LAPORAN TUGAS

MATA KULIAH PEMROGRAMAN 2

Dosen Pengampu: Irham Maulani Abdul Gani, S.Kom., M.Kom.
NIP. 199710312025061009.

TEMPLATE CLI

Disusun Oleh:

AFRIAN PRADIPTA RIZKY

2410817210028

PROGRAM STUDI TEKNOLOGI INFORMASI

FAKULTAS TEKNIK

UNIVERSITAS LAMBUNG MANGKURAT

2025

SOAL

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

Spec:

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

Tabel 1. Source Code file GuestEntry

```
1   package CLI;
2
3   public class GuestEntry {
4       private String name;
5       private String purpose;
6
7       public GuestEntry(String name, String purpose) {
8           this.name = name;
9           this.purpose = purpose;
10      }
11
12      public String getName() {
13          return name;
14      }
15
16      public String getPurpose() {
17          return purpose;
18      }
19
20      @Override
21      public String toString() {
```

```
22          return "Nama: " + name + ", tujuan: " + purpose;
23      }
24  }
```

Tabel 2. Source Code GuestBookContract

```
1   package CLI;
2   import java.util.List;
3
4   public interface GuestBookContract {
5       void addGuest(String name, String purpose);
6       List<GuestEntry> getAllGuest();
7   }
```

Tabel 3. Source Code ContractImplement

```
1   package CLI;
2   import java.util.ArrayList;
3   import java.util.List;
4
5   public class ContractImplement implements GuestBookContract {
6       private List<GuestEntry> entries = new ArrayList<>();
7
8       @Override
9       public void addGuest(String name, String purpose) {
10          GuestEntry newEntry = new GuestEntry(name, purpose);
11          this.entries.add(newEntry);
12          System.out.println("\n  Tamu  '"+  name  +  "'  berhasil
    ditambahkan.");
13      }
14
15      @Override
16      public List<GuestEntry> getAllGuest() {
17          return this.entries;
18      }
19  }
```

Tabel 4. Source Code Abstractpage

```
1   package CLI;
```

```
2
3   import java.util.Scanner;
4
5   public abstract class Abstractpage {
6       protected Scanner scanner;
7       protected GuestBookContract GuestBookContract;
8
9       public Abstractpage(Scanner scanner, GuestBookContract
    guestBookContract) {
10          this.scanner = scanner;
11          GuestBookContract = guestBookContract;
12      }
13
14      public abstract void display();
15  }
```

Tabel 5. Source Code AddGuestPage

```
1   package CLI;
2   import java.util.Scanner;
3
4   public class AddGuestPage extends Abstractpage {
5       public AddGuestPage(Scanner scanner, GuestBookContract
    guestBookContract) {
6           super(scanner, guestBookContract);
7       }
8
9       @Override
10      public void display() {
11          System.out.println("1. tambah kunjungan tamu: ");
12          System.out.println("Masukkan nama anda: ");
13          String name = scanner.nextLine();
14
15          System.out.println("Masukkan tujuan anda: ");
16          String purpose = scanner.nextLine();
17
18          GuestBookContract.addGuest(name, purpose);
19      }
```

```
20  }
```

Tabel 6. Source Code ViewGuestPage

```
1   package CLI;
2   import java.util.List;
3   import java.util.Scanner;
4
5   public class ViewGuestPage extends Abstractpage {
6       public ViewGuestPage(Scanner scanner, GuestBookContract
    guestBookContract) {
7           super(scanner, guestBookContract);
8       }
9
10      @Override
11      public void display() {
12          System.out.println("2. Lihat daftar tamu: ");
13          List<GuestEntry>                guests                =
    GuestBookContract.getAllGuest();
14
15          if (guests.isEmpty()){
16              System.out.println("belum     ada     tamu     yang
    berkunjung");
17          }else{
18              int i = 1;
19              for (GuestEntry guest : guests){
20                  System.out.println(i +". " + guest.toString());
21                  i++;
22              }
23          }
24      }
25  }
```

Tabel 7. Source Code GuestBookApp

```
1   package CLI;
2
3   import java.util.Scanner;
4
```

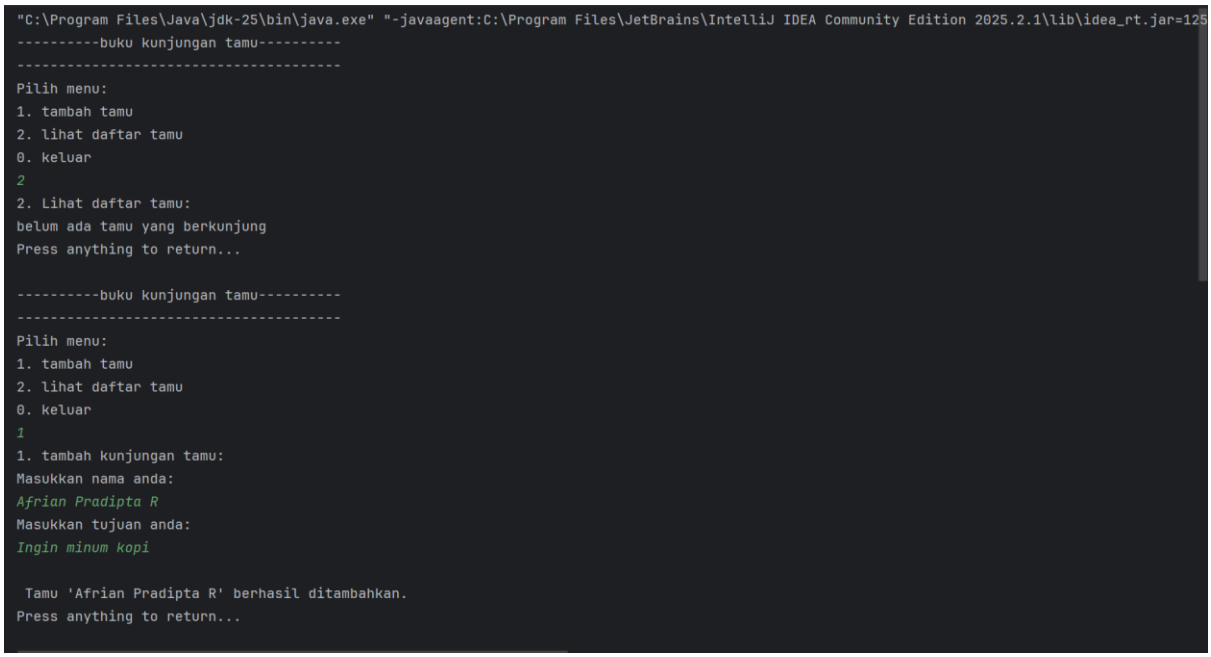
```
5  public class GuestBookApp {
6      private Scanner scanner;
7      private GuestBookContract GuestBookContract;
8      private Abstractpage addGuestPage;
9      private Abstractpage viewGuestPage;
10
11     public GuestBookApp() {
12         this.scanner = new Scanner(System.in);
13         GuestBookContract = new ContractImplement();
14         this.addGuestPage    =    new    AddGuestPage(scanner,
   GuestBookContract);
15         this.viewGuestPage    =    new    ViewGuestPage(scanner,
   GuestBookContract);
16     }
17
18     public void run(){
19         boolean running = true;
20
21         while (running){
22             printMenu();
23             String choice = scanner.nextLine();
24
25             switch (choice) {
26                 case "1":
27                     addGuestPage.display();
28                     break;
29                 case "2":
30                     viewGuestPage.display();
31                     break;
32                 case "0":
33                     running = false;
34                     break;
35                 default:
36                     break;
37
38             }
39
```

```
40              if (running){
41                  System.out.println("Press     anything     to
    return...");
42                  scanner.nextLine();
43              }
44          }
45          System.out.println("thankyou for using this shit");
46          scanner.close();
47      }
48
49      private void printMenu(){
50          System.out.println("----------buku kunjungan tamu-----
    -----");
51          System.out.println("----------------------------------
    -----");
52          System.out.println("Pilih menu: ");
53          System.out.println("1. tambah tamu");
54          System.out.println("2. lihat daftar tamu");
55          System.out.println("0. keluar");
56      }
57  }
```

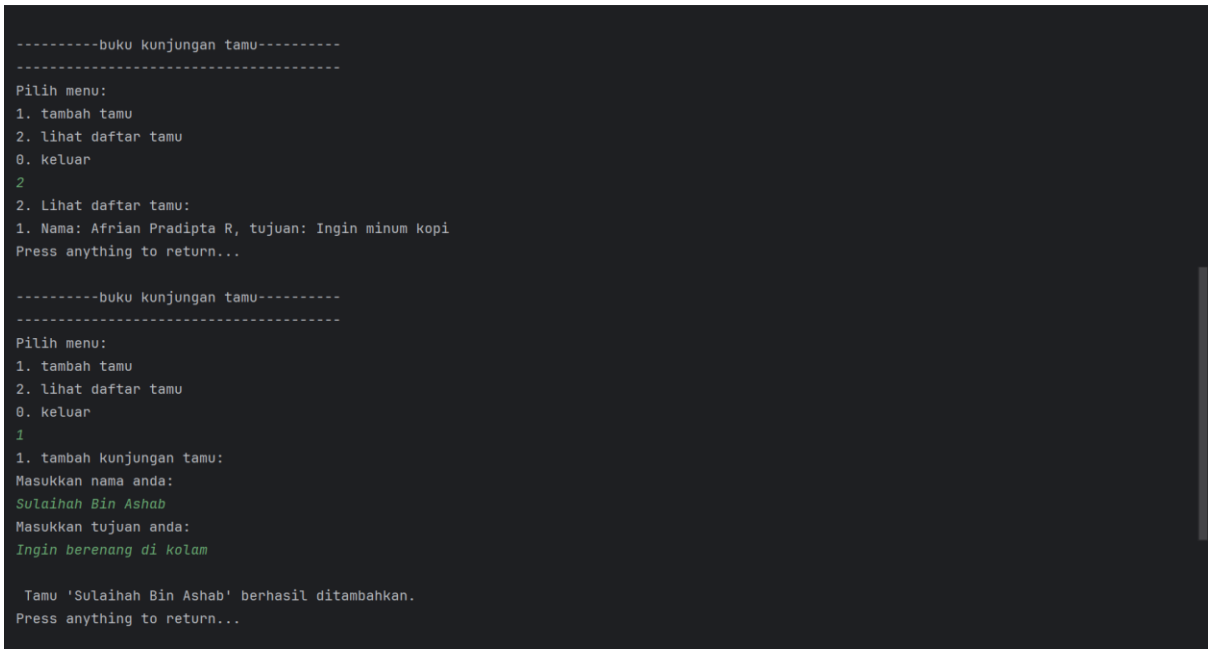
Tabel 8. Source Code Main

```
1   package CLI;
2
3   public class Main {
4       public static void main(String []args){
5           GuestBookApp app = new GuestBookApp();
6           app.run();
7       }
8   }
```

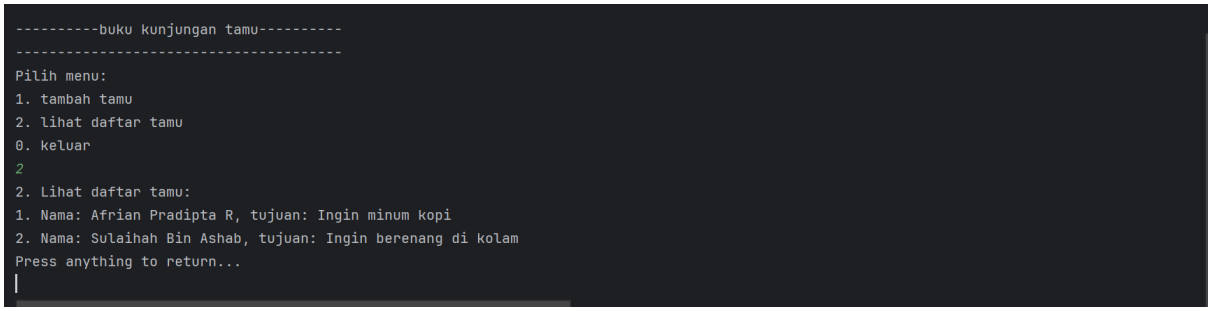
B. OUTPUT PROGRAM



```
"C:\Program Files\Java\jdk-25\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.2.1\lib\idea_rt.jar=125
----------buku kunjungan tamu----------
---------------------------------------
Pilih menu:
1. tambah tamu
2. lihat daftar tamu
0. keluar
2
2. Lihat daftar tamu:
belum ada tamu yang berkunjung
Press anything to return...

----------buku kunjungan tamu----------
---------------------------------------
Pilih menu:
1. tambah tamu
2. lihat daftar tamu
0. keluar
1
1. tambah kunjungan tamu:
Masukkan nama anda:
Afrian Pradipta R
Masukkan tujuan anda:
Ingin minum kopi

 Tamu 'Afrian Pradipta R' berhasil ditambahkan.
Press anything to return...
```



```
----------buku kunjungan tamu----------
---------------------------------------
Pilih menu:
1. tambah tamu
2. lihat daftar tamu
0. keluar
2
2. Lihat daftar tamu:
1. Nama: Afrian Pradipta R, tujuan: Ingin minum kopi
Press anything to return...

----------buku kunjungan tamu----------
---------------------------------------
Pilih menu:
1. tambah tamu
2. lihat daftar tamu
0. keluar
1
1. tambah kunjungan tamu:
Masukkan nama anda:
Sulaihah Bin Ashab
Masukkan tujuan anda:
Ingin berenang di kolam

 Tamu 'Sulaihah Bin Ashab' berhasil ditambahkan.
Press anything to return...
```



```
----------buku kunjungan tamu----------
---------------------------------------
Pilih menu:
1. tambah tamu
2. lihat daftar tamu
0. keluar
2
2. Lihat daftar tamu:
1. Nama: Afrian Pradipta R, tujuan: Ingin minum kopi
2. Nama: Sulaihah Bin Ashab, tujuan: Ingin berenang di kolam
Press anything to return...
```

C. PENJELASAN

Berikut adalah laporan yang menjelaskan pilihan desain dan wawasan yang didapat.

1. Alasan Penggunaan Teknik Desain

Proyek ini dibangun di atas tiga pilar OOP utama (Interface, Abstract Class, Composition) untuk mencapai aplikasi yang modular, mudah dikelola, dan terpisah (decoupled).

- Interface (GuestBookService)

  Prinsip: "Program to an interface, not an implementation."

  Alasan: Saya menggunakan Interface untuk mendefinisikan kontrak (apa yang harus dilakukan) dari layanan buku tamu. Kelas GuestBookApp dan AbstractPage tidak peduli bagaimana data disimpan (apakah di ArrayList, di file, atau di database). Mereka hanya peduli bahwa mereka dapat memanggil addGuest() dan getAllGuests().

  Keuntungan: Ini memisahkan logika bisnis (menyimpan tamu) dari logika aplikasi (menampilkan menu). Jika besok kita ingin mengganti penyimpanan dari ArrayList ke file, kita hanya perlu membuat GuestBookFileService yang baru dan meng-inject-nya ke GuestBookApp tanpa mengubah satu baris pun kode di kelas Halaman atau Aplikasi.

- Abstract Class (AbstractPage)

  Prinsip: "Template Method Pattern" dan "Code Reusability".

  Alasan: Saya mengidentifikasi bahwa semua "halaman" (seperti AddGuestPage dan ViewGuestsPage) memiliki kesamaan:

Mereka membutuhkan state (data) yang sama: Scanner untuk input dan GuestBookService untuk data.

Mereka memiliki perilaku (behavior) yang sama: mereka semua harus bisa display().

  Keuntungan: AbstractClass mengizinkan saya membuat "template".

Constructor-nya memastikan setiap halaman pasti menerima dependensi yang diperlukan (Scanner dan GuestBookService).

Metode abstract void display() memaksa setiap subclass (halaman) untuk menyediakan implementasi unik mereka sendiri. Ini adalah gabungan sempurna antara code sharing (untuk state) dan polimorfisme (untuk behavior).

- Composition (GuestBookApp dan GuestBookServiceImpl)

  Prinsip: "Favor Composition over Inheritance" (Utamakan Komposisi daripada Pewarisan).

  Alasan:

Di GuestBookApp: Kelas GuestBookApp bukanlah sebuah Scanner, dan bukanlah sebuah GuestBookService. Sebaliknya, GuestBookApp memiliki (has-a) Scanner, GuestBookService, dan semua Page. Ini adalah Composition. Aplikasi ini bertindak sebagai "Orchestrator" yang
Di GuestBookServiceImpl: Kelas ini bukanlah sebuah ArrayList. Kelas ini memiliki (has-a) ArrayList untuk menyimpan datanya. Ini adalah Encapsulation dan Composition yang bekerja sama.

Keuntungan: Desain ini jauh lebih fleksibel daripada pewarisan. GuestBookApp bisa dengan mudah mengganti implementasi GuestBookService-nya karena ia hanya menggunakannya (uses-a), bukan menjadi dia (is-a).

- DUMB Main dan Encapsulation

  Prinsip: "Single Responsibility Principle" (SRP).

Alasan: main (di Main.java) memiliki satu tanggung jawab: memulai program. GuestBookApp memiliki satu tanggung jawab: mengelola alur aplikasi. GuestBookService memiliki satu tanggung jawab: mengelola data tamu. AddGuestPage memiliki satu tanggung jawab: menampilkan halaman tambah tamu.

Keuntungan: Dengan memisahkan main, kita mencegah main menjadi "GOD Class" yang tahu segalanya dan melakukan segalanya. Semua logika ter-enkapsulasi dengan rapi di dalam kelas GuestBookApp.

2. Wawasan (Insight) yang Didapat

Pemisahan Tanggung Jawab (Separation of Concerns) adalah Kunci: Wawasan terbesar adalah betapa bersihnya kode ketika setiap kelas hanya melakukan satu hal. Service hanya mengurus data, Page hanya mengurus tampilan, dan App hanya mengurus alur (flow).

Interface Mendorong Desain yang 'Decoupled': Awalnya, mungkin terasa berlebihan membuat Interface hanya untuk ArrayList. Tetapi, ini "memaksa" saya untuk berpikir dalam kerangka "kontrak". Kelas-kelas Halaman sekarang tidak tahu (dan tidak peduli) tentang ArrayList. Mereka hanya tahu tentang "kontrak" GuestBookService.

Abstract Class adalah 'Jalan Tengah' yang Sempurna: Jika saya hanya menggunakan Interface untuk Page, saya harus mengulang kode untuk 'menyimpan' Scanner dan GuestBookService di setiap kelas Halaman. Jika saya menggunakan Inheritance biasa, saya mungkin tergoda memasukkan logika yang tidak perlu di superclass. Abstract Class memberikan yang terbaik dari keduanya: state yang dibagikan (shared state) dan kontrak perilaku yang dipaksakan (forced behavior).

Composition Membuat Kode Fleksibel: Desain ini (disebut Dependency Injection sederhana) di mana GuestBookApp membuat Service dan Scanner lalu "menyuntikkannya" ke dalam Page melalui constructor, membuat seluruh alur dependensi sangat jelas dan mudah dilacak. Ini juga membuat kode sangat mudah untuk di-test di masa depan.