Implementasi Sistem Manajemen Kendaraan Berbasis PBO

Rafael Julio Suseno, Audrey Raymond Immanuel
Student Number: 5803024012, 5803024017

May 8, 2025

Contents

1 Introduction

Laporan ini menyajikan implementasi sistem manajemen kendaraan menggunakan konsep pemrograman berorientasi objek (Object-Oriented Programming atau OOP) dalam bahasa pemrograman Python. Sistem ini dirancang untuk mengelola berbagai jenis kendaraan seperti mobil, motor, dan truk, dengan mempertimbangkan karakteristik dan fungsi unik dari masing-masing jenis kendaraan. Proyek ini bertujuan untuk mendemonstrasikan penerapan prinsip-prinsip OOP seperti inheritance (pewarisan), polymorphism (polimorfisme), encapsulation (enkapsulasi), dan abstraction (abstraksi) dalam konteks manajemen kendaraan.

1.1 Project Scope

Ruang lingkup proyek ini mencakup:

- Pengembangan kelas dasar *Kendaraan* yang mendefinisikan atribut dan metode umum
- Implementasi kelas turunan untuk tipe kendaraan spesifik: *Mobil*, *Motor*, dan *Truk*
- Pengembangan sistem manajemen untuk menambah dan melihat koleksi kendaraan
- Antarmuka pengguna berbasis konsol untuk interaksi dengan sistem

Proyek ini tidak mencakup penyimpanan data persisten (database), antarmuka grafis, atau fitur lanjutan seperti pencarian dan filter.

1.1.1 Related Work

Sistem manajemen kendaraan telah banyak dikembangkan dalam berbagai konteks seperti perusahaan rental kendaraan, manajemen armada, dan inventaris dealer kendaraan. Dibandingkan dengan sistem yang ada, implementasi ini berfokus pada aspek edukasi dan demonstrasi konsep OOP daripada fungsionalitas bisnis yang komprehensif. Beberapa karya terkait dalam bidang ini termasuk sistem manajemen armada perusahaan, aplikasi peminjaman kendaraan, dan sistem inventaris dealer.

1.2 Objectives

Tujuan utama dari proyek ini adalah:

1. Mendemonstrasikan penerapan konsep OOP dalam pengembangan sistem informasi
2. Mengimplementasikan hierarki kelas dengan inheritance untuk mengelola berbagai jenis kendaraan
3. Mengembangkan sistem manajemen kendaraan yang modular dan extensible
4. Memfasilitasi penggunaan polimorfisme melalui override metode untuk perilaku khusus kendaraan

2 Konsep Dasar OOP

Pemrograman berorientasi objek (OOP) adalah paradigma pemrograman yang berfokus pada objek dan data daripada logika dan fungsi. Empat prinsip utama OOP yang diimplementasikan dalam proyek ini adalah:

2.1 Encapsulation (Enkapsulasi)

Enkapsulasi merujuk pada pembungkusan data dan metode yang memanipulasi data tersebut menjadi satu unit. Dalam proyek ini, setiap kelas kendaraan mengenkapsulasi atribut dan perilaku spesifiknya.

2.2 Inheritance (Pewarisan)

Pewarisan memungkinkan kelas untuk mewarisi atribut dan metode dari kelas lain. Dalam sistem ini, kelas *Mobil*, *Motor*, dan *Truk* mewarisi dari kelas dasar *Kendaraan*.

2.3 Polymorphism (Polimorfisme)

Polimorfisme memungkinkan metode melakukan aksi berbeda berdasarkan objek yang memanggil metode tersebut. Implementasi metode *bergerak()* dan *info()* yang berbeda di setiap kelas turunan adalah contoh polimorfisme.

2.4 Abstraction (Abstraksi)

Abstraksi berarti menyembunyikan detail implementasi yang kompleks dan hanya menampilkan fungsionalitas esensial. Kelas dasar *Kendaraan* berfungsi sebagai abstraksi yang mendefinisikan interface umum untuk semua jenis kendaraan.

3 Arsitektur Sistem

Sistem manajemen kendaraan dibangun dengan arsitektur berorientasi objek yang terdiri dari beberapa komponen utama.

3.1 Class Diagram

3.2 Struktur Komponen

Sistem ini terdiri dari empat kelas utama dan beberapa fungsi pembantu:

1. **Kelas Kendaraan**: Kelas dasar (parent class) yang mendefinisikan atribut dan metode umum untuk semua jenis kendaraan.

2. **Kelas Mobil**: Kelas turunan dari Kendaraan dengan atribut dan metode khusus untuk mobil.

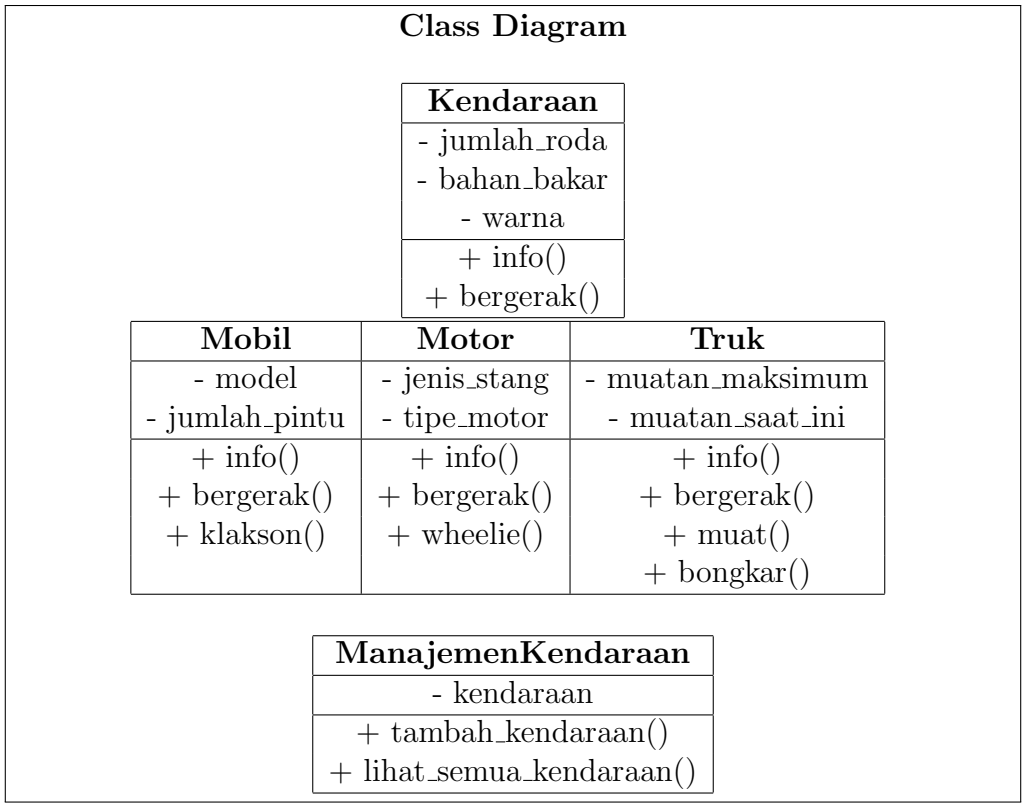

Figure 1: Diagram Kelas Sistem Manajemen Kendaraan

3. **Kelas Motor**: Kelas turunan dari Kendaraan dengan atribut dan metode khusus untuk motor.

4. **Kelas Truk**: Kelas turunan dari Kendaraan dengan atribut dan metode khusus untuk truk, termasuk manajemen muatan.

5. **Kelas ManajemenKendaraan**: Kelas untuk mengelola koleksi kendaraan.

6. **Fungsi Pembantu**: Fungsi untuk membuat instance kendaraan dan menjalankan program utama.

4 Flowchart Program Manajemen Kendaraan dan Struktur Program

4.1 Program dimulai

Dengan inisialisasi sistem manajemen kendaraan dan menampilkan menu utama.

4.2 Menu Utama

- Tambah Mobil

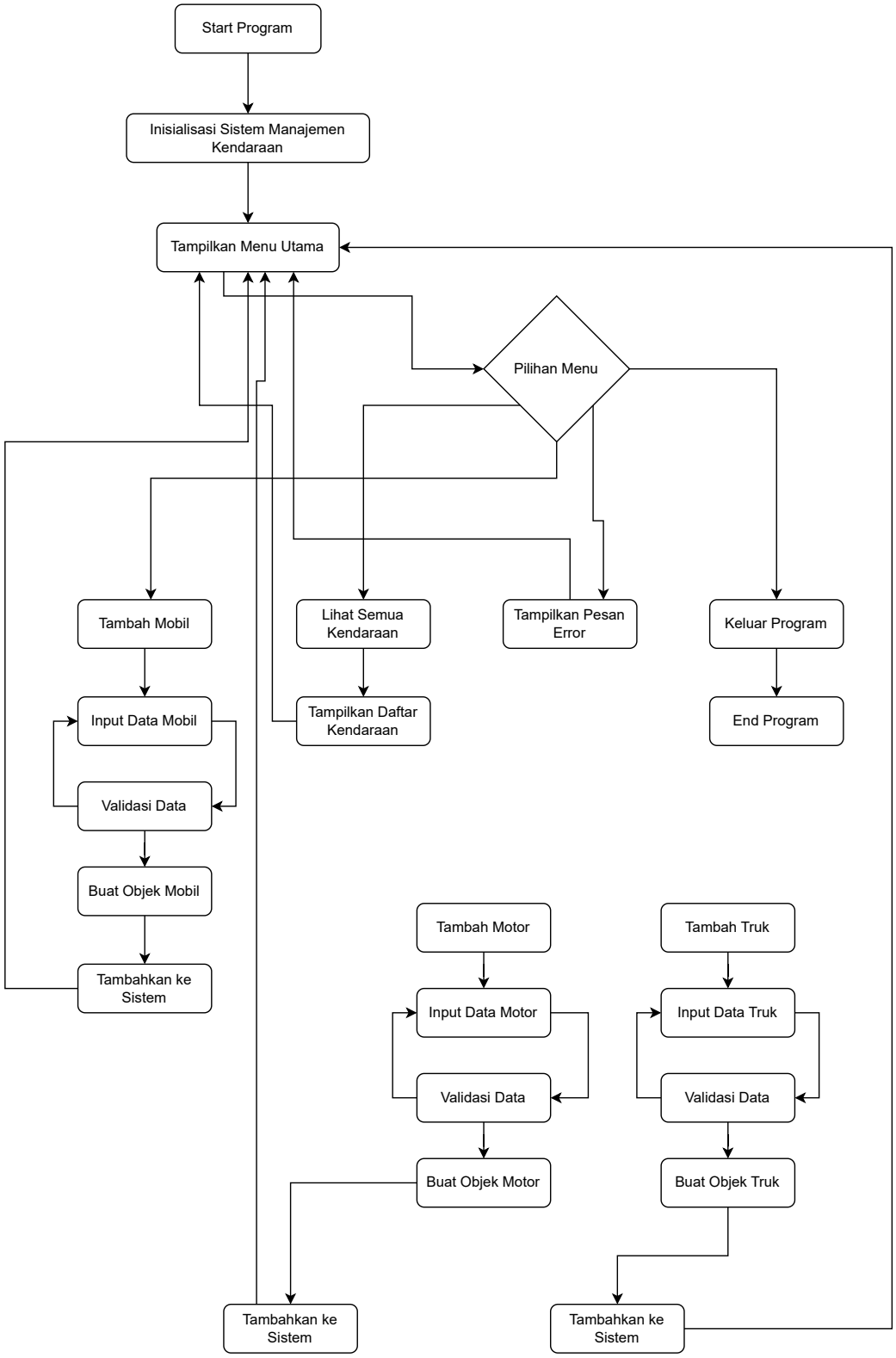

Figure 2: Flowchart Program

- Tambah Motor
- Tambah Truk
- Lihat Semua Kendaraan
- Keluar Program

4.3 Proses Tambah Kendaraan

Untuk setiap jenis kendaraan(Mobil, Motor, Truk), program akan:

- Meminta input data
- Memvalidasi data
- Membuat objek kendaraan jika data valid
- Menambahkan objek ke sistem manajemen
- Kembali ke menu utama

4.4 Lihat Semua Kendaraan

- Menampilkan daftar semua kendaraan yang sudah terdaftar
- Kembali ke menu utama

4.5 Struktur Kelas

Di bagian bawah flowchart, ditunjukkan hubungan inheritance(pewarisan) dimana:

- Kelas Kendaraan menjadi parent class
- Kelas Mobil, Motor, dan Truk mewarisi dari kelas Kendaraan

5 Implementasi

Bagian ini menjelaskan implementasi sistem manajemen kendaraan dalam bahasa Python.

5.1 Kelas Dasar: Kendaraan

Kelas *Kendaraan* merupakan kelas dasar yang mendefinisikan atribut dan metode umum untuk semua jenis kendaraan.

```
class Kendaraan:
    """
    Kelas dasar untuk semua jenis kendaraan.
    """
    def __init__(self, jumlah_roda, bahan_bakar, warna):
        self.jumlah_roda = jumlah_roda
        self.bahan_bakar = bahan_bakar
        self.warna = warna

    def info(self):
        return f"Kendaraan dengan {self.jumlah_roda} roda, berbahan bakar
   {self.bahan_bakar}, berwarna {self.warna}"

    def bergerak(self):
        return "Kendaraan bergerak"
```

Listing 1: Implementasi Kelas Kendaraan

5.2 Kelas Turunan

5.2.1 Kelas Mobil

Kelas *Mobil* mewarisi atribut dan metode dari kelas *Kendaraan* dan menambahkan fungsionalitas khusus untuk mobil.

```
class Mobil(Kendaraan):
    """
    Kelas untuk jenis kendaraan Mobil.
    """
    def __init__(self, bahan_bakar, warna, model, jumlah_pintu):
        super().__init__(4, bahan_bakar, warna)
        self.model = model
        self.jumlah_pintu = jumlah_pintu

    def info(self):
        return f"{super().info()}. Model: {self.model}, Jumlah pintu: {
   self.jumlah_pintu}"

    def bergerak(self):
        return "Mobil melaju di jalan raya"

    def klakson(self):
        return "Tin Tin!"
```

Listing 2: Implementasi Kelas Mobil

5.2.2 Kelas Motor

Kelas *Motor* juga mewarisi dari kelas *Kendaraan* dan mengimplementasikan perilaku khusus motor.

```
class Motor(Kendaraan):
    """
```

```
    Kelas untuk jenis kendaraan Motor.
    """
    def __init__(self, bahan_bakar, warna, jenis_stang, tipe_motor):
        super().__init__(2, bahan_bakar, warna)
        self.jenis_stang = jenis_stang
        self.tipe_motor = tipe_motor

    def info(self):
        return f"{super().info()}. Tipe: {self.tipe_motor}, Jenis stang: {
   self.jenis_stang}"

    def bergerak(self):
        return "Motor melaju dengan lincah"

    def wheelie(self):
        return "Motor melakukan wheelie!"
```

Listing 3: Implementasi Kelas Motor

5.2.3 Kelas Truk

Kelas *Truk* mewarisi dari kelas *Kendaraan* dan menambahkan fungsionalitas manajemen muatan.

```
class Truk(Kendaraan):
    """
    Kelas untuk jenis kendaraan Truk.
    """
    def __init__(self, bahan_bakar, warna, muatan_maksimum, jumlah_roda=6)
   :
        if jumlah_roda < 4:
            raise ValueError("Truk harus memiliki minimal 4 roda!")
        super().__init__(jumlah_roda, bahan_bakar, warna)
        self.muatan_maksimum = muatan_maksimum
        self.muatan_saat_ini = 0

    def info(self):
        return f"{super().info()}. Muatan maksimum: {self.muatan_maksimum}
    kg, Muatan saat ini: {self.muatan_saat_ini} kg"

    def bergerak(self):
        return "Truk berjalan dengan berat"

    def muat(self, berat):
        if berat < 0:
            return "Berat muatan tidak boleh negatif!"
        if self.muatan_saat_ini + berat <= self.muatan_maksimum:
            self.muatan_saat_ini += berat
            return f"Berhasil memuat {berat} kg. Total muatan sekarang: {
   self.muatan_saat_ini} kg"
        else:
            return f"Gagal memuat! Muatan akan melebihi kapasitas maksimum
    {self.muatan_maksimum} kg"
```

```
    def bongkar(self, berat):
        if berat < 0:
            return "Berat bongkar tidak boleh negatif!"
        if berat <= self.muatan_saat_ini:
            self.muatan_saat_ini -= berat
            return f"Berhasil membongkar {berat} kg. Total muatan sekarang
   : {self.muatan_saat_ini} kg"
        else:
            return f"Gagal membongkar! Tidak cukup muatan (muatan saat ini
   : {self.muatan_saat_ini} kg)"
```

Listing 4: Implementasi Kelas Truk

5.3 Sistem Manajemen

Kelas *ManajemenKendaraan* mengelola koleksi kendaraan dan menyediakan fungsionalitas untuk menambah dan melihat kendaraan.

```
class ManajemenKendaraan:
    """
    Kelas untuk mengelola koleksi kendaraan
    """
    def __init__(self):
        self.kendaraan = []

    def tambah_kendaraan(self, kendaraan):
        self.kendaraan.append(kendaraan)
        return f"{type(kendaraan).__name__} berhasil ditambahkan ke sistem
   "

    def lihat_semua_kendaraan(self):
        if not self.kendaraan:
            return "Belum ada kendaraan yang terdaftar"

        hasil = "=== Daftar Kendaraan ===\n"
        for idx, kendaraan in enumerate(self.kendaraan, 1):
            hasil += f"{idx}. {type(kendaraan).__name__}: {kendaraan.info
   ()}\n"
        return hasil
```

Listing 5: Implementasi Kelas ManajemenKendaraan

5.4 Fungsi Pembantu dan Program Utama

Program ini menyediakan fungsi-fungsi pembantu untuk membuat objek kendaraan dan fungsi main untuk menjalankan program.

```
def buat_mobil():
    print("\n=== Tambah Mobil Baru ===")
    bahan_bakar = input("Masukkan jenis bahan bakar: ")
    warna = input("Masukkan warna: ")
    model = input("Masukkan model mobil: ")
```

```
    while True:
        try:
            jumlah_pintu = int(input("Masukkan jumlah pintu: "))
            return Mobil(bahan_bakar, warna, model, jumlah_pintu)
        except ValueError:
            print("Error: Masukkan angka untuk jumlah pintu!")


def buat_motor():
    print("\n=== Tambah Motor Baru ===")
    bahan_bakar = input("Masukkan jenis bahan bakar: ")
    warna = input("Masukkan warna: ")
    jenis_stang = input("Masukkan jenis stang (standar/clip-on/dll): ")
    tipe_motor = input("Masukkan tipe motor (sport/bebek/matic/dll): ")

    return Motor(bahan_bakar, warna, jenis_stang, tipe_motor)


def buat_truk():
    print("\n=== Tambah Truk Baru ===")
    bahan_bakar = input("Masukkan jenis bahan bakar: ")
    warna = input("Masukkan warna: ")

    while True:
        try:
            muatan_maksimum = float(input("Masukkan muatan maksimum (kg):
   "))
            jumlah_roda = int(input("Masukkan jumlah roda (minimal 4): "))
            return Truk(bahan_bakar, warna, muatan_maksimum, jumlah_roda)
        except ValueError:
            print("Error: Masukkan angka yang valid!")


def main():
    sistem = ManajemenKendaraan()

    while True:
        print("\n=== MANAJEMEN KENDARAAN ===")
        print("1. Tambah Mobil")
        print("2. Tambah Motor")
        print("3. Tambah Truk")
        print("4. Lihat Semua Kendaraan")
        print("5. Keluar")

        pilihan = input("Pilih menu (1-5): ")

        if pilihan == "1":
            mobil = buat_mobil()
            print(sistem.tambah_kendaraan(mobil))

        elif pilihan == "2":
            motor = buat_motor()
            print(sistem.tambah_kendaraan(motor))
```

```
        elif pilihan == "3":
            truk = buat_truk()
            print(sistem.tambah_kendaraan(truk))

        elif pilihan == "4":
            print(sistem.lihat_semua_kendaraan())

        elif pilihan == "5":
            print("Terima kasih telah menggunakan program!")
            break

        else:
            print("Pilihan tidak valid!")


if __name__ == "__main__":
    main()
```

Listing 6: Implementasi Fungsi Pembantu dan Program Utama

6 Fitur Sistem

Sistem manajemen kendaraan memiliki fitur-fitur berikut:

6.1 Input Validasi

Sistem mengimplementasikan validasi input untuk memastikan data yang dimasukkan valid:

- Untuk jumlah pintu mobil, hanya menerima input berupa angka
- Untuk muatan maksimum truk, memverifikasi bahwa nilai yang dimasukkan berupa angka
- Untuk jumlah roda truk, memastikan nilai minimal 4
- Untuk operasi muat dan bongkar, memverifikasi bahwa berat tidak negatif dan sesuai dengan batasan muatan

6.2 Penanganan Error

Sistem menangani berbagai jenis error:

- *ValueError* untuk input yang tidak valid
- Pesan error yang informatif untuk operasi yang tidak berhasil
- Validasi data untuk mencegah pembentukan objek yang tidak valid

6.3 Menu Interaktif

Program menyediakan menu interaktif yang memungkinkan pengguna untuk:

- Menambahkan mobil baru ke sistem
- Menambahkan motor baru ke sistem
- Menambahkan truk baru ke sistem
- Melihat semua kendaraan yang terdaftar
- Keluar dari program

7 Analisis dan Hasil

7.1 Analisis Performa

Sistem manajemen kendaraan ini memiliki performa yang efisien untuk operasi dasar:

Table 1: Analisis Performa Operasi

Operasi	**Kompleksitas Waktu**	**Kompleksitas Ruang**
Tambah Kendaraan	O(1)	O(1)
Lihat Semua Kendaraan	O(n)	O(n)
Muat/Bongkar Muatan	O(1)	O(1)

7.2 Penerapan Prinsip OOP

Implementasi sistem ini berhasil menerapkan prinsip-prinsip OOP:

Table 2: Evaluasi Penerapan Prinsip OOP

Prinsip	**Implementasi**
Encapsulation	Atribut dan metode dikelompokkan dalam kelas yang sesuai
Inheritance	Kelas *Mobil*, *Motor*, dan *Truk* mewarisi dari kelas *Kendaraan*
Polymorphism	Metode *info()* dan *bergerak()* diimplementasikan secara berbeda di setiap kelas
Abstraction	Kelas *Kendaraan* menyediakan interface umum tanpa implementasi detail

7.3 Fleksibilitas dan Ekstensibilitas

Sistem ini dirancang dengan mempertimbangkan fleksibilitas dan ekstensibilitas:

- Hierarki kelas memudahkan penambahan jenis kendaraan baru
- Penggunaan polimorfisme memungkinkan perilaku khusus untuk setiap kelas
- Sistem manajemen terpisah dari kelas kendaraan, memungkinkan pengembangan fitur manajemen tanpa mengubah kelas kendaraan

8 Kesimpulan dan Pengembangan Masa Depan

8.1 Kesimpulan

Proyek ini berhasil mengimplementasikan sistem manajemen kendaraan menggunakan konsep pemrograman berorientasi objek dalam bahasa Python. Sistem ini mendemonstrasikan penerapan prinsip-prinsip OOP seperti encapsulation, inheritance, polymorphism, dan abstraction. Implementasi ini berhasil mencapai tujuan untuk:

- Menyediakan hierarki kelas yang sesuai untuk berbagai jenis kendaraan
- Mengimplementasikan fungsionalitas khusus untuk setiap jenis kendaraan
- Menyediakan sistem manajemen untuk mengelola koleksi kendaraan
- Menyediakan antarmuka pengguna yang sederhana dan intuitif

8.2 Pengembangan Masa Depan

Untuk pengembangan masa depan, beberapa fitur dan peningkatan dapat dipertimbangkan:

1. **Penyimpanan Data Persisten**: Implementasi database untuk menyimpan data kendaraan secara permanen
2. **Antarmuka Grafis**: Pengembangan GUI untuk meningkatkan pengalaman pengguna
3. **Fitur Pencarian dan Filter**: Menambahkan kemampuan untuk mencari dan memfilter kendaraan berdasarkan kriteria tertentu
4. **Penambahan Jenis Kendaraan**: Memperluas hierarki kelas dengan jenis kendaraan tambahan seperti sepeda, bus, dan kereta
5. **Manajemen Lanjutan**: Penambahan fitur seperti penghitungan konsumsi bahan bakar, jadwal perawatan, dan pencatatan perjalanan

9 References

References

[1] Python Software Foundation. *Python Language Reference, version 3.9*. Available at `http://www.python.org`.

[2] Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

[3] Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall.