PEMROGRAMAN II

Dosen Pengampu:

Irham Maulani Abdul Gani, S.Kom., M.Kom.

TUGAS CLI TEMPLATE

Oleh:

Achmad Reihan Alfaiz NIM. 2410817210019

PROGRAM STUDI TEKNOLOGI INFORMASI

FAKULTAS TEKNIK

UNIVERSITAS LAMBUNG MANGKURAT

NOVEMBER 2025

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

SOAL

Menggunakan prinsip Abstract class, Interface dan Composition. Buatkan CLI template menggunakan Bahasa Java. Tidak boleh menggunakan Library apapun di luar base Java.

Specification:

1. CLI harus memiliki halaman yang bisa dipilih di menu utama.
2. CLI harus memiliki dua fitur memilih menu, scan input user dan menampilkannya.
3. Menggunakan prinsip OOP penyembunyian, main static tidak boleh menjadi GOD class dan harus DUMB.
4. Buat report berisikan alasan kalian menggunakan teknik kalian dan insight yang kalian dapatkan.

A. Source Code

Components

Tabel 1. Source Code Interface Displayable

```java
package components;

public interface Displayable {
    void display();
}
```

Tabel 2. Source Code Abstract Class Menu

```java
package components;

import controllers.UserActionHandler;

public abstract class Menu implements Displayable,
UserActionHandler {
    protected String title;
    protected Menu parentMenu;
    protected HeaderMenu header;
    protected BodyMenu body;
    protected FooterMenu footer;

    public Menu(String title) {
        this.title = title;
        this.header = new HeaderMenu(title);
        this.footer = new FooterMenu();
    }

    public Menu(String title, Menu parentMenu) {
        this(title);
        this.parentMenu = parentMenu;
    }

    public final void display() {
        clearConsole();
        header.display();
        body.display();
        footer.display();
    }

    public Menu getParentMenu() {
        return parentMenu;
    }

    public BodyMenu getBody() {
        return body;
    }

    public abstract void run();

    public static void clearConsole() {
        for (int i = 0; i < 50; ++i) { System.out.println(); }
    }

    public static void waitForEnter(java.util.Scanner scanner) {
        System.out.println("\nPress Enter to continue...");
        scanner.nextLine();
    }
}
```

Tabel 3. Source Code Class HeaderMenu

```java
package components;

public class HeaderMenu implements Displayable{
    private final String title;

    public HeaderMenu(String title) {
        this.title = title;
    }

    @Override
    public void display() {
        System.out.println("\n" + "=".repeat(50));
        System.out.println(title.toUpperCase());
        System.out.println("=".repeat(50));
    }
}
```

Tabel 4. Source Code Class BodyMenu

```java
package components;

import java.util.List;
import java.util.ArrayList;

import static utils.TerminalFormatter.*;

public class BodyMenu implements Displayable{
    private final List<String> options;

    public BodyMenu() {
        this.options = new ArrayList<>();
    }

    public void addOption(int number, String label) {
        options.add(number + ". " + label);
    }

    public void clearOptions() {
        options.clear();
    }

    @Override
    public void display() {
        addNewLine(5);
        for (String option : options) {
            System.out.println(option);
        }
        addNewLine(5);
    }
}
```

Tabel 5. Source Code Class FooterMenu

```java
package components;

import static utils.TerminalFormatter.*;

public class FooterMenu implements Displayable {
    private final String prompt;

    public FooterMenu() {
        this.prompt = "Choose an option: ";
    }

    public FooterMenu(String prompt) {
        this.prompt = prompt;
    }

    @Override
    public void display() {
        System.out.println("=".repeat(50));
        System.out.println("© 2025 Achmad Reihan Alfaiz. All rights reserved.");
        System.out.println("=".repeat(50));
        addNewLine(1);
        System.out.print(prompt);
    }
}
```

Tabel 6. Source Code Class MainMenu

```java
package components;

import controllers.MainMenuController;
import java.util.Scanner;

public class MainMenu extends Menu {
    private final Scanner scanner;
    private final MainMenuController controller;

    public MainMenu() {
        super("Main Menu");
        this.body = new BodyMenu();
        this.scanner = new Scanner(System.in);
        this.controller = new MainMenuController(this);
    }

    @Override
    public void run() {
        while (true) {
            display();
            handleUserInput(scanner.nextLine());
        }
    }

    @Override
    public boolean handleUserInput(String input) {
        return controller.handleUserInput(input);
    }
}
```

Tabel 7. Source Code Class PlayChessMenu

```java
package components;

import controllers.PlayChessMenuController;
import java.util.Scanner;

public class PlayChessMenu extends Menu {
    private final Scanner scanner;
    private final PlayChessMenuController controller;

    public PlayChessMenu(Menu parentMenu) {
        super("Play Chess", parentMenu);
        this.body = new BodyMenu();
        this.scanner = new Scanner(System.in);
        this.controller = new PlayChessMenuController(this);
    }

    @Override
    public void run() {
        boolean running = true;
        while (running) {
            display();
            String input = scanner.nextLine();
            running = handleUserInput(input);
            if (running) {
                Menu.waitForEnter(scanner);
            }
        }
    }

    @Override
    public boolean handleUserInput(String input) {
        return controller.handleUserInput(input);
    }
}
```

Tabel 8. Source Code Class ChessPiecesDetailMenu

```java
package components;

import controllers.ChessPiecesDetailMenuController;
import java.util.Scanner;

public class ChessPiecesDetailMenu extends Menu {
    private final Scanner scanner;
    private final ChessPiecesDetailMenuController controller;

    public ChessPiecesDetailMenu(Menu parentMenu) {
        super("Chess Pieces Detail Menu", parentMenu);
        this.body = new BodyMenu();
        this.scanner = new Scanner(System.in);
        this.controller = new ChessPiecesDetailMenuController(this);
    }

    @Override
    public void run() {
        boolean running = true;
        while (running) {
            display();
            String input = scanner.nextLine();
            running = handleUserInput(input);
            if (running) {
                Menu.waitForEnter(scanner);
            }
        }
    }

    @Override
    public boolean handleUserInput(String input) {
        return controller.handleUserInput(input);
    }
}
```

Controllers

Tabel 9. Source Code Interface UserActionHandler

```java
package controllers;

public interface UserActionHandler {
    boolean handleUserInput(String input);
}
```

Tabel 10. Source Code Class MainMenuController

```java
package controllers;

import components.MainMenu;
import components.PlayChessMenu;
import components.ChessPiecesDetailMenu;

public class MainMenuController implements UserActionHandler {
    private final MainMenu menu;

    public MainMenuController(MainMenu menu) {
        this.menu = menu;
        initializeOptions();
    }

    private void initializeOptions() {
        menu.getBody().addOption(1, "Play Chess");
        menu.getBody().addOption(2, "Chess Pieces Detail");
        menu.getBody().addOption(0, "Exit");
    }

    @Override
    public boolean handleUserInput(String input) {
        switch (input) {
            case "1":
                new PlayChessMenu(menu).run();
                break;
            case "2":
                new ChessPiecesDetailMenu(menu).run();
                break;
            case "0":
                System.exit(0);
                break;
            default:
                System.out.println("Invalid option! Please try again.");

        }
        return true;
    }
}
```

Tabel 11. Source Code Class PlayChessMenuController

```java
package controllers;

import components.PlayChessMenu;

public class PlayChessMenuController implements UserActionHandler {
    private final PlayChessMenu menu;

    public PlayChessMenuController(PlayChessMenu menu) {
        this.menu = menu;
        initializeOptions();
    }

    private void initializeOptions() {
        menu.getBody().addOption(1, "New Game");
        menu.getBody().addOption(2, "Load Game");
        menu.getBody().addOption(0, "Back to Main Menu");
    }

    @Override
    public boolean handleUserInput(String input) {
        switch (input) {
            case "1":
                System.out.println("Starting a new game...");
                break;
            case "2":
                System.out.println("Loading a saved game...");
                break;
            case "0":
                return false;
            default:
                System.out.println("Invalid option. Please try again.");
        }
        return true;
    }
}
```

Tabel 12. Source Code Class ChessPiecesDetailMenuController

```
package controllers;

import components.ChessPiecesDetailMenu;
import chess.pieces.*;

public class ChessPiecesDetailMenuController implements UserActionHandler {
    private final ChessPiecesDetailMenu menu;

    public ChessPiecesDetailMenuController(ChessPiecesDetailMenu menu) {
        this.menu = menu;
        initializeOptions();
    }

    private void initializeOptions() {
        menu.getBody().addOption(1, "Pawn");
        menu.getBody().addOption(2, "Knight");
        menu.getBody().addOption(3, "Bishop");
        menu.getBody().addOption(4, "Rook");
        menu.getBody().addOption(5, "Queen");
        menu.getBody().addOption(6, "King");
        menu.getBody().addOption(0, "Back to Previous Menu");
    }

    @Override
    public boolean handleUserInput(String input) {
        switch (input) {
            case "1":
                Piece pawn = new Pawn();
                pawn.displayDetails();
                break;
            case "2":
                Piece knight = new Knight();
                knight.displayDetails();
                break;
            case "3":
                Piece bishop = new Bishop();
                bishop.displayDetails();
                break;
            case "4":
                Piece rook = new Rook();
                rook.displayDetails();
                break;
            case "5":
                Piece queen = new Queen();
                queen.displayDetails();
                break;
            case "6":
                Piece king = new King();
                king.displayDetails();
                break;
            case "0":
                return false;
            default:
                System.out.println("Invalid option!
Please try again.");
        }
        return true;
    }
}
```

Utils

Tabel 13. Source Code Class TerminalFormatter

```
package utils;

public final class TerminalFormatter {
    private TerminalFormatter(){};

    public static void addNewLine() {
System.out.println(); }

    public static void addNewLine(int count)
{System.out.print("\n".repeat(Math.max(0, count))); }
}
```

Chess/Pieces

Tabel 14. Source Code Abstract Class Piece

```
package chess.pieces;

public abstract class Piece {
    public abstract void displayDetails();
}
```

Tabel 15. Source Code Class Pawn

```
package chess.pieces;

import static utils.TerminalFormatter.*;

```

```java
public class Pawn extends Piece {
    @Override
    public void displayDetails() {
        addNewLine(50);
        System.out.println("""
                   __
                  /  \\
                  \\__/
                 /____\\
                  |  |
                  |__|
                 (====)
                 }===={
                (______)

                Pawn:
                - Moves forward one square
                - Captures diagonally
                - Can promote to another piece upon reaching the opposite end of the board""");
    }
}
```

Tabel 16. Source Code Class Knight

```java
package chess.pieces;

import static utils.TerminalFormatter.*;

public class Knight extends Piece {
    @Override
    public void displayDetails() {
        addNewLine(50);
        System.out.println("""
                 (\\=,
                //  .\\
               (( \\_  \\
                ))   `\\_)
               (/     \\
                | _.-'|
                 )___(
                (=====)
                }====={
               (_______)

                Knight:
                - Moves in an 'L' shape: two squares in one direction and then one square perpendicular
                - Can jump over other pieces
                - Valuable for controlling the center of the board""");
    }
}
```

Tabel 17. Source Code Class Bishop

```java
package chess.pieces;

import static utils.TerminalFormatter.*;

public class Bishop extends Piece {
    @Override
    public void displayDetails() {
        addNewLine(50);
        System.out.println("""
                    ()
                    /\\
                   //\\\\
                  (    )
                   )__(
                  /____\\
                   |  |
                   |  |
                  /____\\
                 (======)
                 }======{
                (________)

                 Bishop:
                 - Moves diagonally any number of squares
                 - Cannot jump over other pieces
                 - Strong on long diagonals and in open positions""");
    }
}
```

Tabel 18. Source Code Class Rook

```java
package chess.pieces;

import static utils.TerminalFormatter.*;

public class Rook extends Piece {
    @Override
    public void displayDetails() {
        addNewLine(50);
        System.out.println("""
                 |'-'-'|
                 |_____|
                  |===|
                  |   |
                  |   |
                  )___(
                 (=====)
                 }====={
```

```
                (_______)

               Rook:
               - Moves horizontally or vertically any number of squares
               - Cannot jump over other pieces
               - Essential for controlling open files and ranks""");
    }
}
```

Tabel 19. Source Code Class Queen

```
package chess.pieces;

import static utils.TerminalFormatter.*;

public class Queen extends Piece {
    @Override
    public void displayDetails() {
        addNewLine(50);
        System.out.println("""
                   ()
                .-:--:-.
                 \\____/
                 {====}
                  )__(
                 /____\\
                  |  |
                  |  |
                  |  |
                  |  |
                 /____\\
                (======)
                }======{
               (________)

               Queen:
               - Moves horizontally, vertically, or diagonally any number of squares
               - Cannot jump over other pieces
               - The most powerful piece on the board""");
    }
}
```

Tabel 20. Source Code Class King

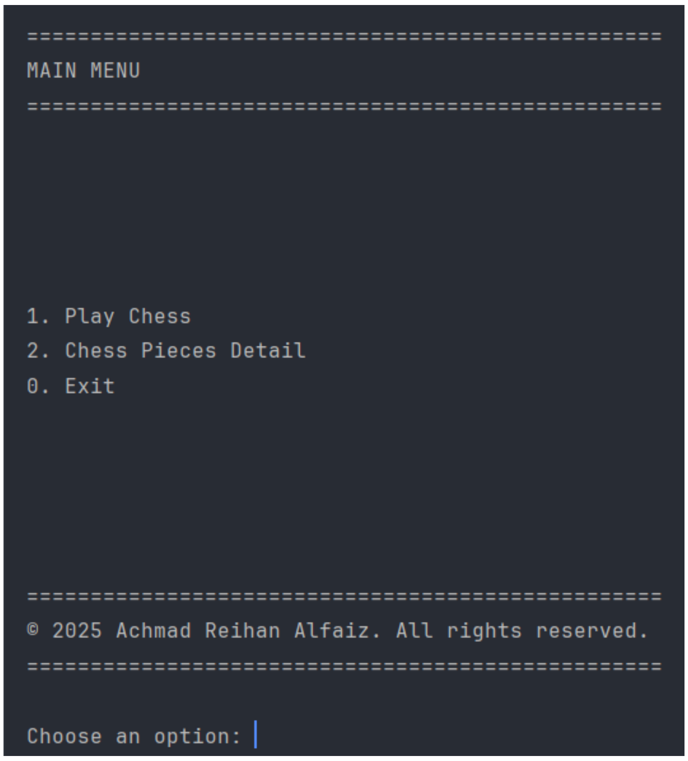
```
package chess.pieces;

import static utils.TerminalFormatter.*;

public class King extends Piece {
    @Override
    public void displayDetails() {
        addNewLine(50);
        System.out.println("""
                  _:_
                 '-.-'
                __.'.__
               |_______|
                \\=====/
                 )___(
                /_____\\
                 |   |
                 |   |
                 |   |
                 |   |
                 |   |
                /_____\\
               (=======)
               }======={
              (_________)

              King:
              - Moves one square in any direction
              - Special move: Castling with a rook
              - Crucial piece; checkmate ends the game""");
    }
}
```

Main

Tabel 21. Source Code Class CLI

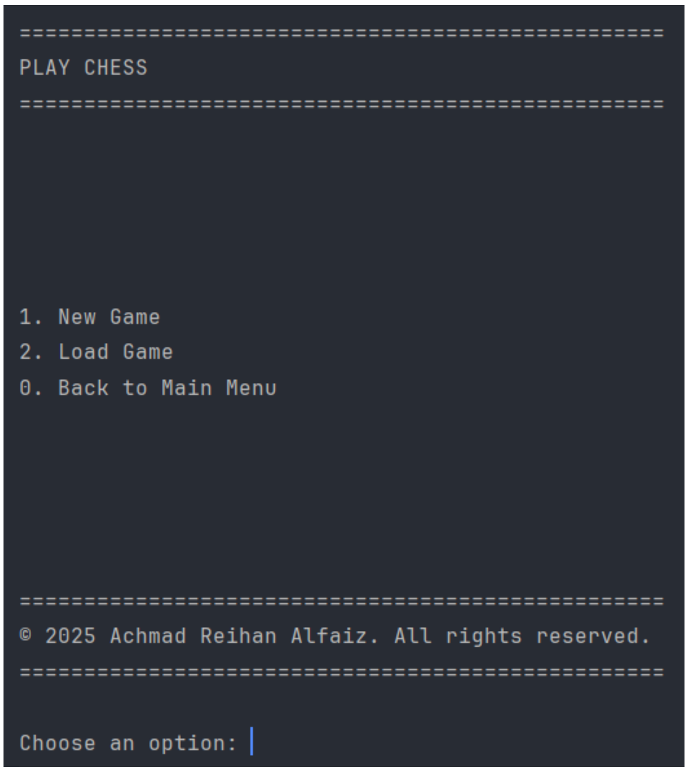
```
import components.MainMenu;

public class CLI {
    public static void main(String[] args) {
        new MainMenu().run();
    }
}
```

B. Output Program



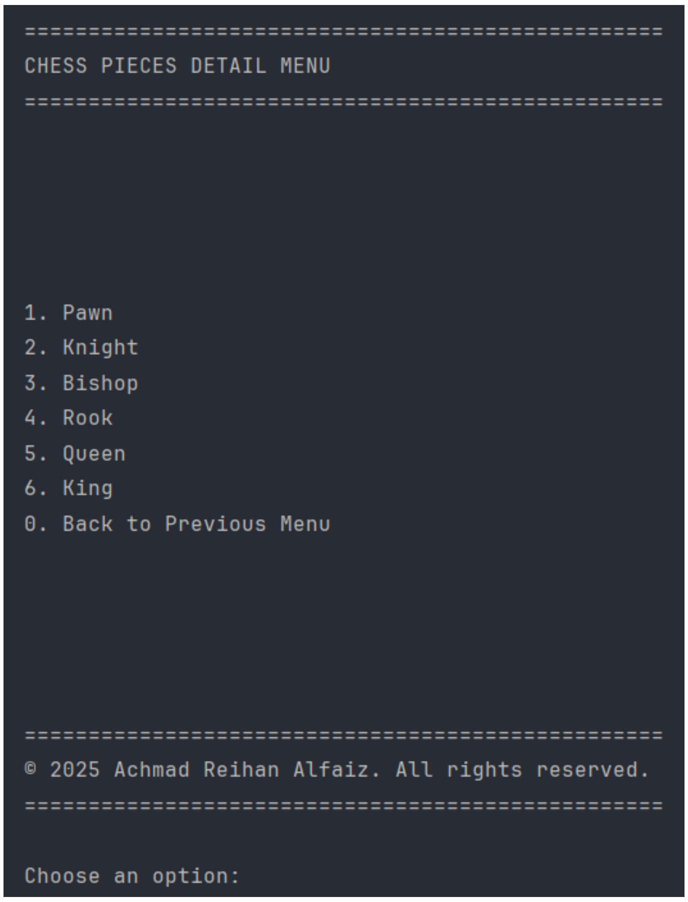
```
==================================================
MAIN MENU
==================================================




1. Play Chess
2. Chess Pieces Detail
0. Exit




==================================================
© 2025 Achmad Reihan Alfaiz. All rights reserved.
==================================================

Choose an option:
```

Gambar 1. Screenshot Tampilan Main Menu



```
==================================================
PLAY CHESS
==================================================




1. New Game
2. Load Game
0. Back to Main Menu




==================================================
© 2025 Achmad Reihan Alfaiz. All rights reserved.
==================================================

Choose an option:
```

Gambar 2. Screenshot Tampilan Play Chess Menu



```
==================================================
CHESS PIECES DETAIL MENU
==================================================





1. Pawn
2. Knight
3. Bishop
4. Rook
5. Queen
6. King
0. Back to Previous Menu





==================================================
© 2025 Achmad Reihan Alfaiz. All rights reserved.
==================================================

Choose an option:
```

Gambar 3. Screenshot Tampilan Chess Pieces Detail Menu

C. Teknik dan Insight

Teknik yang Digunakan

Dalam membangun aplikasi CLI ini, fokus utama adalah menerapkan prinsip-prinsip OOP untuk menghasilkan kode yang bersih dan terstruktur.

- Pengorganisasian Berdasarkan Fungsionalitas

  Saya membagi kelas-kelas ke dalam beberapa package (components, controllers, utils, chess.pieces) agar setiap bagian memiliki tanggung jawab yang jelas. Components untuk tampilan, controllers untuk logika, utils untuk alat bantu, dan chess.pieces untuk data.

- Abstract Class sebagai Template

  Abstract class Menu dan Piece saya gunakan sebagai template dasar. Hal ini memastikan semua menu dan bidak catur turunan memiliki struktur dan perilaku yang konsisten tanpa perlu duplikasi kode.

- Interface sebagai Kontrak
  Saya menggunakan interface seperti Displayable dan UserActionHandler untuk mendefinisikan "kontrak" atau kemampuan wajib. Hal ini memaksa setiap komponen UI untuk bisa ditampilkan dan setiap controller untuk bisa menangani input, sehingga membuat sistem lebih dapat diprediksi.
- Composition untuk Fleksibilitas
  Selain inheritance, composition juga diterapkan dan menjadi highlight utama. Contohnya, kelas Menu memiliki Header, Body, dan Footer. Hal ini membuat komponen-komponen tersebut sangat modular dan mudah digunakan kembali di menu mana pun.
- Pemisahan Logika
  Kelas tampilan (MainMenu) saya buat "bodoh", ia hanya tahu cara menampilkan diri. Semua logika, apa yang terjadi saat user memilih menu, sepenuhnya ditangani oleh kelas Controller.
- Main Class Sebagai Entry Point
  Kelas CLI dengan method main hanya bertindak sebagai titik masuk (entry point) untuk menjalankan program, sesuai dengan best practices.

Insight yang Didapatkan

Dari proses ini, saya mendapatkan beberapa pemahaman penting, yaitu:

- Struktur adalah Kunci
  Pengorganisasian kode ke dalam package yang jelas membuat navigasi dan pencarian bug di kemudian hari menjadi jauh lebih mudah. Saya tahu persis harus ke mana jika ingin memperbaiki logika atau tampilan.
- Composition Lebih Unggul untuk Reusability
  Menggunakan composition benar-benar membuat komponen UI saya bisa dipakai ulang di mana saja. Jika butuh menu baru, saya tinggal merakitnya dari komponen yang sudah ada.
- Kode Jadi Jauh Lebih Mudah Dikembangkan
  Karena logika dan tampilan terpisah, menambahkan fitur baru (misalnya menu "Opening Books") tidak akan merusak bagian lain. Saya hanya perlu membuat pasangan kelas Menu dan Controller yang baru.

- Keterbacaan Kode Meningkat Drastis

  Pada akhirnya, setiap kelas kini memiliki satu tanggung jawab yang spesifik. Hal ini membuat alur program sangat mudah diikuti, baik untuk saya maupun orang lain yang mungkin akan membaca kode ini.

TAUTAN GIT

https://github.com/ach-reihan/praktikum-pemrograman-ii/tree/main/cli-template