Deskripsi Project Aplikasi Rental Mobil

Nama Proyek: Aplikasi Rental Mobil

Deskripsi Singkat: Aplikasi ini bertujuan untuk memudahkan proses penyewaan mobil, baik bagi penyewa maupun bagi pemilik mobil. Aplikasi ini akan memiliki fitur-fitur utama seperti pencarian mobil, pemesanan, pengelolaan data pengguna, serta pengelolaan data mobil yang tersedia untuk disewa. Data akan disimpan dalam database yang aman dan terstruktur.

Fitur Utama:

1. **Registrasi dan Login Pengguna:** Pengguna (baik penyewa maupun pemilik mobil) dapat membuat akun dan masuk ke aplikasi.
2. **Pencarian Mobil:** Pengguna dapat mencari mobil berdasarkan berbagai kriteria seperti jenis mobil, harga, lokasi, dan ketersediaan.
3. **Pemesanan Mobil:** Penyewa dapat melakukan pemesanan mobil yang tersedia dan melakukan pembayaran secara online.
4. **Pengelolaan Data Mobil:** Pemilik mobil dapat menambah, menghapus, atau mengubah informasi mobil yang tersedia untuk disewa.
5. **Pengelolaan Transaksi:** Aplikasi akan mencatat semua transaksi yang terjadi antara penyewa dan pemilik mobil.

Arsitektur dan Implementasi OOP

1. Encapsulation (Enkapsulasi)

Encapsulation adalah konsep dimana data (variabel) dan metode (fungsi) yang beroperasi pada data tersebut disatukan dalam satu unit, yaitu class. Ini menjaga data dari akses langsung dari luar class dan hanya bisa diakses melalui metode yang disediakan.

Implementasi:

```
public class Mobil {
    // Access modifiers: private digunakan untuk
data members
    private String platNomor;
    private String merk;
    private String model;
    private double hargaSewaPerHari;
    private boolean tersedia;

    // Getter dan Setter sebagai metode akses
(encapsulation)
    public String getPlatNomor() {
        return platNomor;
    }

    public void setPlatNomor(String platNomor) {
        this.platNomor = platNomor;
    }

    public String getMerk() {
        return merk;
    }

    public void setMerk(String merk) {
        this.merk = merk;
    }

    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        this.model = model;
```

```
    }

    public double getHargaSewaPerHari() {
        return hargaSewaPerHari;
    }

    public void setHargaSewaPerHari(double
hargaSewaPerHari) {
        this.hargaSewaPerHari = hargaSewaPerHari;
    }

    public boolean isTersedia() {
        return tersedia;
    }

    public void setTersedia(boolean tersedia) {
        this.tersedia = tersedia;
    }
}
```

2. Inheritance (Pewarisan)

Inheritance memungkinkan sebuah class untuk mewarisi properti dan metode dari class lain. Hal ini membantu dalam mengurangi kode yang berulang dan memfasilitasi pengelompokan objek yang serupa.

Implementasi:

```
public class Pengguna {
    protected String nama;
    protected String email;
    protected String password;
```

```
    // Konstruktor, getter dan setter
    public Pengguna(String nama, String email,
String password) {
        this.nama = nama;
        this.email = email;
        this.password = password;
    }

    // Metode lain yang relevan
}

public class Penyewa extends Pengguna {
    private String alamat;

    public Penyewa(String nama, String email,
String password, String alamat) {
        super(nama, email, password);
        this.alamat = alamat;
    }

    // Getter dan Setter untuk alamat
}
```

3. Polymorphism (Polimorfisme)

Polymorphism memungkinkan metode yang sama untuk digunakan oleh objek-objek dari class yang berbeda dengan cara yang berbeda. Hal ini meningkatkan fleksibilitas dan kemampuan untuk memperluas program.

Implementasi:

```
public class Transaksi {
    public void prosesTransaksi(Pengguna pengguna)
{
```

```
        // Logika umum untuk memproses transaksi
    }
}
public class PenyewaTransaksi extends Transaksi {
    @Override
    public void prosesTransaksi(Pengguna pengguna)
{
        // Logika khusus untuk transaksi penyewa
        System.out.println("Proses transaksi untuk
penyewa: " + pengguna.nama);
    }
}
public class PemilikTransaksi extends Transaksi {
    @Override
    public void prosesTransaksi(Pengguna pengguna)
{
        // Logika khusus untuk transaksi pemilik
        System.out.println("Proses transaksi untuk
pemilik: " + pengguna.nama);
    }
}
```

4. Access Modifiers (Modifier Akses)

Access modifiers mengontrol aksesibilitas data members dan metode di dalam class. Modifier ini membantu menjaga data dan metode agar tidak dapat diakses secara tidak sah dari luar class.

Implementasi:

- **private:** Hanya bisa diakses di dalam class itu sendiri.
- **protected:** Bisa diakses di dalam class itu sendiri, subclass, dan class dalam paket yang sama.
- **public:** Bisa diakses dari mana saja.

Mengapa Menggunakan Teori OOP?

1. **Encapsulation:** Menjaga integritas data dan memastikan bahwa data hanya bisa diubah melalui metode yang tepat, mengurangi kemungkinan bug.
2. **Inheritance:** Mengurangi redundansi kode dengan memungkinkan class untuk mewarisi properti dan metode dari class lain, mempermudah pemeliharaan dan pengembangan.
3. **Polymorphism:** Memungkinkan penggunaan satu interface untuk berbagai jenis class, meningkatkan fleksibilitas dan skalabilitas aplikasi.
4. **Access Modifiers:** Menyediakan kontrol akses yang baik terhadap data dan metode, menjaga keamanan dan modularitas kode.

Dengan menggunakan teori OOP, aplikasi rental mobil ini dapat dibangun dengan struktur yang lebih terorganisir, kode yang dapat dikelola dengan lebih mudah, serta kemampuan untuk dikembangkan lebih lanjut di masa depan.