

Nama : Rika Fauliana Rahmi

NIM : 2410817120017

## PEMROGRAMAN II

---

### **Link Github untuk Code:**

<https://github.com/rikafaulianarahmi/PEMROII/tree/main/CLI>

## **1. Prinsip Desain yang Diterapkan**

### **1.1 Penggunaan Interface**

Interface yang Diimplementasikan:

- InputHandler - Menangani semua operasi input pengguna
- OutputHandler - Menangani semua operasi output/tampilan
- Page - Kontrak untuk semua implementasi halaman

Alasan: Interface menyediakan abstraksi dan fleksibilitas. Dengan menggunakan interface, dapat dengan mudah mengganti implementasi (misalnya, mengganti ConsoleInputHandler dengan FileInputHandler) tanpa mengubah logika inti.

Insight: Memisahkan urusan input/output melalui interface membuat kode dapat diuji dan mudah dipelihara. Kita dapat membuat implementasi mock untuk unit testing tanpa menyentuh console I/O.

### **1.2 Penggunaan Abstract Class**

Abstract Class: BasePage

Alasan: BasePage menyediakan fungsionalitas umum untuk semua halaman (tampilan header, manajemen flag exit) sambil memungkinkan subclass mengimplementasikan perilaku display() dan handleInput() mereka yang spesifik. Ini menghilangkan duplikasi kode di berbagai jenis halaman.

Insight: Abstract class sangat cocok untuk Template Method pattern. Perilaku umum terpusat di base class sementara implementasi spesifik tetap ada di concrete subclass. Ini mengurangi redundansi kode dan memastikan konsistensi.

### **1.3 Penggunaan Composition**

#### **Class yang Menggunakan Composition:**

- ApplicationController - Menggabungkan InputHandler, OutputHandler, PageNavigator, dan DataStore
- MainMenuPage - Menggabungkan PageNavigator
- UserInputPage & DisplayInfoPage - Menggabungkan DataStore
- PageNavigator - Menggabungkan HashMap dari Pages

Alasan: Composition lebih baik daripada inheritance karena mempromosikan loose coupling dan fleksibilitas. Setiap komponen memiliki tanggung jawab tunggal dan dapat dimodifikasi secara independen. ApplicationController mengatur semua komponen tanpa terikat erat dengan implementasinya.

Insight: Composition memungkinkan kita membangun sistem kompleks dari komponen sederhana dan dapat digunakan kembali. Setiap class memiliki batasan dan tanggung jawab yang jelas, membuat sistem lebih mudah dipahami, diuji, dan dipelihara.

## **2. Information Hiding & Encapsulation**

### **2.1 Main Class yang DUMB**

Class Main hanya berisi entry point (method main) tanpa logika bisnis. Class ini hanya membuat ApplicationController dan memanggil run().

Manfaat: Perubahan pada logika aplikasi tidak pernah memerlukan modifikasi pada class Main. Pemisahan ini memastikan entry point tetap stabil.

### **2.2 Protected Members**

BasePage menggunakan field protected (inputHandler, outputHandler, exitFlag) sehingga subclass dapat mengaksesnya sambil menyembunyikan detail implementasi dari class eksternal.

## **2.3 Detail Implementasi Private**

- DataStore menggunakan ArrayList private secara internal
- PageNavigator menggunakan HashMap private secara internal
- Semua handler mengenkapsulasi implementasi Scanner/System.out mereka

## **3. Kesimpulan**

Desain ini mendemonstrasikan bagaimana prinsip OOP menciptakan aplikasi yang mudah dipelihara dan dapat diskalakan. Pemisahan urusan melalui interface, penggunaan kembali kode melalui abstract class, dan fleksibilitas melalui composition menghasilkan sistem yang robust di mana komponen dapat dimodifikasi, diuji, dan diperluas secara independen.

Class Main yang "dumb" memastikan entry point tetap stabil, sementara ApplicationController mengatur semua komponen tanpa mengetahui implementasi internal mereka. Arsitektur ini dapat dengan mudah diskalakan dari CLI sederhana menjadi aplikasi kompleks dengan menambahkan komponen baru tanpa mengganggu yang sudah ada.

## **4. Penjelasan Teknik yang Digunakan**

### **4.1 Mengapa Interface untuk Input/Output?**

Dalam dunia nyata, aplikasi mungkin perlu membaca input dari berbagai sumber (console, file, network). Dengan interface, kita dapat mengganti sumber input tanpa mengubah logika aplikasi. Misalnya:

- Testing: Gunakan MockInputHandler yang return nilai tetap
- Production: Gunakan ConsoleInputHandler
- File Processing: Gunakan FileInputHandler

### **4.2 Mengapa Abstract Class untuk BasePage?**

Semua halaman memiliki perilaku umum:

- Menampilkan header
- Mengelola status exit

- Memiliki InputHandler dan OutputHandler

Daripada copy-paste kode ini ke setiap halaman, kita taruh di BasePage. Setiap halaman konkret hanya perlu fokus pada logika spesifiknya.

### **4.3 Mengapa Composition di ApplicationController?**

AppController tidak "adalah" InputHandler atau DataStore. AppController "memiliki" komponen-komponen ini. Ini lebih masuk akal secara konseptual dan memberikan fleksibilitas:

- Ganti DataStore dengan DatabaseStore tanpa mengubah ApplicationController
- Tambah komponen baru tanpa inheritance chain yang kompleks
- Setiap komponen dapat di-reuse di tempat lain

### **4.4 Keuntungan Main Class yang DUMB**

Main class yang sederhana berarti:

- Entry point aplikasi stabil, jarang berubah
- Testing lebih mudah (tidak perlu test Main class)
- Logika bisnis terorganisir di class yang tepat
- Prinsip separation of concerns terjaga

## **5. Pelajaran yang Didapat**

### **5.1 Abstraksi adalah Kunci**

Dengan mengabstraksi input/output ke interface, kode menjadi lebih fleksibel dan testable. Ini adalah investasi yang terbayar saat aplikasi berkembang.

### **5.2 Composition > Inheritance**

Composition lebih fleksibel daripada inheritance. Mudah menambah/mengganti komponen tanpa hierarchy class yang rumit.

### **5.3 Setiap Class Punya Tujuan**

Ketika setiap class punya satu tanggung jawab yang jelas, kode lebih mudah dipahami dan di-maintain. Developer baru dapat langsung tahu ke mana harus pergi untuk fitur tertentu.

#### **5.4 Desain untuk Perubahan**

Software pasti berubah. Dengan desain yang baik (interface, composition, encapsulation), perubahan dapat dilakukan dengan minimal impact ke kode yang sudah ada.