode *startGame* mengelola alur permainan Tic-Tac-Toe, memungkinkan pemain untuk saling bergantian melakukan langkah, dan mengumumkan hasil permainan saat permainan berakhir.

d. **Metode** DisplayBoard()

```
public void displayBoard() {  
    // Menampilkan papan permainan ke layar  
    System.out.println(x: "Papan Saat Ini:");  
    for (int i = 0; i < 3; i++) {  
        for (int j = 0; j < 3; j++) {  
            System.out.print(board[i][j] + " ");  
        }  
        System.out.println();  
    }  
}
```

Kode di atas adalah implementasi metode *displayBoard* dalam kelas TicTacToe. Berikut adalah penjelasan singkat:

- 1) `System.out.println("Papan Saat Ini:");` : Menampilkan pesan "Papan Saat Ini:" sebagai *header* papan permainan.
- 2) Menggunakan *nested loop* untuk iterasi melalui setiap baris dan kolom papan permainan.
- 3) `System.out.print(board[i][j] + " ");` : Menampilkan nilai dari setiap sel di papan permainan, diikuti oleh spasi.
- 4) `System.out.println();` : Pindah ke baris berikutnya setelah menampilkan semua kolom dalam satu baris.

Secara keseluruhan, metode *displayBoard* digunakan untuk mencetak tampilan saat ini dari papan permainan Tic-Tac-Toe ke layar, memungkinkan pemain dan pengguna melihat kondisi permainan dengan representasi papan yang jelas.

e. **Metode** makeMove()

```
public boolean makeMove(int row, int col, Player player) {  
    // Melakukan langkah pemain pada posisi tertentu di papan permainan  
    if (row >= 0 && row < 3 && col >= 0 && col < 3 && board[row][col] == '-') {  
        board[row][col] = player.getSymbol();  
        return true;  
    } else {  
        return false;  
    }  
}
```

Kode diatas memungkinkan pemain untuk membuat langkahnya pada posisi yang diinginkan, dengan pengecekan kevalidan langkah.

f. **Metode** `isGameOver()`

```
public boolean isGameOver() {
    // Periksa apakah permainan telah berakhir
    return checkWin() || isBoardFull();
}
```

Kode diatas merupakan sebuah kode untuk memeriksa apakah permainan sudah berakhir.

g. **Metode** `CheckWin()` dan `isBoardFull()`

```
private boolean checkWin() {
    // Implementasikan logika pemeriksaan kemenangan di sini
    for (int i = 0; i < 3; i++) {
        // Check baris
        if (board[i][0] == board[i][1] && board[i][1] == board[i][2] && board[i][0] != '-') {
            return true;
        }
        // Check kolom
        if (board[0][i] == board[1][i] && board[1][i] == board[2][i] && board[0][i] != '-') {
            return true;
        }
    }
    // Check diagonal
    if ((board[0][0] == board[1][1] && board[1][1] == board[2][2] && board[0][0] != '-') ||
        (board[0][2] == board[1][1] && board[1][1] == board[2][0] && board[0][2] != '-')) {
        return true;
    }
    return false;
}

private boolean isBoardFull() {
    // Periksa apakah papan permainan sudah penuh
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[i][j] == '-') {
                return false;
            }
        }
    }
    return true;
}
```

Kode diatas digunakan untuk menentukan apakah ada pemain yang menang atau papan penuh.

4. **Kelas** *PlayerImpl*:

```
class PlayerImpl extends Player {
    public PlayerImpl(String name, char symbol) {
        super(name, symbol);
    }
}
```

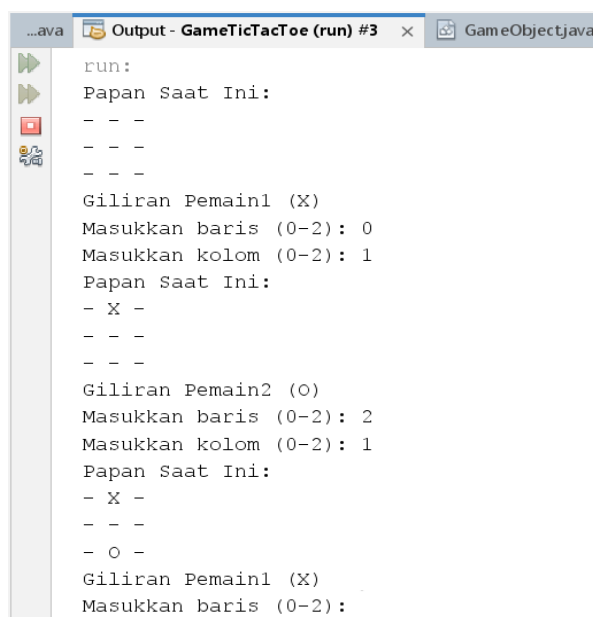
- Mengimplementasikan kelas abstrak *Player*.
- Mewarisi konstruktor dan fungsionalitas dasar dari *Player*.
- Dapat memiliki metode atau perilaku tambahan khusus untuk seorang pemain jika diperlukan.



- d. Dengan kata lain, kelas ini mengimplementasikan kelas abstrak *Player*. Dalam hal ini, ia hanya berfungsi sebagai implementasi konkret dari kelas *Player*.

#### 5. Cara Kerjanya:

- Program menginisialisasi permainan Tic-Tac-Toe dengan dua pemain.
- Pemain bergantian memasukkan langkah mereka (baris dan kolom) melalui *input* konsol.



```
run:
Papan Saat Ini:
- - -
- - -
- - -
Giliran Pemain1 (X)
Masukkan baris (0-2): 0
Masukkan kolom (0-2): 1
Papan Saat Ini:
- X -
- - -
- - -
Giliran Pemain2 (O)
Masukkan baris (0-2): 2
Masukkan kolom (0-2): 1
Papan Saat Ini:
- X -
- - -
- O -
Giliran Pemain1 (X)
Masukkan baris (0-2):
```

- Permainan memeriksa apakah ada pemenang atau seri setelah setiap langkah.
- Permainan terus berlanjut hingga ada pemenang atau papan penuh.

```
Giliran Pemain2 (O)
Masukkan baris (0-2): 1
Masukkan kolom (0-2): 2
Papan Saat Ini:
X X O
- X O
X O O
Pemain2 menang!
BUILD SUCCESSFUL (total time: 4 minutes 9 seconds)
```

- Hasil akhir (pemenang atau seri) ditampilkan di konsol.

Implementasi ini menggunakan *input/output* konsol dasar dan cocok untuk permainan Tic-Tac-Toe sederhana berbasis baris perintah.