Nama: Noor Muhammad Akmal Sulaiman

NIM : 2410817210007

Laporan CLI

Alasan

Sebelum saya menjelaskan alasan saya, izinkan saya memohon maaf karena program CLI saya sangat sederhana. Saya ditimpa (*overridden*) oleh banyak sekali tugas secara bersamaan dan saya terlalu lama mengerjakan tugas lain seperti laporan UTS Jaringan Komputer dan Komunikasi Data. Alhasil saya hanya mempunyai waktu sedikit untuk mengerjakan tugas ini sebelum pengumpulan. Namun, saya tetap mengusahakannya sebisa mungkin sesuai dengan perintah tugas, yaitu penggunaan prinsip *abstract class*, *interface*, dan *composition* untuk membuat program CLI yang bisa menerima input dari pengguna dan menampilkannya kembali.

Jadi, terdapat beberapa teknik pemrograman yang saya gunakan dalam program ini, dan salah satunya adalah konsep MVC (*model*, *view*, *controller*). Saya diberi tahu akan hal ini oleh salah seorang teman saya. Bapak Achmad Mujaddid Islami juga menyebutkan konsep MVC pada kelas minggu kemarin. *Model* adalah *class* yang fungsinya menyimpan data. *View* adalah *class* yang fungsinya menampilkan data dan menangkap input. *Controller* adalah *class* yang mengatur perpindahan data.

Konsep MVC ini merupakan penerapan dari *separation of concerns* (SoC). Alasan saya menggunakannya adalah karena menurut saya konsepnya memang sangat intuitif untuk hal seperti ini. Saya mudah memodelkan apa saja yang harus dilakukan *Class* `Fridge` itu adalah *model*, dan kegunaannya memang menyimpan data berupa `FoodItem` yang berada di dalam kulkas. *View* adalah *abstract class* `Page`, *interface* `Displayable`, dan segala *class* turunannya. Kegunaannya adalah mengurus segala hal yang berkenaan dengan apa yang ditampilkan kepada pengguna. *Controller* adalah `App` dan `Main` yang keduanya sama-sama sederhana dibandingkan *class* pada bagian *view*.

Saya menerapkan konsep komposisi (*composition*) pada *class* `App`, yaitu dengan komponen berupa `Fridge fridge` dan `Displayable currentPage`. Di dalam *class* `Fridge`, juga terdapat `List<FoodItem> items`, yang sepertinya juga bisa dikatakan sebagai

penerapan *composition*. Pada *class* `App`, `Fridge fridge` dan `Displayable currentPage` merupakan komponen karena relasinya dengan *class* `App` adalah *has-a*. `App has-a fridge`. `App has-a currentPage`. Bukan seperti pewarisan yang merupakan relasi *is-a*. `App` bukan merupakan `fridge` dan bukan merupakan `currentPage`. Jadi alasan digunakannya prinsip komposisi bukanlah karena diwajibkan (tetapi itu sangat berpengaruh), melainkan karena hubungan paling logis pada kasus ini hanyalah itu. Sebuah `Fridge` juga mempunyai (*has-a*) `List<FoodItem> items`. Tidak mungkin sebuah `Fridge` merupakan (*is-a*) `List<FoodItem> items`.

Walaupun begitu, pewarisan tetap digunakan, yaitu pada *abstract class* `Page` dan *subclass*-nya, yaitu `MainMenuPage`, `AddItemPage`, dan `ViewContentPage`. Alasannya adalah sebagaimana disebutkan di atas, pewarisan merupakan hubungan *is-a*. `MainMenuPage`, `AddItemPage`, dan `ViewContentPage` semuanya merupakan (*is-a*) `Page`. `Page` merupakan sebuah *abstract class* karena saya tidak ingin ada sebuah `Page` yang diinisialisasi. `Page` hanya berfungsi sebagai *template*, bukan sebagai objek jadi.

Beberapa atribut dalam `Page` mempunyai *keyword* `protected` karena alasan yang mirip seperti di atas. Saya tidak ingin atribut-atribut tersebut diakses sembarang *class*. Namun, saya tetap memerlukan atribut tersebut pada *subclass* dari `Page`. Alhasil, atribut tersebut merupakan atribut `protected`. Jadi, hal ini saya lakukan bukan karena saya pelit *file*.

Lalu, prinsip terakhir yang diminta oleh soal adalah penggunaan *interface*. Satu-satunya *interface* yang ada pada program saya adalah `Displayable`. *Class* yang mengimplementasikannya adalah `Page`. `Displayable` sendiri mempunyai dua metode saja, yaitu `void display()` dan `Displayable handleInput(String input)`. `display()` diimplementasikan oleh `Page` dalam bentuk memanggil `System.out.println(title)` dan memanggil metode `displayContent()` yang merupakan *abstract method* dari `Page`. Hal ini dilakukan karena konten yang akan ditampilkan masing-masing *subclass* `Page` berbeda, tetapi tetap ada hal yang akan selalu ditampilkan, yaitu judul (`title`) dari halaman saat itu. Dengan demikian, *subclass* `Page` diperkenankan mengimplementasikan *abstract method* itu sesuka hatinya, tetapi satu hal yang pasti adalah sebuah halaman akan selalu mempunyai judul.

Sebagian dari prinsip SOLID juga digunakan dalam program ini. Contohnya adalah *Single Responsibility Principle* (SRP) dan *Open-Closed Principle* (OCP). SRP dapat dilihat pada

metode main() yang *dumb*. Tanggung jawabnya hanya satu, yaitu meng-*instantiate* `App` dan menjalankannya (`run()`). Sementara itu, OCP digunakan pada *abstract class* `Page` dan *subclass*-nya. Metode `display()` pada `Page`, sekali dibuat, tidak diubah lagi. Yang diubah oleh *subclass*-nya adalah metode `displayContent()`.

Kedua prinsip tersebut secara teknis bukan merupakan "teknik". Lebih tepatnya adalah *mindset* yang mengarahkan program. Dari kelima prinsip SOLID, dua tersebut menjadi yang paling saya tonjolkan karena manfaatnya sangat mudah dirasakan. Tanpa SRP, program pasti akan mengalami *code smell*. Tanpa OCP, *debugging* akan sangat sulit dilakukan karena banyak hal yang berubah. Hal ini sangat krusial apalagi saya sedang mengalami krisis waktu sehingga saya tidak bisa berlama-lama pada suatu tugas.

Insight

Tugas ini sangat mengajarkan saya banyak hal, terutama perancangan *software* yang baik. Sejujurnya, saya menggunakan Gemini 2.5 Pro untuk memberikan saya gambaran tentang susunan program, karena pada awalnya saya tidak tahu harus memulai dari mana. Namun, saya tidak memintanya untuk membuat kode, melainkan hanya mengarahkan saya. Keahlian *prompting* saya juga tidak sebagus teman sebaya saya, jadi saya tidak mampu menyelesaikan tugas ini dalam waktu 15 menit. Saya juga lebih tertarik belajar lebih dalam mengenai prinsip pengembangan *software* yang baik daripada hanya menyelesaikan tugasnya. Tidak seperti sebagian besar mahasiswa lain. Bahkan, aslinya saya tidak ingin menggunakan AI, tetapi saya terpaksa memanfaatkan sumber daya yang ada karena kekurangan waktu yang saya alami seminggu ini.

Satu hal yang baru pertama kalinya saya lihat, apalagi gunakan, adalah metode yang mengembalikan (*return*) nilai berupa sebuah *interface*, yaitu Displayable handleInput(String input). Secara riilnya, yang dikembalikan adalah *class* yang mengimplementasikan *interface* tersebut, terutama `MainMenuPage`, `AddItemPage`, dan `ViewContentPage`. Namun, tetap saja ini merupakan sebuah wawasan (*insight*) baru bagi saya. Saya sudah pernah melihat metode yang mengembalikan nilai Boolean dalam bahasa C#, tetapi karena saya terbiasa menyaksikan metode mengembalikan nilai berupa tipe data primitif seperti *integer* dan *character*, saya juga cukup terkejut.

Tautan GitHub

https://github.com/orizynpx/CLI-Template