LAPORAN UAS
PRAKTIKUM PEMOGRAMAN BERORIERNTASI OBJEK (PBO)

2411102441285 - HAFIDZAL MUFTY

2411102441309 – HAIDAR HALIM

2411102441227 – HIFZI KHAIRI

2411102441302 - MUHAMMAD IQBAL NUR SALIM

2411102441211 - MUHAMMAD NABIEL

2411102441207 – RIVOLDY MAKATITA

2411102441285 – ROHMI IHSAN

FAKULTAS SAINS DAN TEKNOLOGI

PROGRAM STUDI S1 TEKNIK INFORMATIKA

UNIVERSITAS MUHAMMADIYAH KALIMANTAN TIMUR

BAB I
PENDAHULUAN

A. Latar Belakang

Indonesia secara geografis terletak di kawasan cincin api yang membuatnya sangat rentan terhadap berbagai jenis bencana alam, seperti gempa bumi, banjir, dan tanah longsor. Dalam situasi darurat pasca-bencana, manajemen penanganan kesehatan menjadi faktor krusial yang menentukan keselamatan korban. Kecepatan dan ketepatan tindakan medis, atau yang sering dikenal dengan istilah golden hour, sangat bergantung pada ketersediaan data yang akurat mengenai kondisi korban, ketersediaan tenaga medis, serta stok obat-obatan di lapangan. Tanpa sistem yang terintegrasi, koordinasi antar posko bantuan sering kali terhambat, yang berpotensi menyebabkan keterlambatan penanganan yang fatal.

Meskipun urgensi penanganan kesehatan sangat tinggi, praktik manajemen di lapangan sering kali masih menghadapi kendala fundamental akibat penggunaan metode konvensional. Pendataan korban yang dilakukan secara manual menggunakan kertas berisiko tinggi mengalami kerusakan, hilang, atau duplikasi data, sehingga menyulitkan petugas untuk memprioritaskan korban yang berada dalam kondisi kritis (triase merah). Selain itu, masalah logistik medis juga sering terjadi, di mana stok obat vital tidak terpantau pergerakannya secara real-time. Akibatnya, sering ditemukan kasus kekurangan obat di satu posko sementara posko lain mengalami penumpukan, atau bahkan penggunaan obat yang sudah kadaluarsa karena minimnya validasi saat penerimaan bantuan logistik.

Sebagai solusi atas permasalahan tersebut, dikembangkanlah Disaster Healthcare Management System, sebuah sistem informasi berbasis Command Line Interface (CLI) yang dirancang menggunakan pendekatan Object-Oriented Programming (OOP). Sistem ini menawarkan sentralisasi data melalui penerapan Repository Pattern untuk mengelola informasi bencana, posko, dan sumber daya manusia secara terstruktur. Untuk mengatasi masalah logistik, aplikasi ini memiliki fitur otomasi inventaris yang mampu memvalidasi tanggal kadaluarsa obat dan memantau stok secara otomatis setiap kali resep dibuat. Lebih jauh lagi, sistem ini mengintegrasikan data klinis melalui layanan pemeriksaan yang secara otomatis menyinkronkan status triase korban pasca-penanganan, sehingga kontinuitas rekam medis korban dapat terjaga dengan baik. Dengan

demikian, kehadiran sistem ini diharapkan mampu meningkatkan efisiensi, akuntabilitas, dan kecepatan respon dalam penanganan kesehatan di wilayah terdampak bencana.

B. Rumusan Masalah

- Bagaimana merancang sistem yang dapat mengelola data bencana, posko, korban, dan logistik obat secara terintegrasi menggunakan konsep OOP?
- Bagaimana menerapkan arsitektur *Layered Architecture* untuk memisahkan logika bisnis dan data?

C. Tujuan

- Membangun aplikasi manajemen kesehatan bencana berbasis CLI.
- Menerapkan konsep OOP (Enkapsulasi, Inheritance, Polymorphism) dan prinsip SOLID.

BAB II
PERANCANGAN SISTEM

A. Use Case Scenario (Narasi Alur)

Skenario ini menggambarkan urutan aktivitas yang dilakukan oleh Admin atau Petugas Posko dalam mengoperasikan Disaster Healthcare Management System untuk menangani situasi tanggap darurat bencana.

Skenario Utama: Manajemen Terpadu Penanganan Medis Bencana Gempa Bumi.

Aktor: Admin / Petugas Posko.

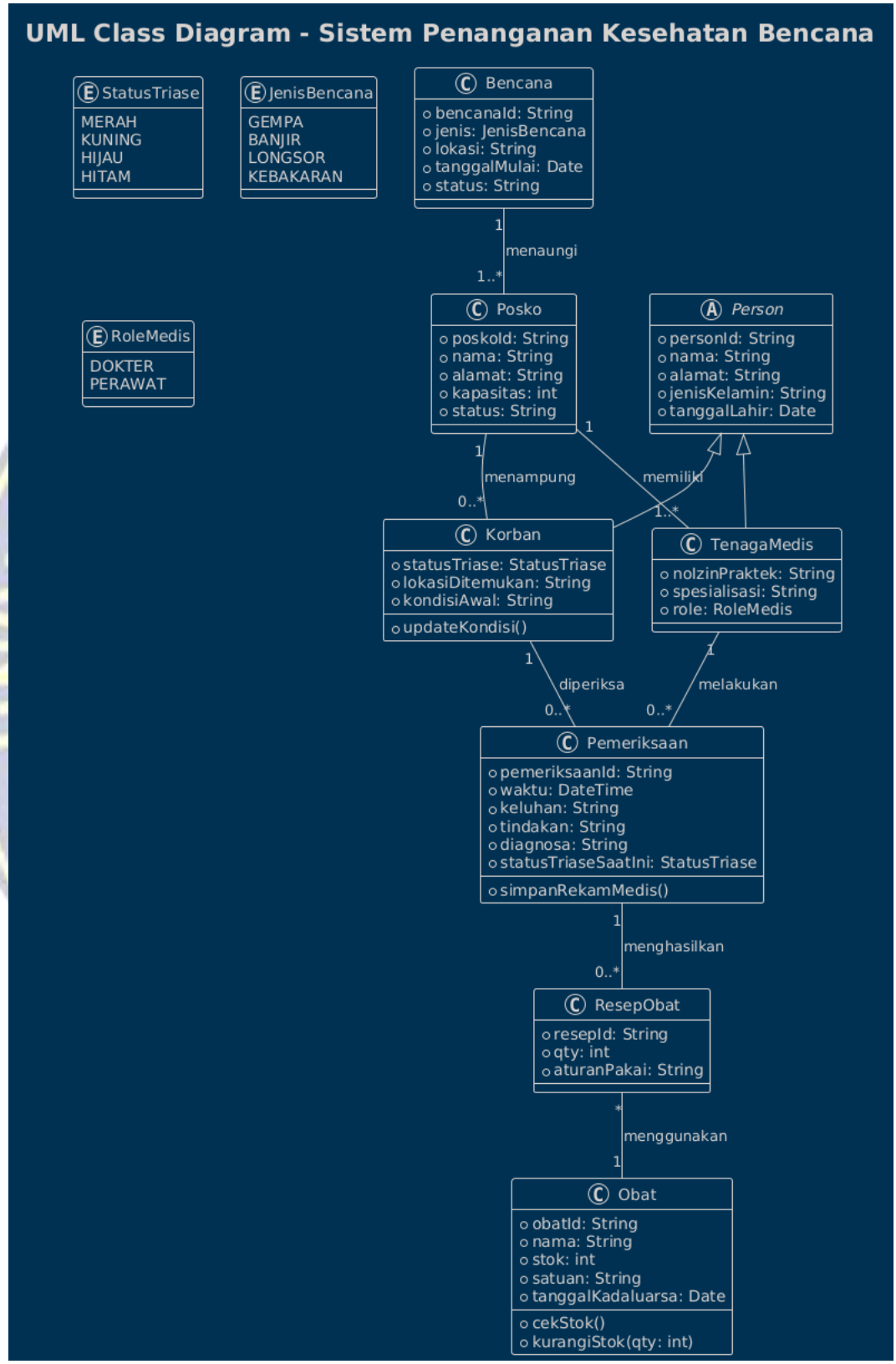
Alur Cerita:

1. Inisialisasi Data Bencana dan Lokasi Sistem dimulai dengan Admin mencatat kejadian bencana baru ke dalam sistem. Admin memasukkan data "Gempa Bumi" yang terjadi di lokasi "Cianjur" dengan status "Aktif". Setelah data bencana terbentuk, Admin mendirikan pusat bantuan dengan membuat data Posko (misalnya: "Posko Utama Alun-alun") yang secara sistem terhubung (berelasi) dengan ID bencana yang baru saja dibuat.
2. Mobilisasi Tenaga Medis Setelah posko tersedia, Admin mendaftarkan Tenaga Medis yang bertugas di lokasi tersebut. Admin menginput data diri dokter (contoh: Dr. Budi Santoso), termasuk nomor izin praktik dan spesialisasi. Sistem menyimpan data ini dan menetapkan dokter tersebut sebagai personel yang bertanggung jawab di Posko Utama.
3. Registrasi dan Triase Korban Seorang korban bernama Ahmad dievakuasi ke Posko Utama. Admin segera mendaftarkan data Korban tersebut ke dalam sistem dengan mencatat kondisi awal "Luka ringan" dan lokasi ditemukannya. Berdasarkan penilaian awal, Admin menetapkan status triase "Kuning" (perlu penanganan segera namun tidak kritis). Data korban ini secara otomatis terhubung dengan Posko Utama.
4. Pemeriksaan Medis (Diagnosis) Dr. Budi melakukan pemeriksaan terhadap Ahmad. Admin mencatat hasil pemeriksaan ini ke dalam modul Pemeriksaan. Data yang direkam meliputi keluhan ("Nyeri kaki"), diagnosa dokter ("Lecet dan memar"), serta tenaga medis yang menangani. Setelah penanganan, status triase korban diperbarui di

sistem menjadi "Hijau" (kondisi membaik/stabil) sebagai hasil dari sinkronisasi otomatis sistem.

5. Pemberian Resep Obat Berdasarkan diagnosa, dokter meresepkan obat Paracetamol dan Betadine. Admin memasukkan data Resep Obat yang terhubung dengan ID Pemeriksaan sebelumnya. Sistem secara otomatis memvalidasi ketersediaan stok di inventaris. Jika stok mencukupi, sistem akan mengurangi jumlah obat secara real-time dan menyimpan resep tersebut sebagai riwayat pengobatan korban.

B. UML Class Diagram

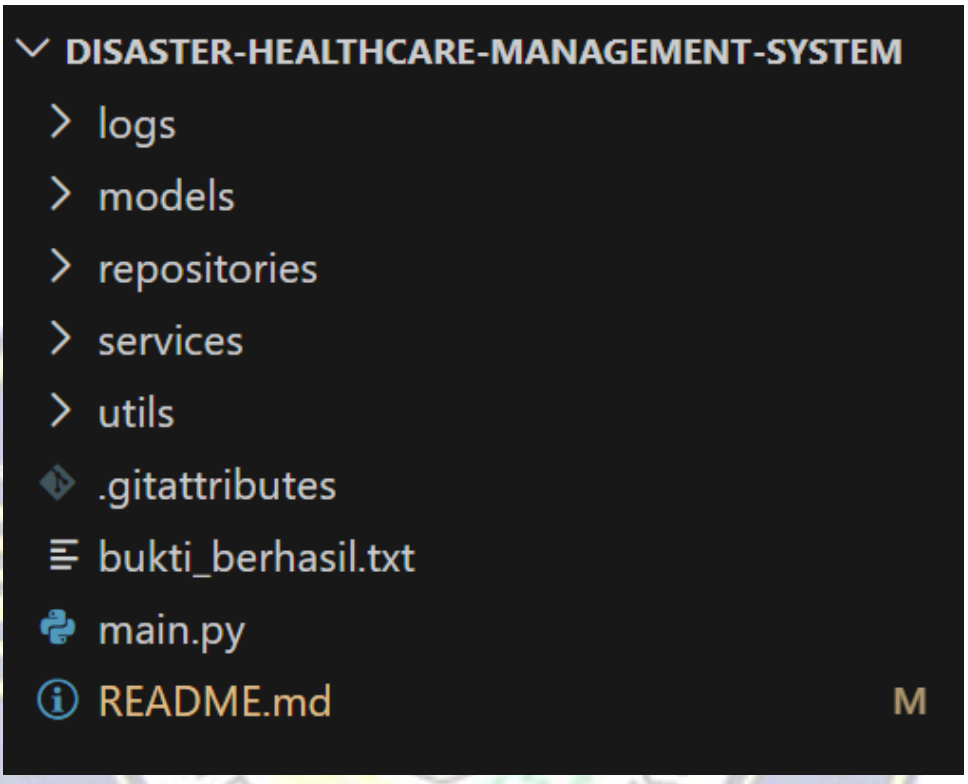


1. Inheritance (Pewarisan)
   Hubungan: Korban dan TenagaMedis adalah turunan dari Orang.
   a. Konsep: Hubungan ini bersifat "is-a" (adalah sebuah). Artinya, seorang Korban adalah Orang, dan seorang Tenaga Medis juga adalah Orang. Pewarisan digunakan untuk menghindari duplikasi kode (code reuse) pada atribut yang sama.
   b. Implementasi dalam Kode:
      1) Parent Class (Orang): Didefinisikan sebagai Abstract Base Class (ABC). Kelas ini menyimpan atribut umum yang dimiliki oleh setiap manusia dalam sistem, yaitu: id_orang, nama_orang, alamat_orang, jenis_kelamin_orang, dan tanggal_lahir_orang.
      2) Child Class (Korban): Mewarisi semua atribut dari Orang, lalu menambahkan atribut spesifik untuk pasien bencana, seperti status_triase, kondisi_awal, dan lokasi_ditemukan.
      3) Child Class (TenagaMedis): Mewarisi atribut Orang, lalu menambahkan atribut spesifik profesi, seperti no_izin_praktik, role, dan spesialisasi.
2. Association (Asosiasi)
   Hubungan: Posko memiliki hubungan dengan Bencana.
   a. Konsep: Hubungan ini bersifat "has-a" atau "related-to". Asosiasi menggambarkan bahwa suatu objek terhubung dengan objek lain untuk melengkapi informasi konteksnya. Dalam kasus ini, sebuah Posko tidak berdiri sendiri, melainkan didirikan untuk menangani Bencana tertentu.
   b. Implementasi dalam Kode:
      1) Di dalam class Posko, terdapat atribut bencana yang bertipe data objek Bencana.
      2) Saat objek Posko diinisialisasi (__init__), ia meminta objek Bencana sebagai salah satu argumennya. Ini menciptakan tautan navigasi di mana kita bisa mengetahui bencana apa yang sedang ditangani oleh posko tersebut hanya dengan memanggil posko.get_bencana().
3. Dependency (Ketergantungan)
   Hubungan: ResepObatService bergantung pada ObatRepository dan PemeriksaanRepository.
   a. Konsep: Hubungan ini bersifat "uses-a" (menggunakan). Sebuah class (Service) membutuhkan bantuan dari class lain (Repository) untuk dapat menjalankan fungsinya (logika bisnis). Jika repository tidak ada, service tidak bisa bekerja.
   b. Implementasi dalam Kode:

1) Dependency Injection: Pada constructor __init__ di ResepObatService, service ini menerima parameter resep_repo, pemeriksaan_repo, dan obat_repo. Service tidak membuat objek repository sendiri (tidak new Repository()), melainkan "disuntikkan" dari luar.
2) Penggunaan:
    a) Service menggunakan pemeriksaan_repo untuk memvalidasi apakah ID pemeriksaan yang dimasukkan benar-benar ada (ambil_berdasarkan_id).
    b) Service menggunakan obat_repo untuk mengecek stok obat (ambil_berdasarkan_id) dan mengurangi stok (perbarui) saat resep berhasil dibuat.
    3) Tanpa keberadaan repository tersebut, logika bisnis di ResepObatService (seperti validasi stok dan pengurangan barang) akan gagal total.

BAB III
IMPLEMENTASI TEKNIS & KONSEP OOP

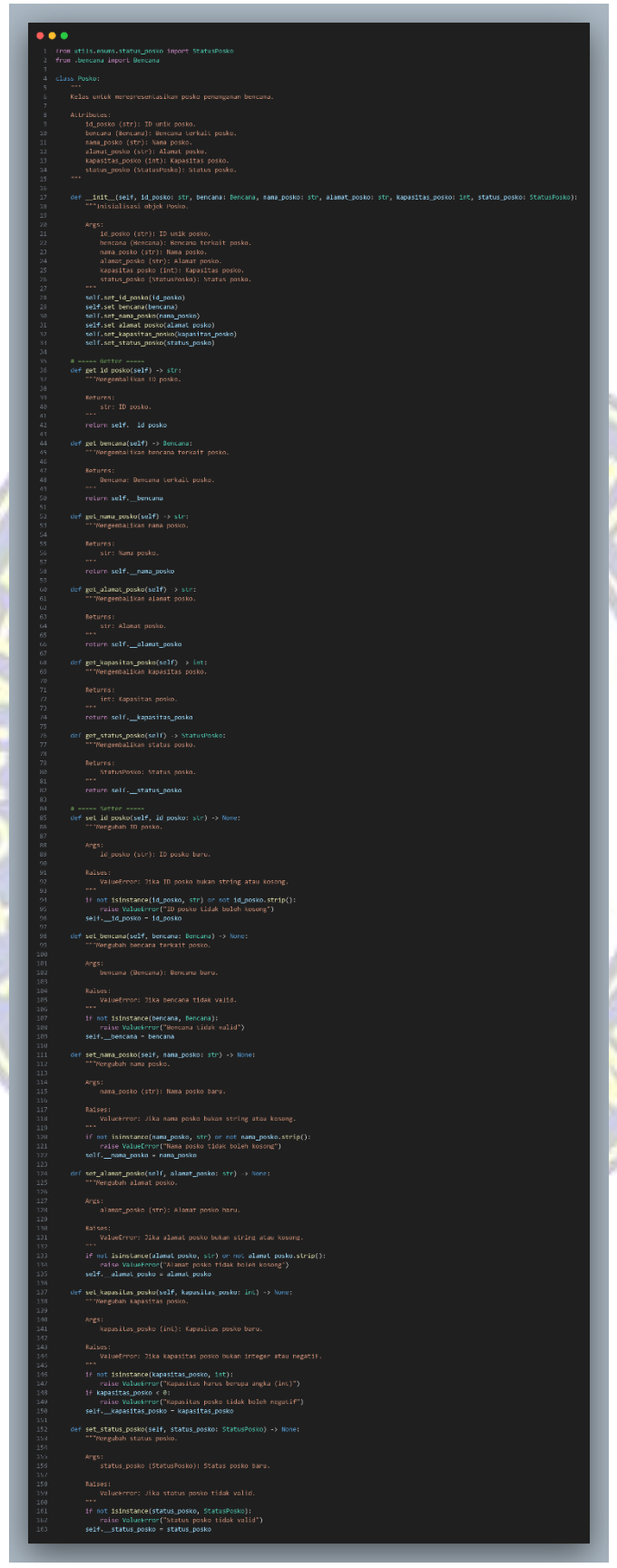
A. Struktur Proyek & Modularisasi



1. models/: Menyimpan definisi objek (blueprint).
2. repositories/: Menyimpan data in-memory (pengganti database).
3. services/: Mengatur logika bisnis dan validasi.
4. main.py: Titik masuk aplikasi (Orchestrator).

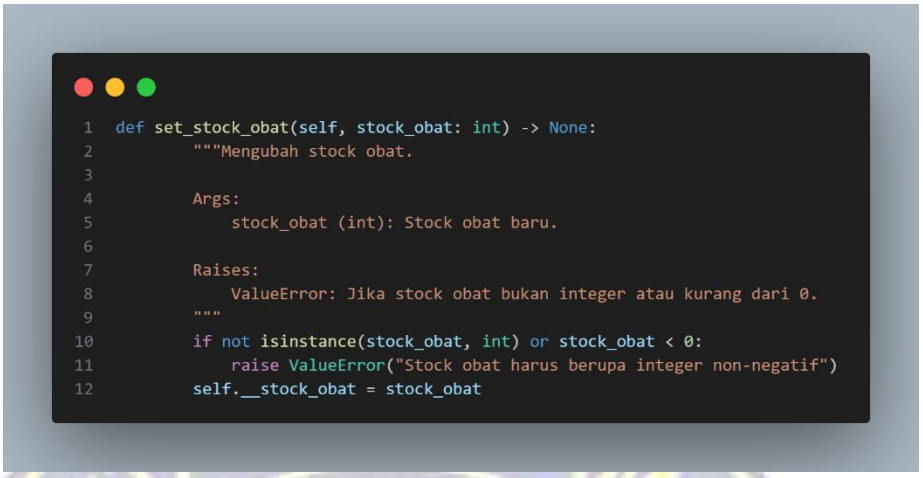
B. Penerapan Core OOP

1. Class & Object:



```python
from utils.enums.status_posko import StatusPosko
from .bencana import Bencana

class Posko:
    """
    Kelas untuk merepresentasikan posko penanganan bencana.

    Atributtes:
        id_posko (str): ID unik posko.
        bencana (Bencana): Bencana terkait posko.
        nama_posko (str): Nama posko.
        alamat_posko (str): Alamat posko.
        kapasitas_posko (int): Kapasitas posko.
        status_posko (StatusPosko): Status posko.
    """

    def __init__(self, id_posko: str, bencana: Bencana, nama_posko: str, alamat_posko: str, kapasitas_posko: int, status_posko: StatusPosko):
        """Inisialisasi objek Posko.

        Args:
            id_posko (str): ID unik posko.
            bencana (Bencana): Bencana terkait posko.
            nama_posko (str): Nama posko.
            alamat_posko (str): Alamat posko.
            kapasitas_posko (int): Kapasitas posko.
            status_posko (StatusPosko): Status posko.
        """
        self.set_id_posko(id_posko)
        self.set_bencana(bencana)
        self.set_nama_posko(nama_posko)
        self.set_alamat_posko(alamat_posko)
        self.set_kapasitas_posko(kapasitas_posko)
        self.set_status_posko(status_posko)

    # ===== Getter =====
    def get_id_posko(self) -> str:
        """Mengembalikan ID posko.

        Returns:
            str: ID posko.
        """
        return self.__id_posko

    def get_bencana(self) -> Bencana:
        """Mengembalikan bencana terkait posko.

        Returns:
            Bencana: Bencana terkait posko.
        """
        return self.__bencana

    def get_nama_posko(self) -> str:
        """Mengembalikan nama posko.

        Returns:
            str: Nama posko.
        """
        return self.__nama_posko

    def get_alamat_posko(self)  -> str:
        """Mengembalikan alamat posko.

        Returns:
            str: Alamat posko.
        """
        return self.__alamat_posko

    def get_kapasitas_posko(self)  -> int:
        """Mengembalikan kapasitas posko.

        Returns:
            int: Kapasitas posko.
        """
        return self.__kapasitas_posko

    def get_status_posko(self) -> StatusPosko:
        """Mengembalikan status posko.

        Returns:
            StatusPosko: Status posko.
        """
        return self.__status_posko

    # ===== Setter =====
    def set_id_posko(self, id_posko: str) -> None:
        """Mengubah ID posko.

        Args:
            id_posko (str): ID posko baru.

        Raises:
            ValueError: Jika ID posko bukan string atau kosong.
        """
        if not isinstance(id_posko, str) or not id_posko.strip():
            raise ValueError("ID posko tidak boleh kosong")
        self.__id_posko = id_posko

    def set_bencana(self, bencana: Bencana) -> None:
        """Mengubah bencana terkait posko.

        Args:
            bencana (Bencana): Bencana baru.

        Raises:
            ValueError: Jika bencana tidak valid.
        """
        if not isinstance(bencana, Bencana):
            raise ValueError("Bencana tidak valid")
        self.__bencana = bencana

    def set_nama_posko(self, nama_posko: str) -> None:
        """Mengubah nama posko.

        Args:
            nama_posko (str): Nama posko baru.

        Raises:
            ValueError: Jika nama posko bukan string atau kosong.
        """
        if not isinstance(nama_posko, str) or not nama_posko.strip():
            raise ValueError("Nama posko tidak boleh kosong")
        self.__nama_posko = nama_posko

    def set_alamat_posko(self, alamat_posko: str)  -> None:
        """Mengubah alamat posko.

        Args:
            alamat_posko (str): Alamat posko baru.

        Raises:
            ValueError: Jika alamat posko bukan string atau kosong.
        """
        if not isinstance(alamat_posko, str) or not alamat_posko.strip():
            raise ValueError('Alamat posko tidak boleh kosong')
        self.__alamat_posko = alamat_posko

    def set_kapasitas_posko(self, kapasitas_posko: int) -> None:
        """Mengubah kapasitas posko.

        Args:
            kapasitas_posko (int): Kapasitas posko baru.

        Raises:
            ValueError: Jika kapasitas posko bukan integer atau negatif.
        """
        if not isinstance(kapasitas_posko, int):
            raise ValueError("Kapasitas harus berupa angka (int)")
        if kapasitas_posko < 0:
            raise ValueError("Kapasitas posko tidak boleh negatif")
        self.__kapasitas_posko = kapasitas_posko

    def set_status_posko(self, status_posko: StatusPosko) -> None:
        """Mengubah status posko.

        Args:
            status_posko (StatusPosko): Status posko baru.

        Raises:
            ValueError: Jika status posko tidak valid.
        """
        if not isinstance(status_posko, StatusPosko):
            raise ValueError("Status posko tidak valid")
        self.__status_posko = status_posko
```

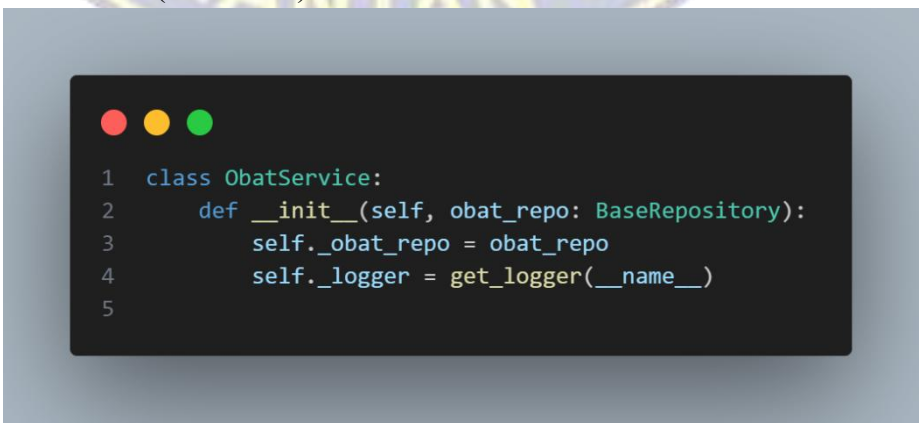
2. Enkapsulasi:



```python
def set_stock_obat(self, stock_obat: int) -> None:
        """Mengubah stock obat.

        Args:
            stock_obat (int): Stock obat baru.

        Raises:
            ValueError: Jika stock obat bukan integer atau kurang dari 0.
        """
        if not isinstance(stock_obat, int) or stock_obat < 0:
            raise ValueError("Stock obat harus berupa integer non-negatif")
        self.__stock_obat = stock_obat
```

   a. Atribut Private (__stock_obat): Dengan menambahkan awalan __, Python akan melakukan name mangling sehingga atribut ini tidak bisa diakses atau diubah secara sembarangan dari luar objek (misalnya: obat.stock_obat = -50 akan gagal atau dianggap atribut baru, tidak mengubah data asli). Hal ini melindungi integritas data dari modifikasi yang tidak disengaja.
   b. Validasi Integritas Data (Melalui Setter): Penerapan Setter (set_stock_obat) berfungsi sebagai gerbang keamanan. Sebelum nilai baru dimasukkan ke dalam sistem, logika program memeriksa validitasnya terlebih dahulu.
      1) Masalah: Tanpa setter, seseorang bisa saja memasukkan jumlah obat -10 atau NaN (Not a Number), yang akan merusak perhitungan logistik.
      2) Solusi: Kode if stock_obat < 0: raise ValueError memastikan bahwa sistem menolak data yang tidak masuk akal. Dengan demikian, objek Obat dijamin selalu berada dalam kondisi valid (valid state) sepanjang siklus hidup aplikasi.

3. Inheritance (Pewarisan):

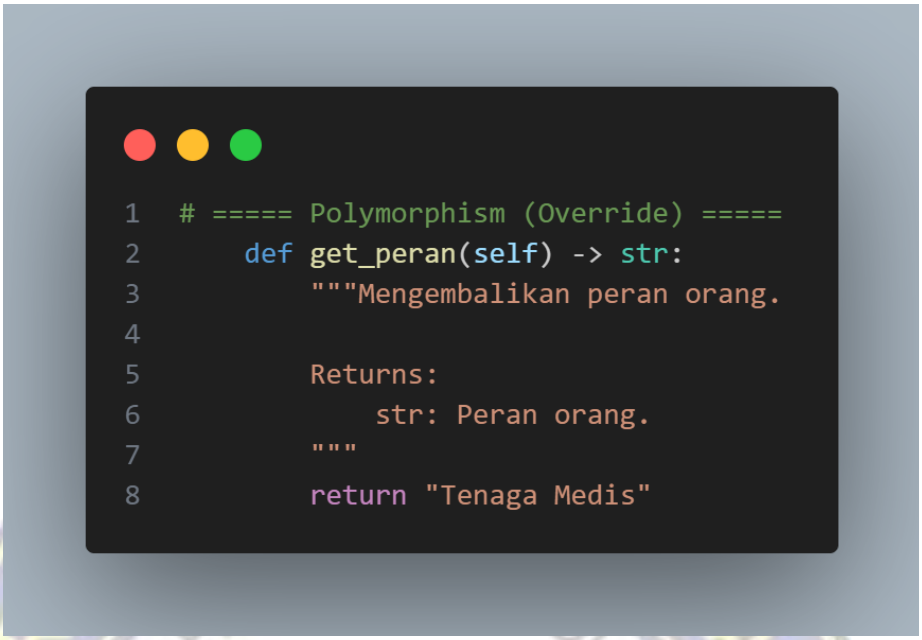


```python
class ObatService:
    def __init__(self, obat_repo: BaseRepository):
        self._obat_repo = obat_repo
        self._logger = get_logger(__name__)

```

4. Polymorphism (Polimorfisme):

```python
# ===== Abstract Method (Polymorphism) =====
    @abstractmethod
    def get_peran(self) -> str:
        """Harus dioverride oleh child class.

        Returns:
            str: Peran orang (contoh: 'Korban', 'Tenaga Medis').
        """
        pass
```

```python
# ===== Polymorphism (Override) =====
    def get_peran(self) -> str:
        """Mengembalikan peran orang.

        Returns:
            str: Peran orang.
        """
        return "Korban"
```



```python
# ===== Polymorphism (Override) =====
    def get_peran(self) -> str:
        """Mengembalikan peran orang.

        Returns:
            str: Peran orang.
        """
        return "Tenaga Medis"
```

C. Penerapan Prinsip SOLID

1. Single Responsibility Principle (SRP):
   a. Model: Hanya Mengurus "Struktur Data"
      1) Tanggung Jawab: Menyimpan atribut (data) dan memastikan validitas tipe data dasar.
      2) Contoh: File models/obat.py.
      3) Analisis SRP: Class Obat hanya peduli pada "apa itu Obat". Ia menyimpan id_obat, nama_obat, stock_obat, dll. Ia memiliki Getter/Setter untuk validasi sederhana (misal: stok tidak boleh negatif).
         a) Apa yang TIDAK dilakukan: Class ini tidak tahu cara menyimpannya ke database, dan tidak tahu aturan bisnis kompleks (misal: "obat kadaluarsa tidak boleh diresepkan").
         b) Alasan Berubah: Hanya jika atribut obat bertambah (misal: menambah kolom harga atau pabrik).
   b. Repository: Hanya Mengurus "Penyimpanan Data"
      1) Tanggung Jawab: Menangani operasi CRUD (Create, Read, Update, Delete) ke media penyimpanan (dalam hal ini in-memory dictionary).
      2) Contoh: File repositories/obat_repository.py.
      3) Analisis SRP: Class ObatRepositoryMemory bertindak sebagai "gudang". Tugasnya hanya menerima objek Obat dan menyimpannya ke dalam self._data (dictionary), atau mengambilnya kembali berdasarkan ID.

a) Apa yang TIDAK dilakukan: Repository tidak memvalidasi tanggal kadaluarsa. Ia menerima objek apa adanya. Ia juga tidak memproses logika transaksi.
b) Alasan Berubah: Hanya jika media penyimpanan berubah (misal: dari Memory/RAM diganti menjadi Database MySQL atau File CSV).

c. Service: Hanya Mengurus "Logika Bisnis"
1) Tanggung Jawab: Menjadi "otak" aplikasi. Mengatur alur kerja, validasi aturan bisnis, logging, dan menyambungkan data antar-repository.
2) Contoh: File services/obat_service.py.
3) Analisis SRP: Class ObatService mengatur kapan data boleh disimpan.
a) Di method buat_obat, ia mengecek: Apakah tanggal kadaluarsa di masa lalu?.
b) Di method kurangi_stok, ia mengecek: Apakah stok cukup?
c) Jika lolos validasi, barulah Service memanggil Repository untuk menyimpan data.
d) Alasan Berubah: Hanya jika aturan bisnis berubah (misal: "Obat kadaluarsa boleh disimpan tapi diberi tanda khusus").

2. Dependency Inversion Principle (DIP):

D. Penerapan Logging & Library

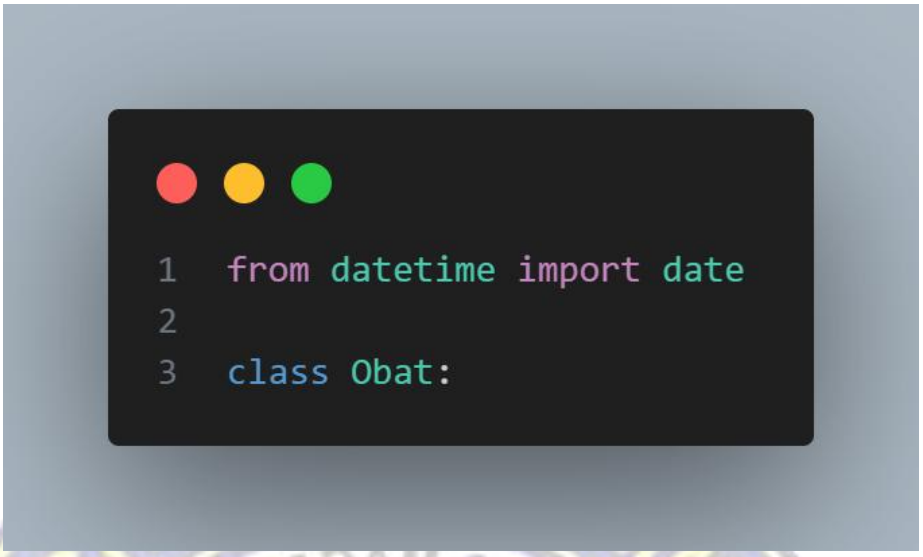
1. Longging

```python
class ObatService:
    def __init__(self, obat_repo: BaseRepository):
        self._obat_repo = obat_repo
        self._logger = get_logger(__name__)

```

2. uuid

```python
import uuid


def generate_id() -> str:
    """
    Menghasilkan ID unik berbasis UUID v4.

    Returns:
        str: ID unik dalam bentuk string.
    """
    return str(uuid.uuid4())
```

3. datetime



```
1  from datetime import date
2
3  class Obat:
```

BAB IV
PENGUJIAN & HASIL

A. Skenario Pengujian

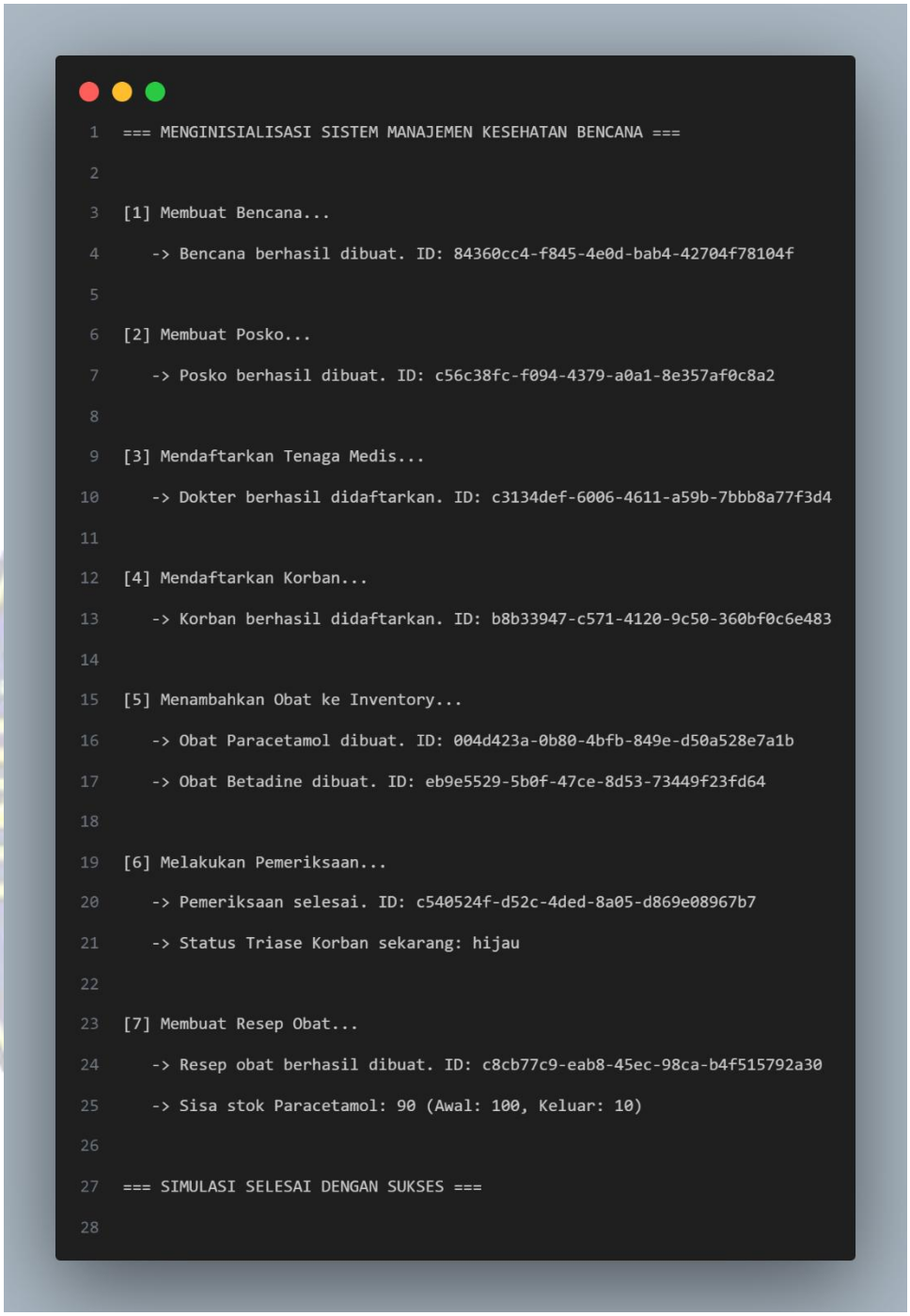
1. Inisialisasi Layer Data (Repository)
   Langkah pertama yang dilakukan sistem adalah menyiapkan tempat penyimpanan data. Karena sistem ini menggunakan penyimpanan in-memory (RAM), program membuat instance dari setiap kelas Repository (seperti BencanaRepositoryMemory, PoskoRepositoryMemory, dll).
   - Tujuan: Menyiapkan wadah kosong (dictionary) untuk menampung data Bencana, Korban, Obat, dll.
2. Inisialisasi Layer Bisnis (Service & Dependency Injection)
   Setelah wadah data siap, program menyiapkan layanan logika bisnis (Services). Pada tahap ini terjadi proses Dependency Injection, di mana objek Repository yang dibuat di langkah pertama "disuntikkan" ke dalam Service.
   - Contoh: PoskoService membutuhkan posko_repo dan bencana_repo agar bisa memvalidasi apakah bencana ada sebelum membuat posko.
3. Skenario: Pembuatan Data Bencana
   Simulasi dimulai dengan mencatat kejadian bencana baru.
   - Aksi: Admin membuat data "Gempa Bumi" yang terjadi di "Cianjur" dengan status "aktif".
   - Output: Sistem mengembalikan id_bencana unik yang akan digunakan untuk langkah selanjutnya.
4. Skenario: Pendirian Posko
   Berdasarkan ID bencana yang baru dibuat, sistem mendirikan posko bantuan.
   - Aksi: Membuat "Posko Utama Alun-alun" yang terhubung secara relasional dengan bencana Gempa Bumi tersebut.
5. Skenario: Pendaftaran Sumber Daya Manusia
   Setelah infrastruktur posko siap, sistem mendaftarkan orang-orang yang terlibat:
   - Tenaga Medis: Mendaftarkan "Dr. Budi Santoso" (Dokter Umum) dan menugaskannya di Posko Utama.
   - Korban: Mendaftarkan korban bernama "Ahmad" yang ditemukan di reruntuhan dengan kondisi luka ringan dan status triase awal "Kuning".
6. Skenario: Manajemen Logistik (Stok Obat)

Sebelum melakukan pengobatan, sistem mengisi inventaris obat.

- Aksi: Menambahkan dua jenis obat, yaitu "Paracetamol" (stok 100) dan "Betadine" (stok 50) lengkap dengan tanggal kadaluarsanya.

7. Skenario: Pemeriksaan Medis (Diagnosis & Update Triase)
   Dokter melakukan pemeriksaan terhadap korban.
   - Aksi: Mencatat keluhan "Nyeri kaki" dan diagnosa "Luka lecet".
   - Efek Sistem: Sistem secara otomatis mengubah status triase korban dari "Kuning" menjadi "Hijau" (membaik) karena parameter sinkron_triase_korban=True diaktifkan.
8. Skenario: Pembuatan Resep & Pengurangan Stok Otomatis
   Langkah terakhir adalah pemberian obat.
   - Aksi: Dokter meresepkan 10 butir Paracetamol dan 1 botol Betadine untuk Ahmad.
   - Validasi: ResepObatService mengecek apakah stok cukup.
   - Efek Sistem: Jika valid, resep dibuat dan stok Paracetamol di gudang otomatis berkurang dari 100 menjadi 90.
9. Selesai
   Jika seluruh proses di atas berjalan tanpa error, program mencetak pesan "SIMULASI SELESAI DENGAN SUKSES".

B. Hasil Output Program



```
=== MENGINISIALISASI SISTEM MANAJEMEN KESEHATAN BENCANA ===

[1] Membuat Bencana...
   -> Bencana berhasil dibuat. ID: 84360cc4-f845-4e0d-bab4-42704f78104f

[2] Membuat Posko...
   -> Posko berhasil dibuat. ID: c56c38fc-f094-4379-a0a1-8e357af0c8a2

[3] Mendaftarkan Tenaga Medis...
   -> Dokter berhasil didaftarkan. ID: c3134def-6006-4611-a59b-7bbb8a77f3d4

[4] Mendaftarkan Korban...
   -> Korban berhasil didaftarkan. ID: b8b33947-c571-4120-9c50-360bf0c6e483

[5] Menambahkan Obat ke Inventory...
   -> Obat Paracetamol dibuat. ID: 004d423a-0b80-4bfb-849e-d50a528e7a1b
   -> Obat Betadine dibuat. ID: eb9e5529-5b0f-47ce-8d53-73449f23fd64

[6] Melakukan Pemeriksaan...
   -> Pemeriksaan selesai. ID: c540524f-d52c-4ded-8a05-d869e08967b7
   -> Status Triase Korban sekarang: hijau

[7] Membuat Resep Obat...
   -> Resep obat berhasil dibuat. ID: c8cb77c9-eab8-45ec-98ca-b4f515792a30
   -> Sisa stok Paracetamol: 90 (Awal: 100, Keluar: 10)

=== SIMULASI SELESAI DENGAN SUKSES ===
```

Sistem berhasil membuat data Bencana dan Posko

BAB V
PENUTUP

A. Kesimpulan

Berdasarkan hasil perancangan dan implementasi kode, dapat disimpulkan bahwa aplikasi Disaster Healthcare Management System telah berhasil memenuhi seluruh kriteria teknis yang disyaratkan dalam Ujian Akhir Semester (UAS). Aplikasi ini tidak hanya berfungsi sebagai simulasi manajemen bencana, tetapi juga menjadi bukti penerapan disiplin rekayasa perangkat lunak yang baik melalui konsep berikut:

1. Penerapan Utuh Konsep OOP (Object-Oriented Programming)
   Aplikasi dibangun di atas empat pilar utama OOP yang menjamin struktur kode yang logis dan terorganisir:
   a. Encapsulation (Enkapsulasi): Seluruh atribut sensitif (seperti stok obat dan data pribadi korban) dilindungi menggunakan akses private dan divalidasi ketat melalui mekanisme Getter/Setter untuk menjaga integritas data.
   b. Inheritance (Pewarisan): Efisiensi kode tercapai melalui penggunaan parent class Orang yang mewariskan atribut dasar kepada child class Korban dan TenagaMedis, menghindari duplikasi kode yang redundan.
   c. Polymorphism (Polimorfisme): Fleksibilitas sistem terlihat pada implementasi metode tambah(), hapus(), dan ambil_semua() yang memiliki perilaku spesifik di setiap Repository (Bencana, Obat, Posko) meskipun berasal dari kontrak antarmuka yang sama.
   d. Abstraction (Abstraksi): Kompleksitas disembunyikan melalui penggunaan Abstract Base Class (BaseRepository) yang menetapkan kontrak CRUD standar tanpa mengekspos detail implementasi penyimpanan in-memory.

2. Kepatuhan Terhadap Prinsip SOLID

Arsitektur aplikasi dirancang agar robust (kokoh) dan maintainable (mudah dirawat) dengan menerapkan prinsip SOLID:

a. Single Responsibility Principle (SRP): Pemisahan tegas antara logika data (Models), penyimpanan (Repositories), dan bisnis (Services) memastikan setiap kelas hanya memiliki satu alasan untuk berubah.

b. Open/Closed Principle (OCP): Sistem terbuka untuk ekstensi (misalnya menambahkan tipe repositori baru seperti Database SQL) tanpa perlu memodifikasi kode Service yang sudah berjalan.
c. Liskov Substitution Principle (LSP): Objek turunan seperti Korban dapat menggantikan peran Orang dalam logika program tanpa menyebabkan error.
d. Interface Segregation Principle (ISP): Penggunaan antarmuka BaseRepository yang fokus dan spesifik memastikan kelas implementasi tidak dipaksa bergantung pada metode yang tidak mereka gunakan.
e. Dependency Inversion Principle (DIP): High-level modules (BencanaService) tidak bergantung pada low-level modules (BencanaRepositoryMemory), melainkan keduanya bergantung pada abstraksi (BaseRepository). Hal ini dibuktikan dengan teknik Dependency Injection pada konstruktor Service.

B. Saran Pengembangan

Meskipun aplikasi Disaster Healthcare Management System saat ini sudah berjalan baik dengan konsep OOP dan SOLID, terdapat beberapa aspek yang dapat ditingkatkan untuk membuat sistem lebih siap pakai di dunia nyata:

1. Implementasi Database Persisten (SQL/NoSQL)
   a. Kondisi Saat Ini: Sistem menggunakan penyimpanan in-memory (RAM) melalui dictionary Python di dalam kelas RepositoryMemory. Data akan hilang setiap kali program dimatikan.
   b. Saran: Mengganti penyimpanan data menggunakan database nyata seperti SQLite, MySQL, atau PostgreSQL.
   c. Cara Implementasi: Karena Anda sudah menggunakan Repository Pattern dan Interface BaseRepository, Anda cukup membuat kelas repository baru (misalnya BencanaRepositorySQL) yang mewarisi BaseRepository, lalu mengganti kode di main.py tanpa perlu mengubah logika bisnis di Service.
2. Pembuatan Antarmuka Grafis (GUI) atau Web
   a. Kondisi Saat Ini: Interaksi pengguna dilakukan melalui terminal/console (Command Line Interface) yang kurang ramah bagi pengguna awam.
   b. Saran: Membangun antarmuka visual agar lebih mudah digunakan.

    1) Desktop GUI: Menggunakan library Tkinter atau PyQt.
    2) Web App: Menggunakan framework Flask atau Django.
  c. Keuntungan Arsitektur: Karena logika bisnis sudah terisolasi di Service Layer (misal: KorbanService, ObatService), Anda bisa membangun GUI/Web tanpa perlu menulis ulang logika validasi stok atau triase. UI hanya perlu memanggil method di Service yang sudah ada.
3. Sistem Autentikasi dan Otorisasi (Login)
  a. Kondisi Saat Ini: Siapa saja yang menjalankan aplikasi memiliki akses penuh sebagai "Admin" (bisa menambah, mengedit, dan menghapus data).
  b. Saran: Menambahkan fitur Login dengan pembagian hak akses (Role-Based Access Control):
    1) Admin: Mengelola data master (Bencana, Posko, User).
    2) Tenaga Medis: Hanya bisa mengakses menu Pemeriksaan dan Resep.
    3) Logistik: Hanya bisa mengelola stok Obat.
4. Fitur Pelaporan (Export Data)
  a. Kondisi Saat Ini: Output data hanya terlihat di layar console atau file log .txt.
  b. Saran: Menambahkan fitur untuk mencetak laporan resmi dalam format PDF atau Excel (.xlsx). Contoh laporan yang berguna:
    1) Laporan jumlah korban berdasarkan status triase.
    2) Laporan pemakaian obat harian untuk audit logistik.
5. Validasi Data yang Lebih Kompleks
  a. Kondisi Saat Ini: Validasi sudah ada namun terbatas (misal: cek stok negatif, cek tanggal kadaluarsa).
  b. Saran: Menambahkan validasi medis yang lebih detail, misalnya:
    1) Peringatan interaksi obat (jika korban diberi dua obat yang tidak boleh diminum bersamaan).
    2) Validasi stok minimum (memberi peringatan "Stok Menipis" jika sisa obat di bawah 10 unit).
6. Unit Testing (Pengujian Otomatis)
  a. Kondisi Saat Ini: Pengujian dilakukan secara manual dengan menjalankan skenario di main.py.
  b. Saran: Membuat script Unit Test menggunakan library unittest atau pytest. Ini berfungsi untuk memastikan setiap fungsi (misalnya fungsi tambah_stok atau buat_resep) berjalan benar secara otomatis tanpa harus menjalankan simulasi penuh dari awal.

DAFTAR PUSTAKA

Guttag, J. V. (2016). *Introduction to Computation and Programming Using Python.* (Bab 8).

Lutz, M. (2013). *Learning Python*. (Bab 8).